UiA University of Agder

Master

Using AI and Robotics for EV battery cable detection.

Development and implementation of end-to-end model-free 3D instance segmentation for industrial purposes.

HENRIK BRÅDLAND[1] & IVER SØRENSEN[2]

SUPERVISORS

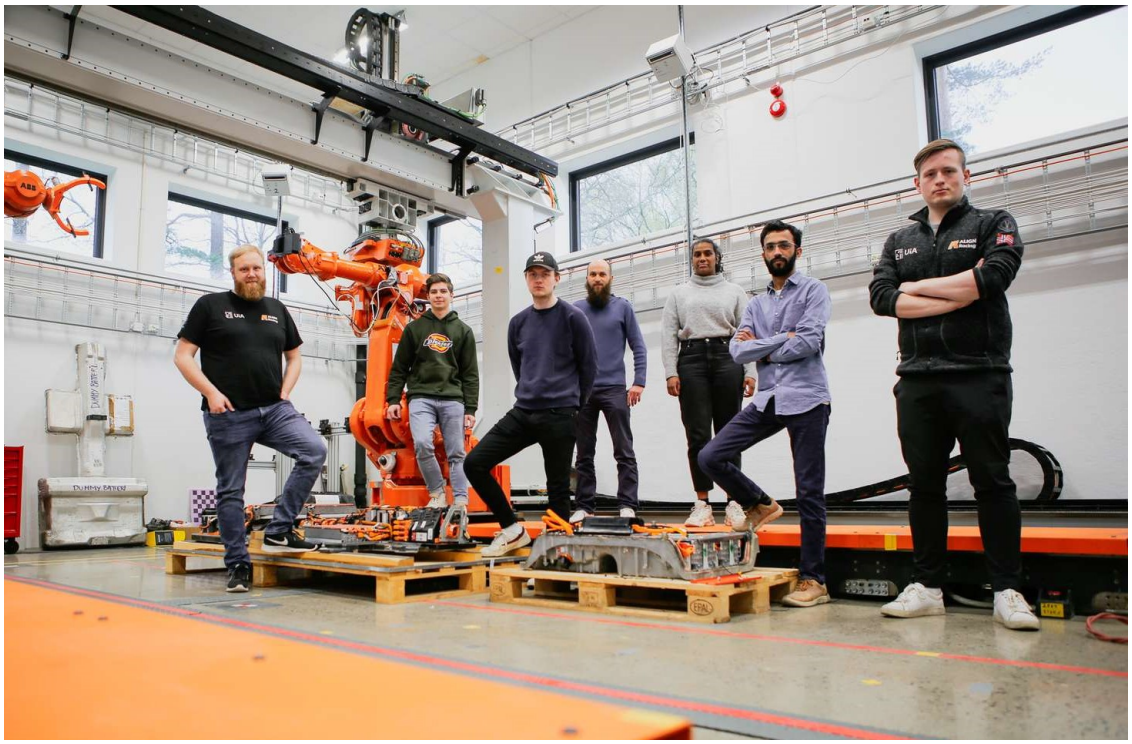
Linga Reddy Cenkeramaddi[1] and Martin Marie Hubert Choux[2]

University of Agder, 2021
Faculty of Engineering and Science
[1]Department of Information and Communication Technology
[2]Department of Engineering Sciences

Acknowledgements

We would like to thank supervisors Martin Marie Hubert Choux and Linga Reddy Cenkeramaddi for their good guidance and support throughout the thesis. In addition, we would like to give thanks to the IT department at UiA for helping with setting up the hardware.

We would also like to thank the rest of the team working on the LIBRES project for being cooperative, and for providing good reflections and for participating in discussions on how the subsystems should be stitched together. This allowed for a more comprehensive design.



The LIBRES team at UiA.

Abstract

This thesis describes a novel method for capturing point clouds and segmenting instances of cabling found on electric vehicle battery packs. The use of cutting-edge perception algorithm architectures, such as graph-based and voxel-based convolution, in industrial autonomous lithium-ion battery pack disassembly is being investigated. The thesis focuses on the challenge of getting a desirable representation of any battery pack using an ABB robot in conjunction with a high-end structured light camera, with "end-to-end" and "model-free" as design constraints. The thesis employs self-captured datasets comprised of several battery packs that have been captured and labeled. Following that, the datasets are used to create a perception system..

This thesis recommends using HDR functionality in an industrial application to capture the full dynamic range of the battery packs. To adequately depict 3D features, a three-point-of-view capture sequence is deemed necessary. A general capture process for an entire battery pack is also presented, but a next-best-scan algorithm is likely required to ensure a "close to complete" representation. Graph-based deep-learning algorithms have been shown to be capable of being scaled up to $50,000$ inputs while still exhibiting strong performance in terms of accuracy and processing time. The results show that an instance segmenting system can be implemented in less than two seconds. Using off-the-shelf hardware, demonstrate that a 3D perception system is industrially viable and competitive with a 2D perception system.

Contents

Chapter 1

Introduction

1.1 Background

A shift from combustion vehicles to electric vehicles seems to be imminent, many would even argue that it has already happened. In 2020 battery-driven vehicles had a share of 54.3% of the new vehicles sold in Norway, with chargeable hybrids and hybrids getting second place in market percentage with 29% [1]. By the end of 2020, EV's made up a total of 12% of Norway's car fleet, with an increase of 30% from 2019 to 2020 [2].

This increase in EV's market share is likely due to EV's becoming cheaper and performing better. But for many, the idea of an environmentally friendly, emission-free electric car is a deciding factor. However, an EV will likely never really be emission-free. Both the production and sourcing of the materials used in an EV can be both environmentally challenging and a humanitarian problem. With some resources requiring large mining operations, in sometimes unstable nations with problematic working conditions [3]. Therefore, when an EV reaches its end of the life cycle, as much as possible of the vehicle should be recycled.

The LIBRES project

LIBRES is a research project where the main goal is to develop a design basis for a lithium-ion battery (LIB) recycling pilot plant by 2022, with the pilot plant able to handle a commercial volume by 2024. The project group consists of several industry partners, with Hydro ASA being the project owner[4]. The University of Agder is one of the universities contributing to the project. Together with BatteriRetur AS, the university is focusing on the battery dismantling stage of the recycling process. This thesis is in association with the LIBRES project.

Why automation

By the time of writing this thesis, dismantling of lithium-ion batteries in Norway is done by hand. The process requires two people, which must be authorised electricians with high voltage experience, due to the batteries containing a large amount of residual power when delivered to dismantling. On average the process of dismantling a battery pack takes about 45 minutes [5]. With the high amount of personnel needed, and the degree of education required to work, the personnel cost of such an operation can become quite expensive. When looking at the trends of the vehicle transport market, it is clear that a higher demand for battery recycling is imminent. By reducing the need for specialised skilled labour, an autonomous LIB dismantling plant can therefore be the solution to decrease the cost and meet the demands of the future vehicle transport market.

Challenges of an autonomous system

Disassembling a LIB pack is an extensive task, with many steps. One of the reasons why this process is done manually is due to the many small parts to be removed, like wires and electronics. One way to automate the disassembly process is to hard-code the right sequence of operations to achieve the goal of dismantling the battery pack. However, this solution leaves the dismantling system sensitive to changes in the battery pack. Say for instance, the battery pack to be disassembled have been damaged to the extent that the hard-coded sequence will fail. In that case, a new sequence to handle that scenario would need to be coded. To get a disassembly system that consistently handles deviations, the number of potential scenarios that would need to be hard-coded will quickly increase to an impractical amount.

With the current number of different EV models approximately being around 100 models, the amount of work needed to hard code all the sequences would be impractical, and with the expected increase to 350 models by the year 2025 the plant would require constant follow up as new models came out. The existing EV models may also vary their battery setup over time, increasing the required amount of hard-coding [6].

Critical pilot plant features

For the disassembly plant to be economically viable it needs to cope with the large variations in battery packs. Therefore, for the pilot plant envisioned in this thesis, need to have the following features:

- **Model-free approach:** The system can not be dependent on having access to 3D models of the battery pack. If the plant is to function with minimal upkeep, the plant can not need extensive updates when a new EV model comes to market. If the plant were dependent on 3D models of the LIB packs, it would likely have problems handling damaged packs.
- **Perception system functionality:** For a model-free approach to be viable, the plant need to build a model of the battery on its own. With a point cloud-based capturing system, a model of the battery can be generated. This model will contain damage features, which would be invisible if the system was to operate on pre-made models.
- **Part segmentation system:** When point clouds are generated. The plant need an intelligent algorithm capable of identifying objects of interest in a wide array of states. A potential fit for this task would be a neural network-based approach.
- **Robotic manipulator:** With the number of complex manipulation tasks needed to dismantle a battery, a robot can provide the needed versatility without drastically increasing the need for machinery. The robot arm will likely need to have a set of interchangeable tools to complete the various tasks required.

1.2 State-of-the-art

Robotics

The current research field of robotics is focused on designing adaptable approaches to robotics. With the adaptive robots often working with incomplete information. Examples of handling incomplete information are as follows: Development of systems capable of recognising that the robot has been in a certain region before [7], and motion planning with incomplete information of the surroundings [8]. Incorporating machine learning into

robotics is also being looked at, with using reinforcement learning to complete complex manipulation task [9].
Another field of robotics that is seeing much attention is the development of systems making robotics more accessible. The development of state-of-the-art open-source projects like ROS and MoveIt helps the field by establishing a standard in the industry, streamlining the process of getting robot systems working. ROS serves as a robot communication system while MoveIt can provide a framework for the robot project to be built on [10][11]. Libraries focused on path planning and inverse kinematics, like OMPL and IKfast, allows for integration with the ROS and MoveIt system[12]. The result of this development is apparent in how robotics research can be done with a high level of complexity, due to a baseline robotic system being rapidly developed, giving the researchers more time to focus on their intended research topics.

Autonomous disassembling

Due to advances in robotics and automation, the research combining robotics and disassembly have been steadily growing. The number of research papers involving "robotics" and "disassembly" have increased by 40% from 2013 to 2018. The most popular research topic within this field being sequence planning[13].

Generally, the trend in the field of sequence planning is focus on designing algorithms to handle disassembly feasibility, path obstacles and cost optimization. A common approach for sequence planning is to try different algorithms, such as Petri nets, fuzzy logic, artificial bee and ant colony [13].

Robotic implementation is another research trend, where 6-DoF robots and collaborative setups are looked at. It includes implementation of sensory systems to identify and learn dismantling processes. The focus seems to be on learning concrete dismantling system, not necessarily a general approach to dismantling [13].

Generally, the research field in automated dismantling is focusing on sequence planning with the planning algorithms becoming more adaptable with the implementation of sensory input. But implementation of automated dismantling process seems to largely be focused on dismantling of a specific known object. There appears to be a knowledge gap in implementing model-free approaches to dismantling, something which would be essential if a system needs to handle a large variety of objects in different states.

Semantic segmentation of point cloud

Point Cloud Semantic Segmentation (PCSS), i.e. the task of assigning a semantic property to every point in a point cloud, is most often done by traditional supervised machine learning approaches or by the usage of modern deep learning [14]. Traditional machine learning refers to shallow neural networks (without any hidden layers) or non-neural network based methods like support vector machine (SVM) [15], random forest [16], boosting algorithms (like AdaBoost [17]) and maximum likelihood classifiers [18].

For most recent studies (From 2018 until today) the preferred method for PCSS, regardless of the application, is some variety of deep learning [14]. 3-dimensional convolutional neural networks (3D-CNN) in combination with octrees and voxel-grids, show good preference on various benchmarks (OctNet [19], O-CNN [20], and VV-NET [21]), but are computationally heavy due to the size of the layers and the number of empty voxels. Multi view-based approaches, where the point cloud is projected down to several 2D-representations from different views, to then be processed with 2D-CNN, have shown

good performance on simpler tasks. Nevertheless, it comes short when the shapes are complicated due to the loss in geometric structure as the 2D-projections are approximations of the 3D geometry. Directly processing the point clouds is based in majority on the PointNet [22] where MLP (Multi-Layer Perceptron) is used to approximate per-point local features for each point, that are later classified by another MLP. The most promising methods are GNNs (Graph Neural Networks) like SPG [23], DGCNN [24] and RGCNN [25], which show excellent performance [14].

Most of the previous work done on PCSS is directed towards LiDAR-based and image-derived point clouds of outdoor environments, or indoors RGB-D based data. The outdoor-directed work is mainly based towards segmentation and classification of terrain from ALS (Airborn LiDAR Scanning) and image-derived data, and work on segmentation and classification of surroundings related to self-driven cars. The work related to indoors RGB-D based data is mostly related to navigation and collision avoidance for small autonomous robots [14]. Most publicly accessible data-sets and benchmarks are therefore ether urban outdoor landscapes with labelled houses, trees and other urban elements (Semantic3d.net [26], Paris-Lille-3D [27], KITTI [28]) or indoor environments labelled mostly with furniture (S3DIS [29], ScanNet [30]). At the time of writing, there are no known data-sets for disassembling that are applicable in this context. Therefor, for the task of disassembling a EV battery pack, a data-set needs to be generated. On the other side, the most relevant of the established benchmarks is ModelNet40 [31] that contains everyday objects with labelled segments.

Most work on semantic segmentation of objects is done on small point clouds, commonly in the range of $2000 - 4000$ points per point cloud [14], [24], [31], [32]. With modern sensors, the resolution of real-world data gets vastly larger, and can easily go over one million points [33]. There are few papers that tackle this problem by for instance random sampling in order to fit the model [34]. Nevertheless, most work related to single object segmentation is done on soft data, thus creating a gap between theory and industrial implementation. There are some exceptions like Mask-MCNet [32], but there is still work to be done in order to connect the models developed on soft data to be implemented in a real-world environment.

1.3 Research questions

The field of end-to-end solutions for 3D-scanning and instance segmentation of real world objects are not well understood yet. This thesis wants to discover and look into fundamental aspects of end-to-end instance segmentation of industrial battery pack dismantling solutions, and in the work answer the following research questions:

- The construction of a good digital representation of the real world object is essential to getting a well functioning end-to-end system. This thesis therefor intends to look into how this can be accomplished in an industrial setting. With a main focus on how to choose and set up hardware so that a digital representation reflects the object of interest in a good and consistent way. While taking into consideration the variety of environmental conditions systems can encounter.
- There are several promising algorithms to be used in instance segmentation of point clouds, but not on entangled point clouds. This thesis will therefor look into what kind of algorithms are most stubble for this kind of application.
- Time is an important aspect of any industrial application. The thesis therefor will look into how the end-to-end process can be performed in a time efficient way while maintaining good performance.

Chapter 2

Background theory

This chapter is intended to give the required background information needed to comprehend the choices done in the thesis. It also intends to set definitions and lay out the mathematical notation used later in the thesis. Since the thesis is multidisciplinary it is assumed that the reader might not be up to date on all background information outside his or her primary field. The theory chapter therefor includes information that otherwise would be seen as common knowledge.

2.1 Robot control

To move a robot to complete a task involves many different levels of complexity. One of the more fundamental levels that need to be designed is the actual robot arm structure, calculating how much the robot can lift compared to the weight. And if a moving robot is needed, the development of the actuation system can also be a field one can dedicate a lot of time to. The field of robotics is filled with these rabbit holes, leading to their field of research. For a researcher then to design a robotic system from nothing would be counter productive.

Over the years of robotics research, as the state-of-the-art has gotten more and more complex, tools for streamlining robotics development have steadily been expanded. Giving the field the ability to relatively quickly get up to the level for state-of-the-art research. However, it is still important to understand the tools that are used to reach the required level.

2.1.1 Spatial transformation

Spatial transformations are the basis of how 3D movement is handled. The movement can be split into translation and rotation, equating to six degrees of freedom. Where half of them represents the three-dimensional axis (XYZ) and the rest represents the rotation around the three-axis often called pitch, roll and yaw.

Translation Often represented as T translation is the simplest to represent in 3D-space as vector addition product, for later purposes showed in 4×4 matrix form. Where the base translation is represented by the vector $(t_x\, t_y\, t_z)$ and the pint to be translated $(v_x\, v_y\, v_z)$, as seen in equation 2.1 [35].

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x + t_x \\ v_y + t_y \\ v_z + t_z \\ 1 \end{bmatrix} \tag{2.1}$$

Rotation A rotation matrix is a matrix which only rotates a point around the origin, meaning the length of the vector defined by the point is not extended or shortened. Rotations can be represented as a sequence of Euler angles. Euler angles is in short rotations around the XYZ axis often denoted as $R_x(\alpha)$, $R_y(\beta)$, $R_z(\gamma)$ or yaw, pitch and roll, where α, β and γ is their respective angle of rotation. The set of rotations can be represented with the following 4×4 matrices[35]:

$$
\begin{aligned}
R &= R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha) \\
&= \begin{bmatrix} cos\gamma & -sin\gamma & 0 & 0 \\ sin\gamma & cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{2.2}
$$

Transformation Transformation is the combination of a rotation and a translation. The transformation matrix is often represented as a 4×4 matrix where the 3×3 section in the left upper corner describes the applied rotation, and the upper rightmost 1×3 represents the translation. The full transformation matrix is shown in the following equation, where H represents the full transformation matrix, R represents the rotation matrix and T represents the translation matrix[35].

$$
H = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \tag{2.3}
$$

Forward kinematics

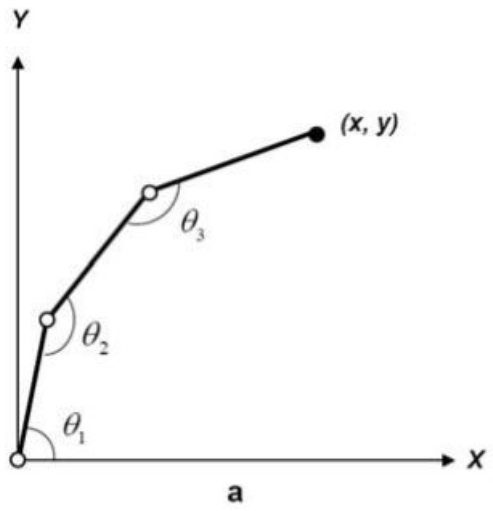


Figure 2.1: Forward kinematic chain[36].

If all angles of a robots joints are known, the position of the end effector can be calculated via forward kinematics. As can be done with the rotation matrix, one can also chain together transformation matrices, giving whats called a kinematic chain. The kinematic chain is set up such that all links, except for the base link, is a transformation in reference to the earlier link in the chain. The translation part of the transformation represents the length of the link and the rotation represents the angle of the joint. A single transformation matrix can be made by combining each transformation in order as follows[36].

$$
H_1^4 = H_3^4 H_2^3 H_1^2 \tag{2.4}
$$

2.1.2 Inverse kinematics

When designing a robotic system, how to get the end-effector to the desired position is often the focus. The problem then becomes how to organize the robot joints into positions that will give the desired position. This problem is called inverse kinematics and is when you start from the end-effector and work your way to the base joint [37].

The challenge with inverse kinematics comes due to there being, in many cases, more than one solution of how to reach a certain pose, as seen in figure 2.2. The many cases there can in theory be an infinite amount of solutions. To solve inverse kinematic problems, complex algorithms are used to find and optimise a solution.

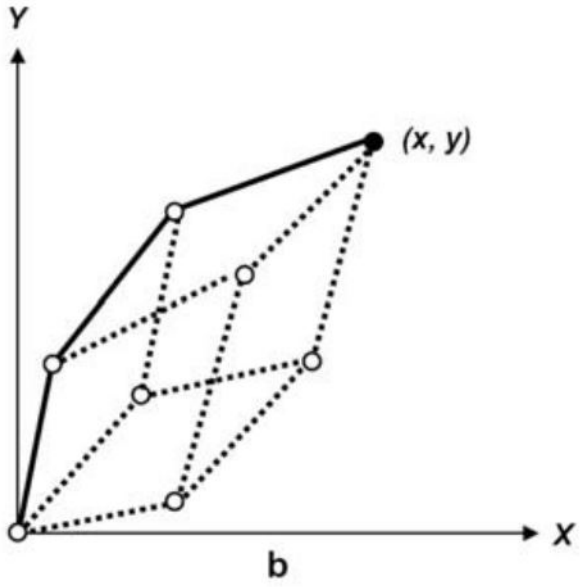


Figure 2.2: Possible solution for inverse kinematics[36].

2.1.3 Robot Operating System

Robot Operating System (ROS) is an open-source software that can be run on Unix-based platforms. ROS is built as a framework for robot communication and serves as a backbone for the robot application. It is a collection of different tools and libraries organized in a system made to streamline the development of complex robotic processes.ROS provides services like hardware abstractions, low-level device control, and a framework for machine to machine communication (M2M).

ROS organizes itself into nodes that do certain processes for the robot system to work. ROS is designed in a modular way, where different nodes run different systems. One node can run the robot arm actuators and another can manage the sensor system. The nodes communicate by a publish/subscribe system, where one node publishes its data to a certain topic and the other node subscribes to that topic. By setting up the system like this, the receiving node does not need to know about each other, only the topic they are working with.
The basic building blocks of ROS is as follows:

- **Nodes**: Processes that do computations, like drive robot motors or run the sensor system. Nodes can publish information or subscribe to information via a topic.
- **Topics**: The message transport system of ROS. The topic is a name used to identify the content of a message. A node can publish or subscribe to a topic to tap into the information. the subscriber and publisher generally do not need to be aware of each other.
- **Message**: A simple data structure that ROS communicate with. The data type can be specified from a set of basic types, like integer, floating-point or boolean. The messages are what is being broadcasted on a topic.
- **Master**: Coordinates the communication between nodes. The Master provides the name registration and lookup for the rest of the system.

- **Services**: A Request/reply system distinct from the publish/subscribe system. A service is provided under a name and the system sends a request to that name and waits for a reply. A service is presented in ROS client as a remote procedure call.

2.1.4 MoveIt

MoveIt is a tool that is built on top of ROS. Whilst ROS provides the necessary backbone for communication of the robot application, MoveIt provides a framework to build the functionality of the application by setting a basic code base for implementing key features of the robot application, such as path planning, object manipulation, inverse kinematics, higher-level robot control, collision checking and 3D perception. All this is organized into a plugin-based architecture which makes MoveIt highly customizable, giving the user the option to design their systems[11].

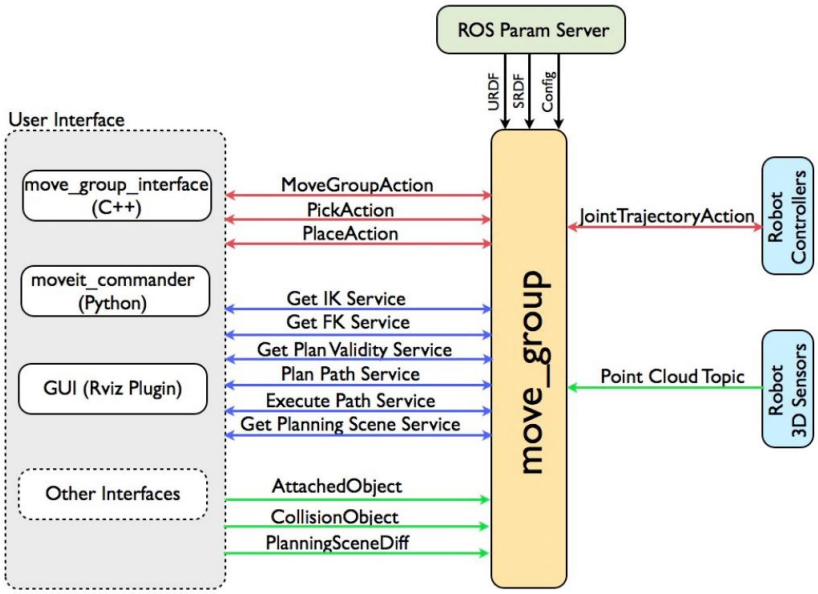


Figure 2.3: MoveIt high level architecture.

Figure 2.3 is a diagram showing the high level architecture of MoveIt. As seen in the diagram, MoveIt is organized around the *move_group* functionality. *move_group* are nodes that combine individual components to provide a set of ROS actions and services. One *move_group* can therefore be controlling the robot movement system, by fetching all features necessary to generate and execute a move order, and sending the required joint controls to the robot controller. Move groups can be called via a set of user interface systems. All variables related to the robot, like the turning limit on a joint, is stored in the ROS Param Server, in formats like URDF and SRDF. A GUI plugin called Rviz can be used to visualize the robot application, configure the sensor system and send robot commands. MoveIt also comes with C++ and Python functionality able to call the move groups[38].

2.2 Motion planning

Motion planning is the process of planning out the sequence of joint movement to move the robot from one place. Much of the same challenges of inverse kinematics reappear when motion planning. This is because when a path is generated a series of inverse kinematic problems are solved until a path from the start position to the goal position is achieved. But there is more to generating a good path than finding valid kinematic poses. Problems like collision avoidance can become a problem when the robot needs to

navigate complex geometry. There are therefore several different approaches to the path planning problem. Some methods may be guaranteed to find a solution of one that exists but requires more computation to find a solution and some may be quick but with no guarantee of finding a solution.

Some popular categories of motion planners are as follows:

- **Combinations motion planners** are planners that do not do approximations. This makes them what is called complete path planners, which means the algorithm will be able to tell you if a solution exists. Combination planners are however resource-intensive and can become impractical when more degrees of freedom is introduced [39].
- **Feedback motion planners** uses feedback control to compute the path to the goal position. Feedback systems work well in situations where the dynamics of the system affects the robots ability to follow a path. Feedback motion planners can however require complex system modelling to get a result that matches the capability of the robot [39].
- **Sampling-based motion planners** Solves the motion planning problem by sampling the configuration space of the robot until it has found a path to the goal position. Sampling-based planners excel at handling a higher degree of freedom systems, due to their ability to approximate. The robot system this thesis works with is a 7-DoF system and is therefore suited for a sampling-based planner [39]. It also the case that the planning library Open Motion Planning Library (OMPL) is a sampling-based planner library that has been implemented into MoveIt.

2.2.1 Configuration space

The configuration space of the robot is an interpretation of the combined joints of the robot. A configuration space for a 2DoF robot can be represented as a 2D field where a location represents a certain configuration of the robot joints and the two joints angles are represented in the two axes. With a higher DoF robot, the dimensions of the configuration space increase to match. The configuration space of the robot can contain configurations that are unreachable to the robot. An area of the configuration space can be unreachable for different reasons, like an obstacle, self-collision or joint restrictions, but in practice, all inaccessible areas are defined as obstacles. The physical space the robot can operate in is named the works space of the robot.

2.2.2 Sampling-based motion planners

Generally, sampling-based approaches start by generating some set of partially randomized states in the robot configurations space, then it connects these configurations with collision-free paths. Sampling-based approaches often provide what is called probability completeness. This means that if a solution exists, the probability of finding a solution approach 100% when the sampling rate approaches infinity. The strength of sampling-based systems is the ability to handle higher dimensional configuration spaces[40].

Probabilistic Road Map (PRM)

The probabilistic road map method (PRM) is an approach to sampling-based motion planning. Where a set of randomly distributed valid configurations of the robot is generated. This set of configurations is then linked in a way to create a basis for a road map throughout the configuration space of the robot. This road map is then used to find a path from the start position to the goal position [40]. An illustration of a typical path generated by

a PRM method is showed in figure 2.4, where the grey regions are inaccessible configurations, the connected nodes represent the random samples, and the orange path is the path chosen by the PRM algorithm.

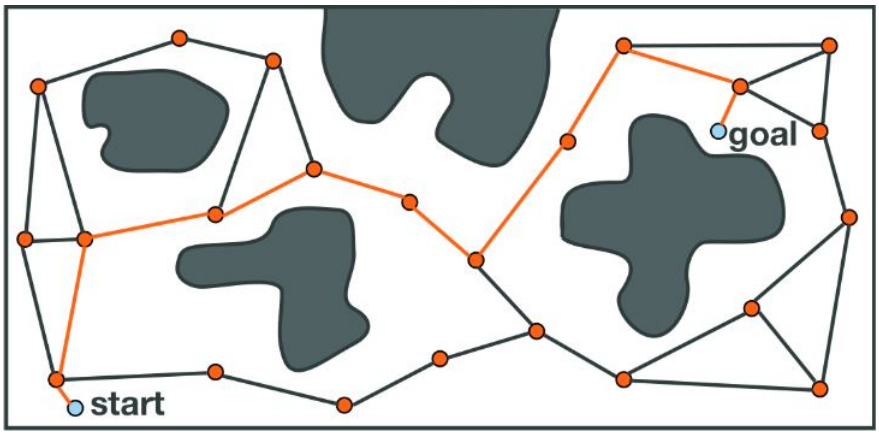


Figure 2.4: PRM-generated path[40].

The advantages of the probabilistic road map method become apparent when there is little to no change in the environment of the robot system. The robot can then reuse the road map, drastically reducing the computation requirements of finding a valid path. However, in situations where the environment is changing in between every path planning task, the road map would need to be re-generated. Since PRM samples the whole configuration space of the robot it can be outperformed by other sampling-based planners which implements a more directed sampling method.

Tree-based motion planning

The sampling done in tree-based motion planners works procedural. The "tree" in tree-based planners come from the way new samplings in the configuration space are generated dependent on a previous sample, creating a tree-like structure of connected samples, all leading back to a small set of seeded samples. With no prior knowledge of the configuration space, the Tree-based algorithms can generate a branching graph network until a path from the starting position to the goal position is reached, eliminating inaccessible configurations as the network is generated.

Figure 2.5 depicts the path generation of a Rapidly Random exploring Trees algorithm. Where the red dot is the starting position, the green rectangle is the goal position and the red rectangles are the inaccessible regions of the configuration space. The black network is the generated tree and the blue dots represents the path the algorithm found.

With a randomly generated tree, a large percent of the new nodes that gets generated will not lead to a successfully generated path to the goal state. A random tree will therefore use a considerable amount of computation, computing trees that will never lead to a solution. To account for this loss a bias towards the goal state is often applied to steer the random tree in the right direction. This bias however may cause the algorithm to be stuck in a local minimum where an obstacle would need to path-ed around. There are however solutions to this problem, like generating more trees from other states, like the goal position, working backwards to the starting position.

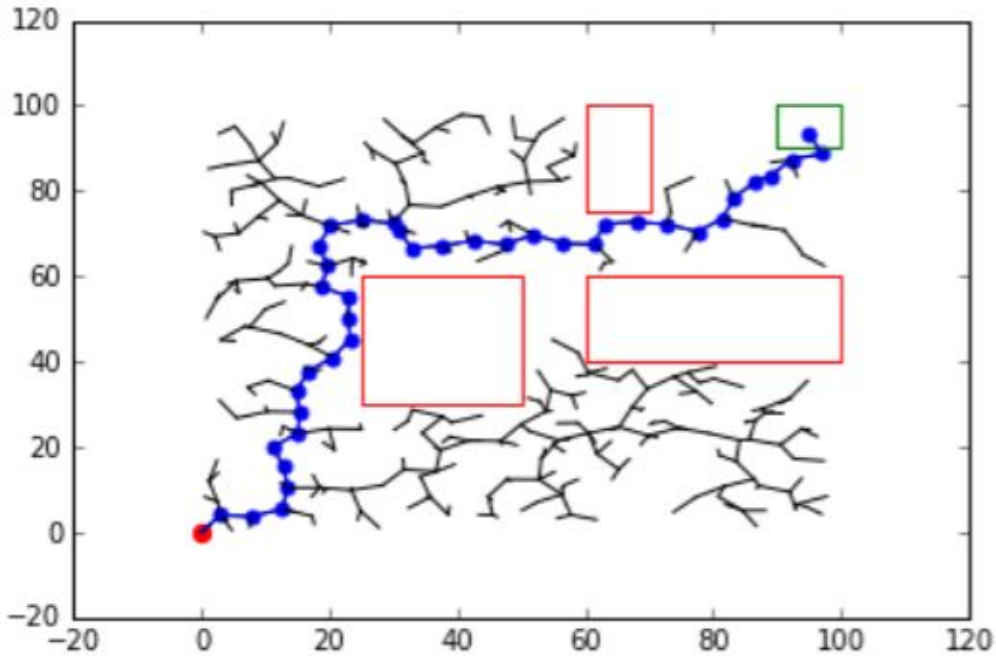


Figure 2.5: Rapidly Random exploring Trees generating a path[41].

PRM vs Tree-based

Though not always able to find an existing solution, the tree-based motion planners are often more suited for changing environments were previously generated configuration space cant be trusted. This is because the tree-based solutions use less computation power for a whole path planning task, whilst the PRM method is dependent on reusing its generated road map to be more efficient. Tree-based methods are also better at handling constraints on the dynamics of the robot. If a robot needs to transport a glass of water, making the robot constrained to keep the glass in an upright position during the whole path execution. This is a relatively straightforward task for the Tree-based system, by modifying the rules for how the tree can be generated. A road map approach however would need to find an approximate route from the generated road map.

2.3 Sensors

Setting up the hardware so that it consistently producers good and stable representations of the batteries are therefore a major aspect of any industrial end-to-end perception system. The perception system can be incredibly competent, but if the data fed to the system is sub par, a accurate result can not be expected.

2.3.1 3D-camera

3D data is used to provide a digital representing of shapes and surfaces of real-world objects. Two of the more common methods of 3D cameras uses either structured light or time-of-flight systems. The most common applications for time-of-flight are autonomous navigation and scanning of buildings [14], since time-of-flight systems often use light outside of the visible spectrum, it is less affected by light pollution. Structured light is more suited for short-range applications in indoor environments where the light conditions can be controlled. This is because a structured light system projects a pattern of visible light to find the range, light reflecting from the environment can therefore distort this pattern.

The data from these cameras is commonly represented as a point cloud with the file format `.PLY` (Polygon File Format) or `.PCD` (Point Cloud Data).

Structured light sensors

The method used to generate point clouds in this thesis is the structured light method. A structured light system uses a camera and a projector to judge the distance to the surface from the point of view of the camera. A pattern is projected onto the surface, often a vertical striped pattern and the camera records the deformation of the pattern.

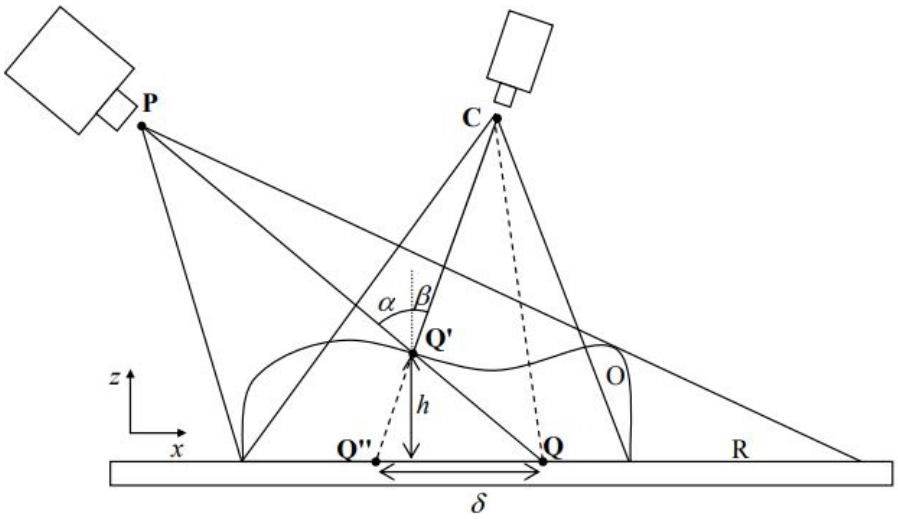


Figure 2.6: Structured Light setup [42]

Figure 2.6 represents a structured light setup, where P represents the point of projection, C represents entrance point to the camera sensor and the plane represented by R is the plane defined as the zero height plane. A line is projected from P to Q. The point where the projected line intersects the object O is represented by Q'. The displacement is visible inn the camera as $\delta = |Q''Q|$. To find the height h of pixel Q' Equation (2.5) can be used, α and β is the projector and camera angle, found knowing they relative position, and the pixel position in the two frames [42].

$$h = \frac{\delta}{tan\,\alpha + tan\,\beta} \tag{2.5}$$

Point by point, a point cloud can be generated with this method. To make the process less recourse intensive, projecting a line is preferred. By projecting a line, the curve along that projection line can be calculated with fewer steps. A representation of the object can be built up by projecting the light on one half of the projectors projection field, calculate the points on the intersection line. Split the halves into fourths with one dark and one bright, and repeat the calculations for the new intersections. Repeat the halving until a fine enough resolution has been achieved. By looking at the change in brightness of the pixel at every pattern, the position of the pixel can be found. A visualisation of the Pattern splitting process is shown in figure 2.7.

2.3.2 Camera configurations

A great camera is usually not enough to take a great picture. Tho a great camera can give some leeway in how to configure it correctly, the camera mostly provides the tools

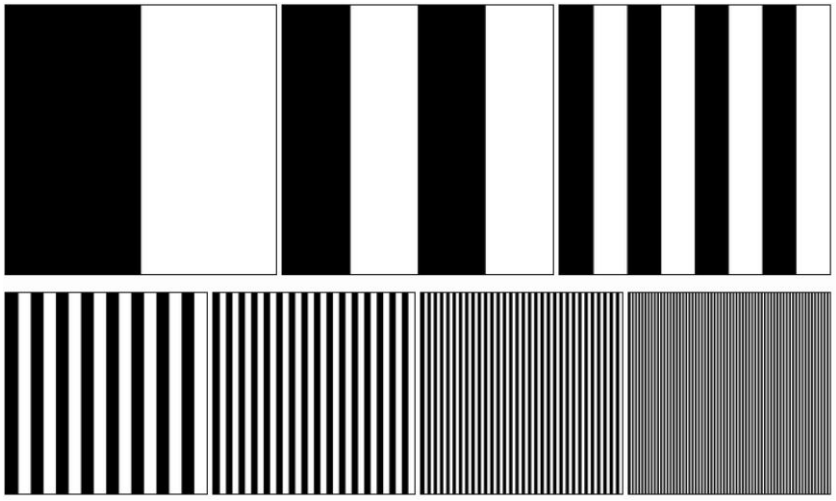


Figure 2.7: Sequence of projected patterns[43].

necessary for taking a great picture. The quality of the photo is often more dependent on how the user configures the camera, taking into consideration the environment and the object of interest, to find the optimal configuration. The same can also be applied to point cloud capturing, especially when using a structured light camera since it uses the same sensor input as a normal camera. A good structured light camera can also produce low quality point clouds if the camera setting is not suited for the environment or the object of interest.

Optical properties of Materials

How light interacts with objects is important for how well a structured light sensor. The structure light senor is dependent on how the light from the projector gets reflected in the camera. How light interacts with an object can be described by the three properties, specularity, absorptivity and transmissivity. The properties of an object can vary based on the frequency of the light[44]. Figure 2.8 features an illustration of the different properties explained in this chapter.

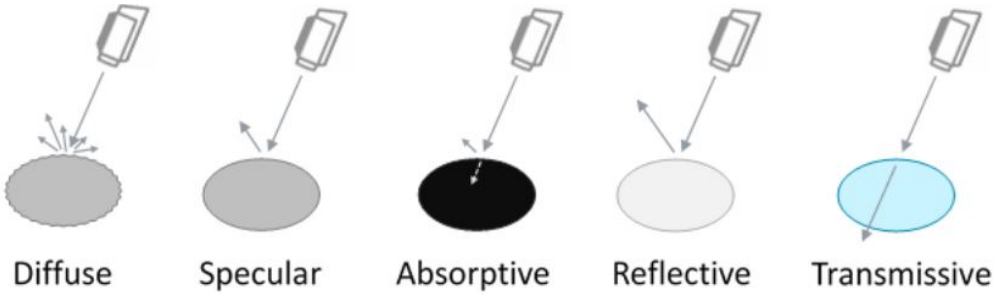


Figure 2.8: Different optical material properties [43].

Specularity Specularity can be thought of as how well an object reflects light. Objects with high specularity are often described as reflective. On the other side of the spectrum, objects with low specularity can be described as mat or diffuse. Specularity is mostly due to the surface smoothness of the object. An object with a smooth surface tends to reflect light more uniformly, giving the effect of being able to see a reflection in the surface[44].

Absorptivity Absorptivity is described as how much of the light that hits the objects get transformed into other forms of energy, like heat or electricity, in effect making the

object appear darker. Objects, where the light gets reflected or passes through, have a low absorptivity score[44].

Transmissivity Transmissivity describes how much light can pass through the object. A typical high transmissive object is glass[44].

Capture settings

The Zivid camera used in this thesis has two ways of configuring the camera to get good results, the camera settings and filter settings. Camera settings giving the basis for a good picture, whilst the filters mostly deal with problems more related to the structured light technology.

Stops Stops are not a camera setting in its self, rather a representation of how bright an image is. To "move up one-stop" is to double the brightness of an image, and to "move down one stop" is to halve the brightness. All the actual settings of the camera, such as exposure time or aperture can be thought of as either increasing or decreasing the amount of stops[45].

The Zivid camera has a total of 23 stops available. The 23 doubling is what gives the Zivid camera its dynamic range, meaning how bright the brightest parts can be, and how the dark darkest parts can be. Moving up and down stops shifts moves the entirety of the dynamic range up or down[45].

Exposure time Exposure time is a measurement of how long the camera shutter stays to let the light sensor collect. The recommended exposure time is dependent on the light frequency of the ambient light in the area. This is because the light level of the image may vary if the exposure time does not match the frequency. In most European countries, including Norway, the standard ambient light frequency is $50\,Hz$. To follow the recommended setting, the following equation can be used[46].

$$t_{exp} = \frac{n}{2f_s}\ ,\ n = |Z| \tag{2.6}$$

The n is a positive integer, and f_s is the light frequency. This results in the recommended exposure time for the system in this thesis to be a multiple of $10\,000\ \mu s$. To double the exposure time results in going up one stop[46].

Aperture Aperture is how wide the opening of the camera lens is. The aperture is often described as the f-number. The f-number is given by equation (2.7)[47]:

$$N = \frac{f}{D} \tag{2.7}$$

Where N is what is referenced as the f-number, the f is the focal length (not to be confused with what is called the f-number), and D is the diameter of the entrance pupil[47].

Most modern lenses uses a standard f-number scale, where each step corresponds to a power of the square root of two. This is so that every integer increase corresponds to one stop up or down. The following rule for f-number gives the correct ratio, where n is an integer[47]:

$$N = \frac{f}{\sqrt{2}^n} \tag{2.8}$$

The Zivid camera combines both focal length and diameter of the entrance pupil into one setting they call iris. This iris setting follows the standard ratio explained in this chapter, uses the values from 0 to 72, where 0 is completely closed and 72 is all the way open. The following table shows the corresponding f-number and Stops in regards to Iris value[47]:

Iris	72	46	35	27	21	17	14	12	10	8
f-number	$f/1.4$	$f/2$	$f/2.8$	$f/4$	$f/5.6$	$f/8$	$f/11$	$f/16$	$f/22$	$f/32$
Stops	$+4$	$+3$	$+2$	$+1$	0	-1	-2	-3	-4	-5

Table 2.1: Corresponding f-number and stops to Iris camera setting.

Brightness Brightness refers to the brightness of the projected light. Increasing the projector brightness is an efficient way of increasing the signal-to-noise (SNR). By increasing the strength of the projector, the range of signal captured by the camera will also increase, decreasing the effect of noise. The increase will however saturate the camera a one point [48].

Gain The Gain controls how much pre-amplification the read-out sensors of the camera gets. Increasing the gain will therefore increase the sensitivity of the camera to light. Increasing the gain does however increase all the input signal, including noise and peaks. Increasing the gain is, therefore, best suited for situations where the time available for capturing is limited [49].

Zivid built inn filters

The built-in filters the Zivid camera provides is largely directed at the challenges a structured light sensor experiences. These are therefore often used to correct problems encountered when a pattern gets projected and reflected in the camera.

Noise filter When the pattern gets reflected in the camera, there is an expected level of noise in the signal. This noise can come from a variety of different sources, but the effect is that when the projected pattern is faint enough, the noise can overshadow the actual signal[50]. The data extracted from signals with a low signal-to-noise can therefore be unreliable. The spectrum signal-to-noise ratios are shown in figure 2.9.

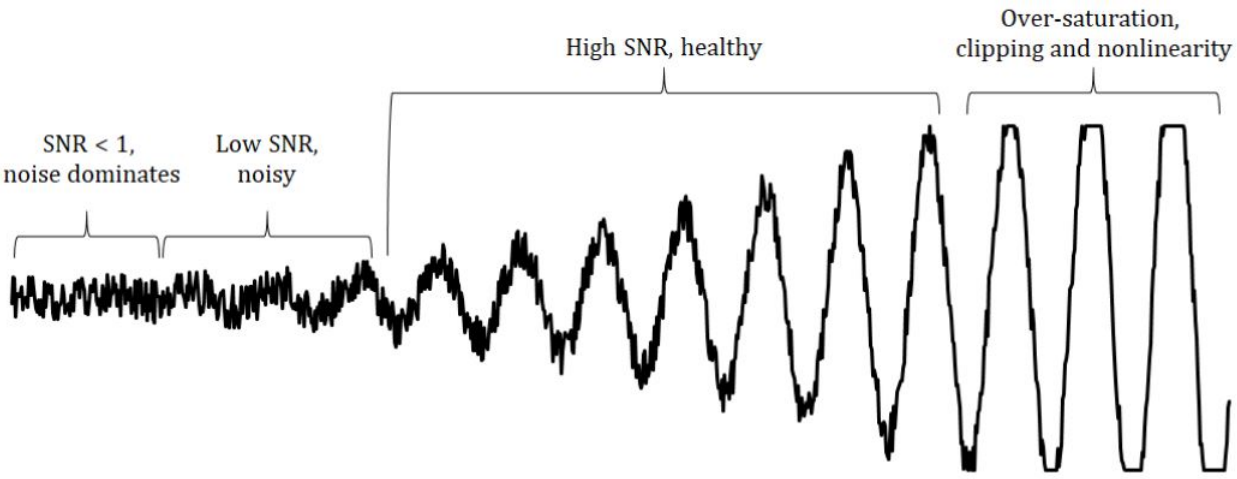


Figure 2.9: Signal-to-noise spectrum [51].

The built-in noise filter of the Zivid camera removes all points which are deemed below a healthy signal-to-noise ratio, inn effect removing unreliable point, but can make darker components less likely to show up in the point cloud[50].

Reflection filter When the projected light hits reflective surfaces, the data given by these points can become unreliable. The point affected by reflections is typically seen as small regions of mid-air floating points which should not be there. These regions are often referred to as "ghost planes" and typically stretches away or towards the camera.

Outlier filter The outlier filter is designed to remove points that appear relatively far from other points, implying it is a result of noise. The outlier filter can be tweaked to the needs of the project, by increasing or decreasing the threshold value for removal. Say if no other point is found in a 5-millimetre radius from the point, the point is removed[52].

The removal of such outliers is essential for project completion since the generated point cloud often is used as input for the obstacle avoidance system of the robot. An outlier in the point cloud can then cause the path planning algorithm to fail to find a path[52].

Gaussian smoothing As is depicted in figure 2.9, there is still noise in the signal even with a healthy signal-to-noise ratio. The Gaussian smoothing filter is mainly there to improve the absolute noise. The filter does this by smoothing the points in a small region. This region can be expanded by increasing the sigma.

The Gaussian filter is great for making surfaces smooth, but if overdone, it can affect the representation of small features, such as cables. The smoothing can however potentially increase the performance of a vision system. It is therefore recommended to find a middle ground between small details and smooth surfaces.

Contrast distortion filter Contrast distortion is an artefact that can be created where there is an abrupt change in intensity in the 2D image. The abrupt change can create a jump in position along the border. The artifact can be seen in figure 2.10 where a checkerboard pattern and a highlight along a metal surface produces the effect. The effect is a result of blurring in the camera lens[53].

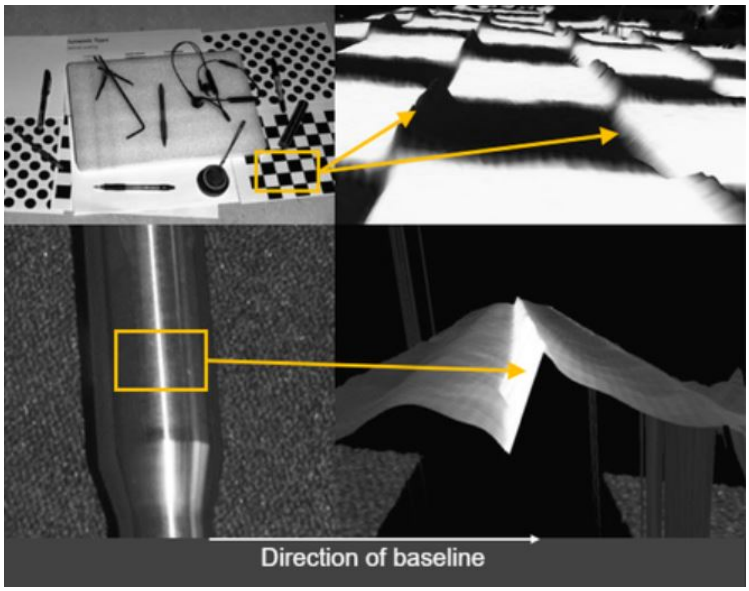


Figure 2.10: Examples of contrast distortion[53].

The Contrast distortion filter is designed to handle this, by either removing the points or correct them, where any points over a certain threshold are removed. The strength of the filter can be overcompensated and remove excessive amounts of points[54].

Saturation filter The saturation filter works on the other side of the spectrum showed in figure 2.9. Where the noise filter handles everything that is below a healthy signal-to-noise ratio, the saturation filter removes every point where intensity is saturated. The confidence of the points which is over the saturation limit is low and can therefore introduce noise or other artefacts in the point cloud.

High Dynamic Range (HDR)

High dynamic range capturing is a technique to extend the range of exposure in an image without saturation. While a human eye can adjust to the environment, decreasing the amount of light through the pupil. A camera often uses 8-bits per channel, giving a span of 256. The range not enough to capture the whole range of lighting in the scene saturating the brightest parts, and underexposing the darker parts[55]

To solve this a HDR capturing algorithm captures a set of pictures with different exposure levels and merging the images into one. The images on their own often look either overexposed or underexposed, in practise the span of 256 levels is moved up or down in the spectrum. If the levels are moved up the normally bright regions will be overexposed, but the dark regions previously not visible will become defined. The same situation happens when moving the range down making the brighter regions of the image defined [55].

2.3.3 Eye-in-hand system

An eye in hand configuration is a robot configuration where the range sensor, commonly a camera, is fixed by a moving robot. This sensor can then be moved around in the workspace of the robot. The mobility an eye-in-hand system provides can expand the sensors use case. It can extend the range of the sensor, by moving the sensor to several points of views, it can cover the regions of several stationary sensors. With an eye-in-hand system, one sensor can serve as both a long-range sensor, tracking a large area with low resolution and a short-range sensor, able to pick up small details.

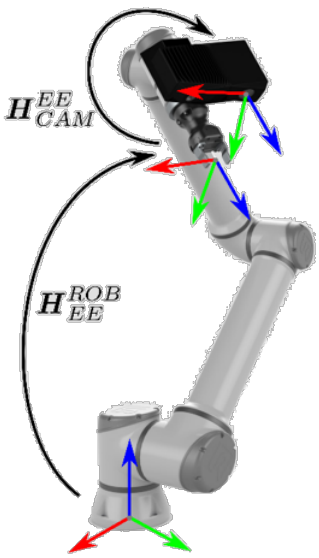


Figure 2.11: Eye-in-hand sensor configuration[56].

An eye-in-hand configuration can also be advantageous in model-free end-to-end systems. In the case of LIB pack disassembling, the variation in battery pack layouts can mean that regions of the pack may avoid getting scanned. An eye-in-hand system, with the right point of view generating algorithm, can be made to adapt to a new geometry, moving the camera to cover un-scanned regions.

When generating data from the sensor, the data do not include what frame of reference the data is generated from. To make use of the sensor data the data needs to be transformed to a stationary which is shared through the system. This frame is often the robot base frame, world frame or origin of the coordinate system. By keeping track of the position and orientation of all the frames leading up to the sensor frame, the position and orientation can be found by forward kinematics. This will give you a transformation matrix with which the sensor data can be multiplied to represent it from the perspective of the stationary frame. If the two transformation matrices $H_{EE}^{ROB} + H_{EE}^{CAM}$ showed in figure 2.11, are multiplied together the resulting matrix would be H_{ROB}^{CAM} witch wold represent the necessary transformation to get the sensor data in the base frame.

2.4 Computational complexity

Any algorithm has a related time complexity that describes how the number of operation needed, N, changes related to the input size n. The time complexity is only approximated for large input spaces. Thus it is not relevant for small point clouds of 100 or 1000 points, but when the point clouds get large, it has a vast impact on the overall computation time.

When setting up a more complex program, combining sequentially run processes that each has its associated time complexity, the total time complexity of the program will be equal to that of the greatest of the processes.

Big-O notation

The time complexity of a function or process is most offend denoted with the Big O-notation, where the smallest complexity is $O(1)$, and the greatest commonly used is $O(n!)$. The time complexity does not tell how much time a process takes, but rather how the time scale as the input scales. The time complexity of any function is the same as its highest-order component after being reduced. In the example shown below, the term c_1x^2 has the highest order.

$$f(x) = c_1x^2 + c_2x + c_3$$
$$f(x) = O(n^2)$$

Picking the first element of a list has a time complexity of $O(1)$ as it will always require the same amount of operations. Lopping over a list has the time complexity of $O(n)$ as the length of the loop is equal to the size of the list. Finding the shortest distance between two point clouds with a brute force method (as describes earlier) has a time complexity of $O(n^2)$.

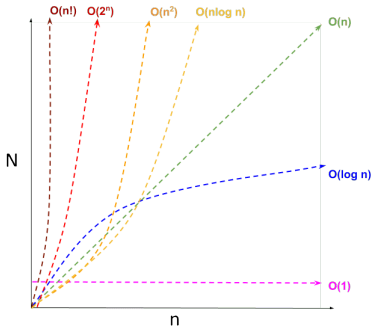


Figure 2.12: The number of operations N plotted versus the input size n for different time complexities.

The difference between having a program with a time-complexity of $O(n)$ versus $O(n^2)$ might seem small, but for large input (like a point cloud) this has vast differences. For instance, a computer that manages $1,000,000n/s$ that operates on a point cloud with $n = 10,000$ points will use $10[ms]$ for a $O(n)$ type process, but will use $100[s]$ on a $O(n^2)$ process. Some common time complexities and how they grow with input size is shown in figure 2.12.

2.5 Data representation

The quality and representation of the data is a vital part of the performance of any classification algorithm. Too diverse and too correlated data makes it hard to find relations [57], while too sparse data can make separate observations indistinguishable. Therefore, it is important to select a proper data representation.

2.5.1 Point cloud

A point cloud is a data structure that represents 3D data. The data itself consists of points p that, often Cartesian coordinates in 3D-space and optional attributes $p =$

$\{x, y, z, a_4, a_5 \dots a_d\}$. The attributes can be any value related to the application. Common attributes can be local curvature, normal vector or RGB-value (colour). Most generally, a point cloud is an unordered set of points $X = \{p_1, p_2 \dots p_n\} \subseteq \mathbb{R}^d$, where d denotes the number of attributes. There are cases of ordered point clouds, hence the points are listed after any of the attributes (commonly one of the positions). Although there exist several ways of representing 3D data like mesh, voxel assemblies or CAD, the point cloud is the simplest data type that still conserves surface contours.

2.5D image

In a 3D scene viewed from a single perspective, the points can be ordered into pixels in a 2D image with an extra channel representing the distance from the observer or the depth coordinate of the corresponding point. This is illustrated in figure 2.13. For multi-view scenes there might accrue an overlap of points, thus assigning a depth to pixel is ambiguous. 2.5D images benefit from being closely related to 2D images, as concepts and methods are transferable.

Figure 2.13: An illustration of a 2.5D image. Left: A regular two-dimensional RGB image illustrating the color attributes of the data. Right: The corresponding depth-map illustrating the z-coordinate of the data.

Voxel grids

A voxel is the 3D equivalent of a 2D pixel, similar to a cube being the 3D equivalent of a 2D square. By dividing a space occupied by a point cloud into a discreet number of fixed size voxels one can create what is known as a voxel grid. For a simple point cloud without any additional attributes, the voxels not containing any points are set as inactive, while those with any number of points are set as active with the centre coordinates of the voxel as its attribute. The attributes of a voxel can be determined by either the max value or the average value of the enclosed points, thus having similarities to pooling operations performed in 2D CNNs.

A voxel grid is ordered such that all the voxels are evenly spaced and can be indexed in a 3D grid, thus allowing for 3D convolution. The voxel grid allows for an alternative representation that reduces the number of points and that can efficiently be accessed by octrees. Octrees are explained later in paragraph 2.5.1. Nevertheless, the voxel grids struggle with preserving details in the data and can in some cases be memory inefficient as a sufficient amount of the voxels are empty. The size of the voxels is a trade-off between representing the underlying data as small voxels can pick up mode details, and memory and time complexity as a large voxel size leads to fewer empty voxels as shown in figure 2.14.

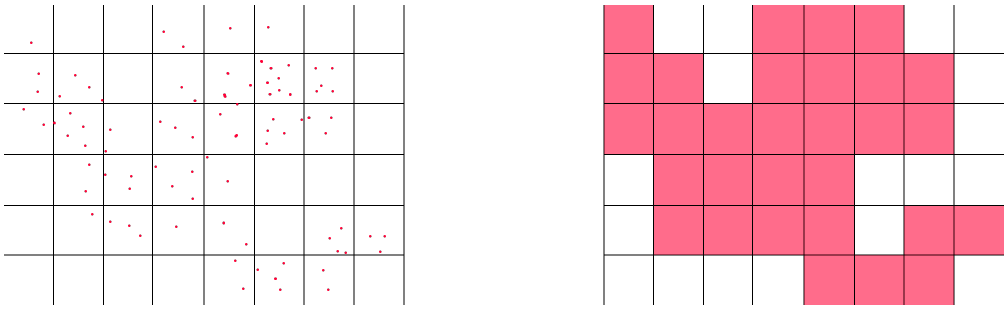


Figure 2.14: An illustration of a voxel grid representation of a point cloud. Left: The original point cloud. Right: The voxel grid representation. Note that the amount of enclosed points does not is not taken into account, thus voxels with only a single enclosed point have the same impact as one with five points.

Data filtering

Any point cloud captured by a sensor will have some amount of unwanted data or noise. Outliers are the most common type of noise and are often dealt with by either a simple pass-thru filter or a statistical filter. A pass-thru filter filters out all data points that do not satisfy a given requirement. For instance, requiring a minimum value for a given coordinate. A statistical filter will look at the distribution of features or some calculated feature. Data points related to extreme cases in the distribution is then removed.

Some data sets are too large to process, contain redundant or over-represented data, thus it needs to down-sampled. A random remove filter will randomly remove points until a target size is reached. The random remove filter can also be weighted, thus being bias towards data points with a given feature. The weighed version is useful for removing over-represented data.

Pooling is similar to voxeling in that it converts a collection of points into a single point representation. While Voxeling is done by evenly dividing space, pooling is done by forming small clusters named "pools". The workings of pooling are illustrated in figure 2.15.

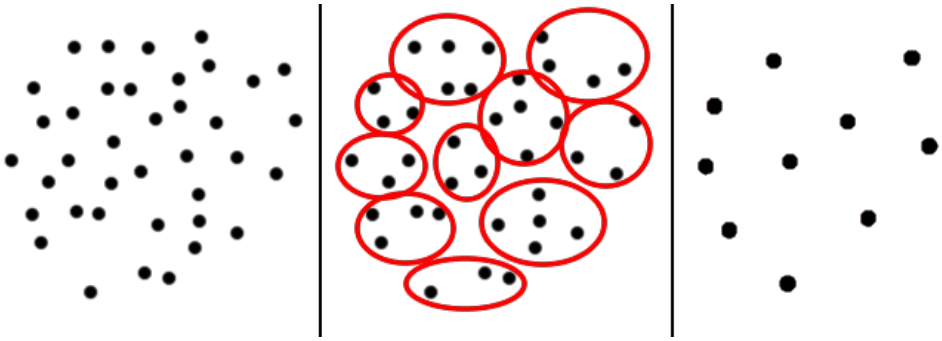


Figure 2.15: An illustration of pooling. Left: The original point cloud. Middle: Pools are formed. Right: New points are formed with attributes based on the original points in the respective pool.

Pooling and voxeling groups several data points together, and represent them as a single point derived from the average or max value of grouped data points. Pooling and voxeling remove redundant data points and over-represented data but removes local details from the data. By dynamically adjusting the pool and voxel sizes, local details can still be preserved.

Octrees

A point cloud is an unordered list, thus the indexing does not provide distance or relational information. If the point cloud only had one attribute, the points could be ordered chronologically. As the points have three spatial attributes, it is ambiguous to order them chronologically by distance.

By splitting the occupied space into nested octets, each point can be indexed by a binary number that also encodes the relation to neighbouring octets. Finding closely distanced points can then be done by a binary search tree. Since the space is split into eight octets at a time, the binary tree search is named octree. They work the same since an octet is equal to three nested binary decisions ($8 = 2^3$). Setting up an octree has its related time complexity, but when done, it improves the time complexity of common processes by converting them from brute force to tree searches. Given a known point, finding the closes point in a brute force approach demands a search thru the entire point cloud, thus a time complexity of $O(N)$, as explained in chapter 2.4. As depicted in figure 2.16, by using an octree, the time complexity is reduced to $O(logN)$. As shown in figure 2.12, the time complexity becomes important for the overall performance when working with large data sets.

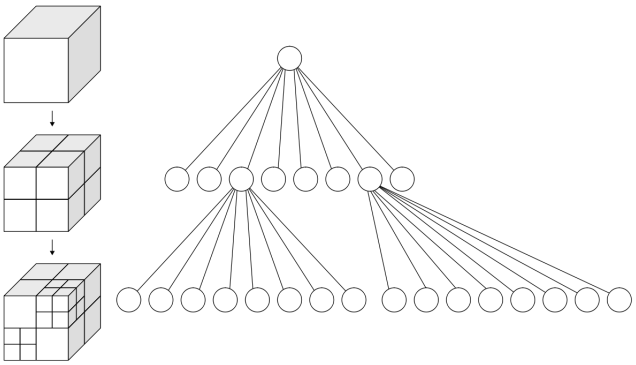


Figure 2.16: An illustration of the workings of octrees.

2.5.2 Graph representation

A graph $G = (V, E)$ represents data as a set of N nodes $v_i \in V$ with edges $(v_i, v_j) \in E$ connecting related nodes. Each node e_i has its own related feature vector x_i containing the initial data of the node. For relation based data, like time-linked or space-linked data, the graph provides a flexible way of representing the data while preserving the relations. The graph is a general concept, where the specific cases go under other names and are easier to relate to. For instance, a time series can be interpreted as a one-dimensional graph where nodes are instances in time and the vertices are time steps.

Common parties used to describe graphs, and classify them are:

- **Heterogeneous/homogeneous**. Whether the nodes are of the same type, thus whether they hold the same types of attributes or not. Homogeneous graphs are what describes point clouds as all the nodes represent points, whereas heterogeneous graphs are more used to describe databases and knowledge graphs.
- **Directed, underacted and weighted edges**. The edges themselves can take a value, thus embedding how strongly related two nodes are. An unweighted edge can be considered as an edge with a weight equal to one, thus in principle, all edges are weighted. Edges are also either directed or underacted, thus describing whether the relation between two nodes only applies when moving from node A to B, and

not from B to A. Underacted edges can be viewed as a special case of directed edges where two directed edges with the same weight connect the same two nodes but are oppositely directed.

- **Node degree and dense/sparse graphs**. The degree of a node denotes the number of related edges. For a directed graph, the nodes have a separate degree for incoming edges and outgoing edges. In a dense graph, most nodes are connected with an edge, thus each node is related to the global properties of the graph. On the other hand, in a spares graph nodes only have few connections and can be clustered, thus being more related to local features of the graph. Dens graphs are often small as their number of edges is close to the square of the number of nodes, while a sparse graph is more scalable. Molecules or small social network are examples of dense graphs, while a point cloud or a database is examples of sparse graphs.

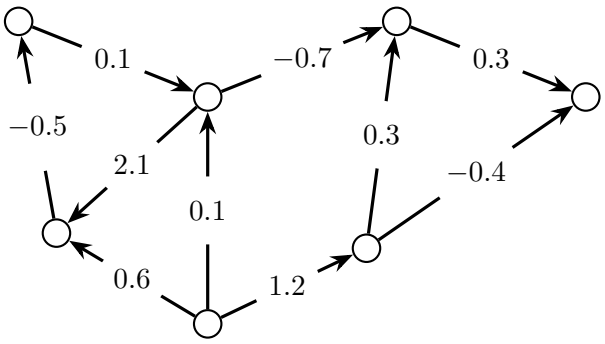


Figure 2.17: An example of an arbitrary dens homogeneous directed graph with weighted edges.

Classification problems

Classification of data in graph representation is divided into three types [58]:

- **Node classification** intends to predict the class of the nodes in a graph based on relations with neighbouring nodes. Commonly done by a recurrent or convolutional graph neural network to produce high-level representations, and a multi-layer perception to perform classification. Common usage is the prediction of optimal advertising based on users relations, completion of incomplete data sets and error correction in code [59].

- **Edge classification** aims to determine how nodes are related. By assuming a fully connected graph, the relation between any nodes can be determined by either assigning the corresponding edge with a binary value. In other settings, the connectivity is known, but not the type of relation. The relations are then decided by assigning the edges with weights taken from a given interval. Common applications are estimation relation between profiles in a social network, and traffic load of a road network [60].

- **Graph classification** focuses on classifying en entire graph. It is most commonly used on smaller graphs (100 nodes, or less), as the complexity becomes too significant for large graphs. An example of graph classification is the prediction of molecules properties [61].

2.5.3 Dimension reduction

In some cases, a high-dimensional data set is too large to process, or it is inefficient due to low variance in some of the dimensions. A statistical dimension reduction of the

data sett conserves most of the information but embeds the data into a lower number of dimensions. The new dimensions do not have any physical meaning but have a higher variance than the original dimensions, hence more useful. There exist several different methods for performing nationality reduction, all with their application [62]. Two methods that are useful for this thesis is the principal component analysis (PCA) and the t-distributed stochastic neighbour embedding (t-SNE). The time complexity of PCA and t-SNE is shown in table 2.2.

	Time Complexity
PCA	$O(d^2n + n^3)$
t-SNE	$O(n^2)$

Table 2.2: The time complexity of the selected dimension reduction algorithms. "d" denotes the original number of dimensions.

Principal component analysis

PCA is an unsupervised linear transformation algorithm commonly used due to its simplicity. It produces new features, named principal components by determining the maximum variance of the data. The original high-dimensional data is then projected down to a lower-dimensional space with orthogonal axes formed from the principal components. The new embedding has the highest variance along with the first principal component, and the following principal components are ordered in descending order after their variance. The last principal components will have low variance, and can therefore be excluded with little to no impact on the data. The workings of PCA is illustrated in figure 2.18.

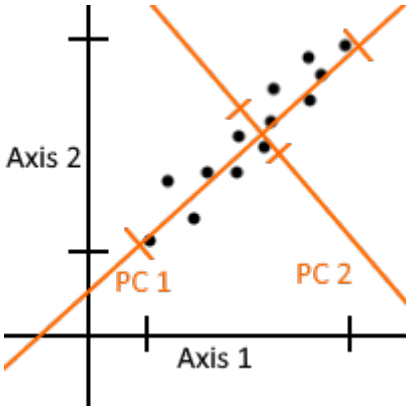


Figure 2.18: An example of a PCA, the two principal components (PC 1 and PC 2) are shown in orange. PC 1 have a grater variance than the original axis, while PC 2 have a lower than the original axis.

t-Distributed Stochastic Neighbor Embedding

t-SNE is an unsupervised non-linear transformation algorithm that preserves the structure of the high-dimensional data when performing dimensional reduction [63]. Since the structure is somewhat preserved, points that are close in the original high-dimensional data will remain close. Therefore, t-SNE is useful for exploring and visualising relations between high-dimensional data [62]. t-SNE is done by first applying Stochastic Neighbor Embedding on the original data set, thus converting the Euclidean distances in the high-dimensional data into a set of conditional probabilities that represents similarities between data pairs. Afterwards, a student t-distribution with one degree of freedom is used to obtain a second set of probabilities in the target dimension (commonly 2 or 3) [62].

2.6 Machine learning

Machine learning is a field within data science that focuses on developing tools and algorithms that can learn complex relations in features from a set of data to then be able to perform classification or regression tasks on similar data. Machine learning is often confused with artificial intelligence and deep learning as the terms are often used interchangeably. Artificial intelligence (AI) categorises all programs and algorithms that perform anything that can be seen as intelligent, thus including primitive and simple algorithms like the k-nearest neighbours' algorithm (K-NN). As illustrated in figure 2.19, machine learning is a subset of artificial intelligence rather than a synonym. The therm deep learning refers to a particular field within machine learning that focuses on deep neural networks, and that has had a big focus last couple of years due to strong results in a variety of tasks and applications.

Machine learning is often divided into three categories based on how they learn from data. In all the cases the algorithm f is provided with data in the form of a feature vector $x \in \mathbb{R}^m$ that is used to make predictions $f(x) = \hat{y} \in \mathbb{R}^v$. m and v denotes the number of attributes for the data, and the number of classes for the prediction. This can be seen as a mapping from input-space to a complex embedding $f : \mathcal{X} \to \hat{\mathcal{Y}}$. The end goal is then to train the model so that the predictions become as closes as possible to the ground truth y, thus $f(x) = \hat{y} \approx y$. The difference between the three categories is how they use data to train. The first, and most used, is supervised learning. In this case, the true value is known for the entire data-set, thus the performance is measured by a simple loss function $\mathcal{L}(y, \hat{y})$ that directly compare the estimation and the ground truth. The second category is called unsupervised learning where it is assumed that there exists a ground truth, but it is not known. This kind of machine learning is most used for clustering big sets of data. The loss function often revolves around perceiving relationships between similar disappoints. In other word, similar inputs should give similar outputs $x_i \sim x_j \to \hat{y}_i \sim \hat{y}_j$. The last category is named reinforcement learning, where the data does not come in the form of a set, but rather is an interactive environment. This kind of machine learning is used for mastering chess and other games with predefined rules.

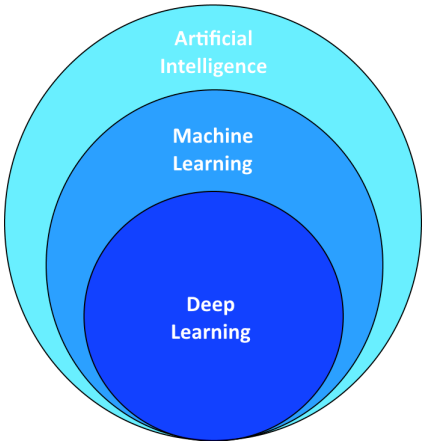


Figure 2.19: An illustration of the relation between artificial intelligence, machine learning and deep learning.

2.6.1 Neural networks

Within machine learning, neural networks are the most common architecture for performing the mapping. The concept of neural networks are based on how the neurons inside the human brain from a micro perspective act under simple rules, but on a macro level perform complex tasks. Any neural network consists of several layers h, each filled up with artificial neurons that holds a single scalar value. The first layer h_1, named the input layer, is where the feature vector $x \in \mathbb{R}^m$ is feed into the network. Therefore, the input layer needs to have m number of artificial neurons, each representing one attribute of the feature vector. The last layer h_n, named the output layer, provides the embedding $\hat{y} \in \mathbb{R}^v$, and therefore have to consist of v artificial neurons. The intermediate layers $h_2 \dots h_{n-1}$ are named the hidden layers as these are only for passing information and are not interfered with.

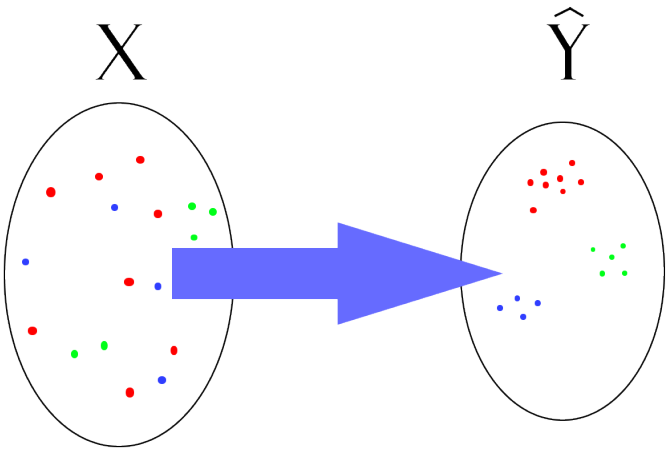


Figure 2.20: An illustration of $f : \mathcal{X} \to \hat{\mathcal{Y}}$ where, based on there attributes in the input space $\mathcal{X}$, the data points gets mapped into the embedding space $\hat{\mathcal{Y}}$. Similar points that were hard to distinguish in the input space but have the same ground truth (the color) have been mapped close together in the embedding.

The layers are then connected by edges representing a one-way connection between two artificial neurons in different layers. The edges are associated with a scalar weight α, which represents the impact level of the connection. The outline of edges is inherent to the architecture of the neural networks itself and provides design options that result in different characteristics of the network. Most commonly used is feed-forward fully connected layers where all neurons in two adjacent layers are connected and the information only passes in one direction (no recurrence).

The idea is that each neuron in the hidden layers will learn some sort of complex feature about the input data that the output layer can use to form a decision. The size and number of hidden layers is application dependent, thus becoming a notable part of the design process. A wide network (many neurons per layer) will be able to pick up more features, while a deep network (many layers) will be able to form more complex features.

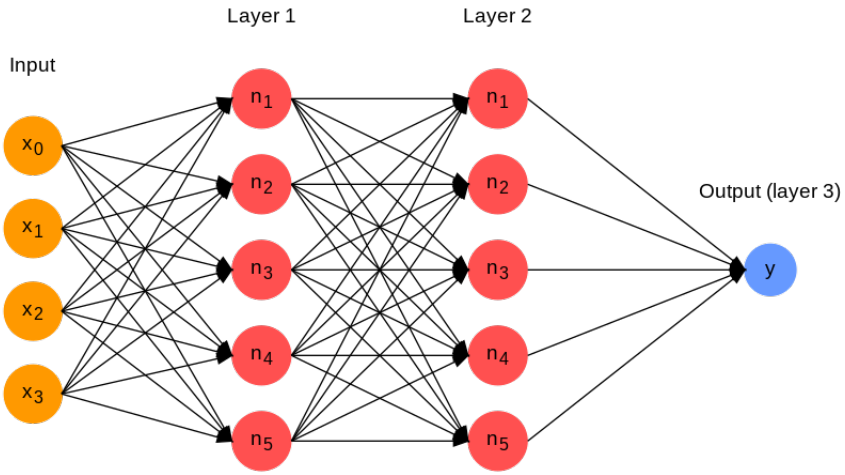


Figure 2.21: An illustration of a feed-forward fully connected neural network with an input layer (orange), two hidden layers (red) and an output layer consisting of a single neuron (blue).

Artificial neurons

The artificial neurons themselves are used to aggregate information. The value of any neuron is calculated with (2.9), where α_k is the weights of the incoming edge to neuron k, x is the values of the connected nodes from the previous layer and β_k is the corresponding bias. $\phi(\cdot)$ denotes the activation function that intends to prevent single nodes from blowing up, thus having a too great of an impact. The choice of the activation function is a part of the design choices for a neural network. The edge weights α and the biases β are learnable parameters that get optimised thru the training process.

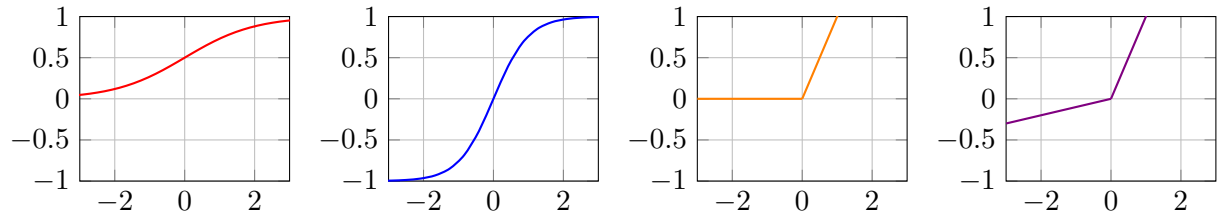
$$x_{h,k} = \phi(\beta_{h,k} + \sum \alpha_{h,k,i} \cdot x_{h-1,i}) \tag{2.9}$$

The entire layer can be calculated at once in feedforward networks since all the nodes are independent of each other. Thus feeding the information thru any layer can be written as a matrix multiplication (2.10), or with the combined parameter-matrix $W = [A|B]$ (2.11) which concatenates the weights and biases into one matrix.

$$X_h = \phi(B_h + A_h \times X_{h-1}) \tag{2.10}$$
$$X_h = \phi(W_h \times X_{h-1}) \tag{2.11}$$

Activation functions

It is desirable that the neural network learns complex features and take a collective decision. Therefore, activation functions are included to avoid the network assigning a single node to light up at each different instance of training data. The activation functions limit the size of the output by mapping to a known, often limited, interval. There exists multiple activation functions [64], and the choice of activation function is an important design choice [64], [65]. But all activation functions have to be differentiable to work [57].



(a) Sigmoid (b) tanh (c) ReLU (d) Leaky ReLU

Figure 2.22: Some selected activation functions plotted for $x \in [-3, 3]$.

The most common activation functions are quite simple, as complex activation functions often do not generalise and are therefore only used in narrow tasks [65]. The most common activation functions are the following, or versions of them:

- The **sigmoid** function (2.12), sometimes referred to as the logistic function or squashing function, is a non-linear activation function mostly used in feedforward neural networks. It is most commonly found at the output layer of shallow deep neural networks, as it suffers drawbacks on deeper neural networks due to struggles with calculating the gradient (sharp damp gradients, gradient saturation and non-zero centred output) [64].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

- **tanh** (2.13) is similar to sigmoid in that both have this squishing property. Tanh is superior to sigmoid as it removes some of sigmoids drawbacks by centring the outputs around zero [57] as shown in figure 2.22b. Both sigmoid and tanh still suffer from gradient saturation due to the asymptotic nature of the functions as shown in figure 2.22b and 2.22a. As the slope of sigmoid and tanh always is less than one, they both struggle with the vanishing gradient problem. The tanh activation function is most commonly used in recurrent neural networks for natural language processing like LSTMs [64].

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.13}$$

- **ReLU** (2.14) as it is known, which stands short for Rectified Linear Unit, is the most widely-used activation function for deep neural networks. By allowing the gradients to flown with positive inputs, neural networks with the ReLU function become easier to optimise compared to sigmoid and tanh [65]. The ReLU function rectifies the value for negative inputs, thereby forcing them to be zero and eliminating the vanishing gradient problem. The ReLU activation function is most often used in the hidden layers of deep neural networks for tasks like object classification and speech recognition, especially in convolutions neural networks [64]. Although a significant improvement, ReLU still has limitations as it is fragile during training which may leave gradients to die, thus not updating neurons, and not learn. Several large classification networks like AlexNet, VGGNet, GoogleNet and ResNet all use ReLU [64].

$$ReLU(x) = \max(0, x) \tag{2.14}$$

- **Leaky-ReLU** (2.15) is very similar to ReLU but it has a slight slope (γ) for negative numbers that manages to deal with the zero-gradient problems. By not forcing the negative inputs to zero, but close to zero, the gradients will always be non-zero except for when the input itself is zero. When dealing with floating-point numbers on a computer, the value passed to a neuron will never be zero due to the nature of floating points, therefore, in practice, the gradient will always be non-zero. The parameter slope is commonly $\gamma \sim 0.01$. It is shown that leaky-ReLU performs well in big classification networks compared to other similar activation functions [65].

$$LReLU(x) = \max(\gamma \cdot x, x) \tag{2.15}$$

- The **Softmax** activation function (2.16) is a bit separate from the others in that it does not work on a single input value, but rather an entire tensor. The function scales every element of a tensor so that the total sum of all elements equals one (2.17), thus the softmax of any value reflects the relative size compared to the rest of the tensor. Because of this functionality, the softmax is mainly used at the output layer of classification networks to make the estimates into probabilities. Together with ReLU, the softmax is used on large neural networks like AlexNet, VGGNet, GoogleNet and ResNet [64].

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.16}$$

$$\sum_j softmax(x_i) = 1 \tag{2.17}$$

Loss functions

To train the network, one needs a metric to evaluate the predictions of the output layer $\hat{y}$. The objective of a loss function $\mathcal{L}(\cdot)$ is therefore to compare the output layer predictions

$\hat{y}$ with the ground truth y. For classification tasks, commonly the ground truth has a binary representation and the prediction is run thru a softmax layer. All loss function return a scoring in form of a scalar value, although the inputs are usually tensors.

In the end, it is the loss function that is being optimised, therefore, the choice of loss function must reflect the problem that one tries to optimise. This makes the choice of loss function one of the most critical design choices when working with neural networks [66].

In its simplest form, the loss function is the mean difference, or the sum, of element-wise differences (2.18). This is commonly known as the mean absolute error (MAE) or L1-loss and together with mean square error (MSE or L2-loss) (2.19) works well for simple regression tasks.

$$\mathcal{L}_{L1}(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} |y_i - \hat{y}_i| \tag{2.18}$$

$$\mathcal{L}_{L2}(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{2.19}$$

In regression tasks, it is accepted that there will be an error as there will be a natural variance in any data set. On the contrary, in binary classification tasks, a perfect model should label the data with close to 100% certainty, hence any error should be punished harder. Therefore, it is common to use binary cross-entropy loss (2.20) for binary classification tasks as it results in extremely steep gradients when the error epochs max. This happens due to the nature of logarithmic functions that reaches a singularity for values close to zero. The binary cross-entropy loss function is created to approaches these singularities when the error approaches one (2.21).

$$\mathcal{L}_{BCE}(y, \hat{y}) = -\frac{1}{m} \sum_{i=1}^{m} y_i \cdot \ln \hat{y}_i + (1 - y_i) \cdot \ln (1 - \hat{y}_i) \tag{2.20}$$

$$\lim_{e \to 1} \mathcal{L}_{BCE}(y, \hat{y}) = \infty \tag{2.21}$$

As described by [67], loss functions for image segmentation can be categorised into four groups based on their objectives:

- **Distribution-based loss** tries to minimise the dissimilarity between the prediction vector and the ground truth vector by treating them as distributions. The loss functions in this group mainly base themselves on cross-entropy and are versions of the cross-entropy loss function. The distribution-based loss functions are known for performing well on classification-based tasks [68]. Some of the most relevant distribution-based loss functions are:
  - **Weighted cross-entropy loss** which is focuses on positive examples by adding a weight on the ground-truth label, thus making instances where the ground truth is one (true) more impact-full on the total loss [69]. This method is commonly used in medical analysis where recall and precision are more important metrics.
  - **Distance map penalized cross-entropy loss** weights the cross-entropy by a distance which is derived from the ground truth. This pushes the network to train on boundaries between regions of different labelling as thees are weighted stronger [70].

  - **Focal loss** modifies the cross-entropy loss function to deal with data that is extremely imbalanced between the classes [71]. Focal loss is typically used for doing pixel-vice classification to find small objects in large images, thus the imbalance is several orders of magnitude.

- **Region-based loss**, in comparison with distribution-based loss that evaluates the prediction of each point/pixel, compares a region. The region is either a 2D-area, or a 3D volume, ether guided by a basic geometric shape like a rectangle or a cube thus creating one label for an entire cluster, or guided by the border points/pixels of the cluster. This way of comparing prediction and ground truth gives room for new network architectures that are more applicable for real-time usage and tracking as the prediction regions are easier to follow throughout time-series data. Two common region-based loss function are:

  - **Dice loss** is based on the Sørensen-Dice coefficient, also known as the F1-score, that finds the number of common elements between the prediction region and the ground truth region and compare it to the combined size of the two regions. The dice loss, and its off-springs, as they are region-based are more robust for imbalanced data [72].
  - **IoU loss** is similar to dice loss and have many of the same advantages, but compare the intersection to the union of the regions, instead of the size of the regions [73].

- **Boundary-based loss** is similar to region-based loss in that it works on comparing shapes or clusters rather than individual points/pixels. But instead of looking at the overlap between the predicted region and the gerund truth, it looks at the distance between the contours. This is not a widely used type of loss as the loss function is hard to define in a differentiable way. Commonly it is done by integrating the along the contours to mitigate the average distance.

- **Compound loss** is any hybrid solution of the previously mentioned methods. The most promising compound loss functions are **combo loss** and **exponential logarithmic loss** that both combines binary cross-entropy loss and dice loss. The advantages of these more complex loss functions are that they can be tailored to have the accuracy and detail focus from the distribution-based loss while being robust to imbalanced data.

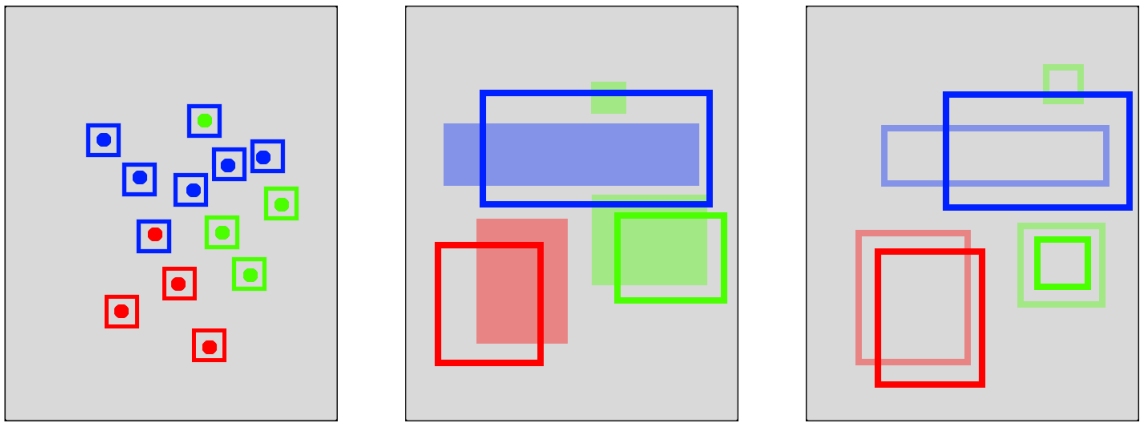


Figure 2.23: An illustration of different types of loss functions. The points and the faded region represent the ground truth, while the rectangles represent the predictions. Left: Distribution-based loss where every point/pixel is individually evaluated. Middle: Region-based loss where the amount of intersection is evaluated. Right: Boundary-based loss where the distance between the contours is evaluated.

Backpropagation and optimisation

To learn from the training data, the network needs a way of relating the loss with the weight. In other words, how to change the weights given a loss value. Backpropagation provides a method for calculating the gradient of the loss surface by finding the partial derivative of all weight with respect to the loss function. Hence, the gradient guides the weights to minimise the loss. The gradient for the entire network can systematically be obtained, by calculating the gradient of the output layer, and then repeatedly using the obtained gradient from a layer to calculate the gradient of the previews layer (2.22). This update rule is the reason why the activation function has to be differentiable.

$$\frac{\partial \mathcal{L}}{\partial W_h} = \frac{\partial \phi}{\partial W}(W_h, X_{h-1})\frac{\partial \mathcal{L}}{\partial W_h} \tag{2.22}$$

Machine learning intends to generalise a model by exposing it to a big number of data and having the weight converge towards an optimal solution. This is commonly done by gradient-descent based methods where the weights are iteratively updated based on the current weights and a weighted gradient (2.23). The weighing of the gradient η, is referred to as the learning rate and provides the step size of any iteration.

$$W(t) = W(t-1) - \eta\frac{\partial \mathcal{L}}{\partial W} \tag{2.23}$$

To compensate for noisy gradients as a result of the variance of the training data, it is common to use batch training rather than updating weights based on a single data point. In batch training, the entire data set is looped over to calculate an "average" gradient for the entire dataset. Although the batch training is more stable it is often too general as it has a hard time picking up extremer cases, and it is time-consuming to loop over the entire data for each update. The concept of mini-batch training (also known as stochastic learning) where the dataset is separated into smaller batches (commonly in the range of 10 to 1000), and the weights are updated for each mini-batch. In a dataset there likely exists some degree of redundancy within the data, this is why a subset of the data can give a good representation of the entire dataset [57]. This technique of working with an estimated gradient is whats commonly known as stochastic gradient descent (SGD) [74].

Determining the learning rate η, or step size, is a big part of the learning process. Ideally, it should be large at the start of the training so that the model can take big steps away from the random staring weights and toward the minimum, and then become small towards the end to let the model converge. A too small initial learning rate would give a slow training as it would need a large number of steps to reach the minimum, besides, if the loss surface is not quadratic nor convex, the model will most likely get stuck in local minimums, thus leading to poor results.

To avoid local minimums, the learning rate is often scheduled with annealing. Thus starting with a high learning rate and gradual reducing it, but at some point resetting it to exit local minimums. This is commonly done by a decreasing function of some sort (trigonometric, polynomial, exponential) combined with a reset function.

In addition to learning rate scheduling, it is commonly added an extra term to the updating equation (2.23) that represents previous updates of the weight to include the general direction the weights are updated (2.24). This is known as momentum and is implemented as a scalar momentum term $\gamma < 1$ and the previous weight update v_t. The momentum allows the network to avoid shallow local minimums as consecutive gradients in the same direction will compensate for a small gradient generated at a local minimum. The momentum term is similar to the physical concept of inertia as it requires more to change the motion of a heavy object in contrast to a light object.

$$W_t = W_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial W_t} + \gamma v_t \tag{2.24}$$

$$v_t = \eta \frac{\partial \mathcal{L}}{\partial W_{t-1}} + v_{t+1} \tag{2.25}$$

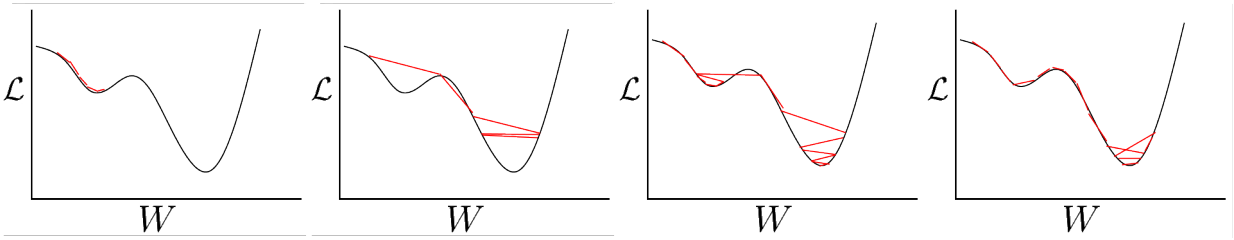


Figure 2.24: An illustration of the effect of different learning rates along the loss surface. The black lines represent the loss surface with a small local minimum to the left and the global minimum to the right. The vertical axis represents the loss $\mathcal{L}$, and the horizontal axis represents the weights W. First: the learning rate is too small and the network gets stuck at the local minimum. Second: the learning rate is too large and the network does not manage to converge in a reasonable time. Third: the networks initially get stuck at the local minimum, but due to annealing, the learning rate gets reset and the network managers to escape the local minimum. Fourth: The network has a smaller learning rate, but due to the momentum, the network manages to overcome the local minimum and converge. The network does oscillate before converging due to the momentum.

Although momentum solves some of the issues with determining a proper learning rate, it introduces a new variable γ that also has to be determined. Too high values of γ results in oscillating behaviours as although the gradient is steep it will be overrun by previous updates. On the other hand, a too low value of γ results in too little impact and the network will get stuck in local minimums. Adaptive moment estimation [75], known as "Adam", is shown to converge faster than traditional SGD [57], [76] and is the common choice of optimiser. Nevertheless, SGD is known for generalising better than Adams and other varieties of Adam [76], [77].

The vanishing gradient problem

For a deep neural network, calculating a good gradient becomes harder as one poor gradient in a previous layer can have a large impact on the current gradient. For small gradients and losses, the lower-layer gradients tend to vanish simply because they become a product of a series of small numbers. This problem is referred to as the vanishing gradient problem and limits the training of deep neural networks as the lower parts of a neural network do not get updated [68]. To counteract the vanishing gradient problem: activation functions not symmetric around zero should be avoided for small initial weights [68], and adding intermediate normalization layers in the model [78].

Overfitting

Overfitting is the situation when the model hard-learning the training data rather than generalizing. Overfitting naturally acquires as a consequence of all machine learning algorithms strives to reduce the loss function. As the network is exposed to the same training data over and over again and do not manages to generalise anymore, it will try to hard-learn the data to minimize the loss function, thus accounting for the noise and outlines in the training data. To prevent overfitting, parts of the data is set aside into a validation set (commonly 20-25 %). The loss is then calculated in parallel with the training data, but no backpropagation. It is expected that the loss from the training data is a bit lower than the validation data due to the natural variance between the datasets. Although, as long as the network is generalising, both losses should decrease. A rising validation loss indicates that the training is gone too far, and the network is starting to overfit. The concept of tracking the validation and training loss is illustrated in figure 2.25.

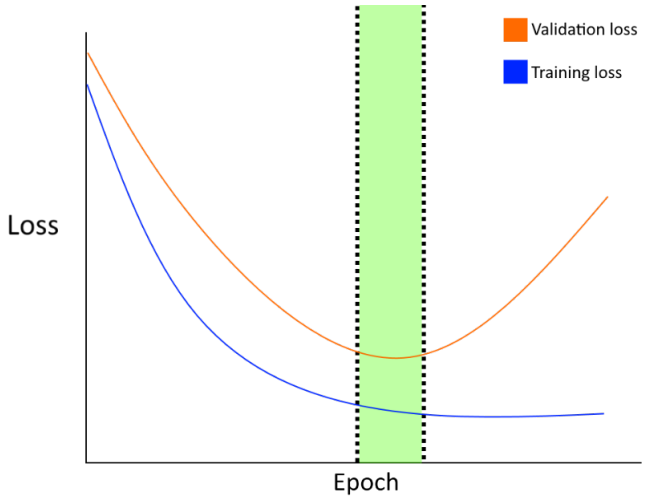


Figure 2.25: The training curve of a general neural network. The validation and training loss declines in the early epochs, but when the network starts to overfit the validation loss increases. The green-shaded area indicates the optimal weights.

Best practice for training

When performing machine learning, the following principles are commonly used and are known to enhance performance [57]:

- Using mini-batches when training rather than the entire training data in one batch.
- Shuffle the data so that the order is changed every epoch, thus the constellation and ordering of the mini-batches does not impact the training.

- Normalize the input as this will help give more balanced gradients.
- Scale the input so that the covariance becomes about the same, thus every input has no bias when starting to train.
- The initial weights should be taken from a normal distribution with a mean of zero to avoid biases.
- The usage of drop-outs makes the network more redundant as it makes decision-based on a larger set of nodes rather than one central supernode.

2.6.2 Convolutional neural network

Convolutional Neural Network (CNN), inspired by the mathematical operation of convolution, is a layer-type that considers the ordering of the input. This allows for the network to evaluate and find local features among the input rather than an entire layer. This idea has shown to give good results in image classifications as shown by the rating on the large image classification benchmark ImageNet [79], [80].

By gliding a small kernel of weights (commonly 3x3 or 5x5) over the original image and forwarding the average value of the Kernel and the overlapping nodes into a new layer, a new image is created that embeds the neighbourhood relations. This new image is often referred to as a feature map as it describes wherein the input a certain feature is located. By learning kernels the networks can aggregate local low-level features like lines and corners up to more complex features like body parts, tools and other objects.

By having this kernel that glides over the input, as shown in figure 2.26, the CNNs acquire a shift-invariance that traditional neural networks do not have. This invariance is what makes CNNs so strong on relation based data (like an image) as a small shift in the input does not impact the output. This relieves the network from having to learn all possible placements of all variations of the object they try to categorise.

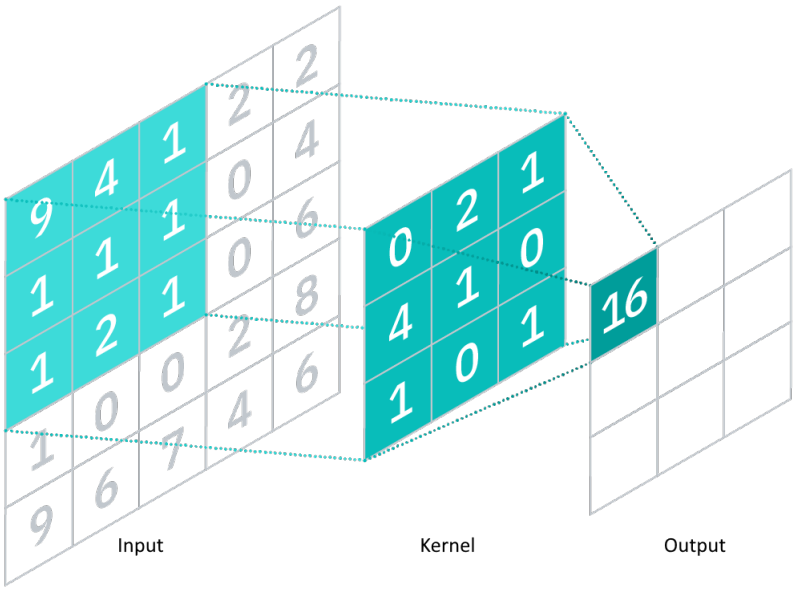


Figure 2.26: An illustration of the workings of a CNN.

Commonly convolution layers have several kernels that each pick up separate fetchers from the previous layer, thus producing several feature maps in parallel that each looks for its learned feature. The common practice for classification tasks is to put several

convolutions layer in series, thus working of each other's output, this is referred to as deep convolution neural network (DCNN). The output of the convolution layers is some complex embedding of the original image. The classification is then commonly done by a separate algorithm, commonly a small neural network.

There are some common design options within CNNs. The following three is the widest spread and impactful elements in any CNN architecture:

- **Pooling** allows for efficient down-sampling of the layers thus reducing redundant information. In an image two adjacent pixels will likely carry more or less the same information, thus making it redundant. Pooling is often performed by taking forwarding ether the max (max-pooling) or average (average-pooling) of a small grid (commonly 2x2 or 3x3) thus reducing the layer size by a factor of m^{-n} where m is the side length of the grid and n is the number of dimensions.
- **Zero padding** is introduced to compensate for the lower impact border pixels gets as they are naturally involved in less of the kernel positions. Zero padding add dummy pixels on the outside with a value of zero so that the kernel has to glide over the original border pixels multiple times. Zero padding can also be added to compensate for the size reduction that naturally accrues when performing convolution.
- **Stride** is a measurement of how fare to move the kernel each time step. By using a stride greater than one, the output size will be reduced similar to that of pooling.

2.6.3 Geometric classification

When working on geometric data, the goal of an algorithm is dependent on the application. There are several terms used to describe the goals, but these are often mixed up.

- **Classification**. The goal is to describe the content of the data, but no additional information about the content (position, size, orientation...) is extracted. This is useful for labelling and separating data within a set like whole pictures or point clouds.
- **Localization**. The algorithm intends to identify the object in the data and assign a label and a boundary box/cuboid, thus defining regions of data points. This is commonly used for self-driving vehicles and collision avoidance as the geometric shape of the objects is not of interest, just what space they occupy. The well known YOLO algorithms work in this manner [81], [82].
- **Segmentation** based tasks aim to cluster similar data point together in an unsupervised manner. This is what is referred to previously as PCS for point clouds. It is mostly used in combination with humans to speeding up the process of labelling data.
- **Semantic segmentation** intends to associate all data points individually with a label. It does not provide any geometric information about any objects in the data, it only intends to classify every point/pixel/voxel. This is the task of PCSS based algorithms. Semitic segmentation algorithms are sometimes just extensions of segmentation based algorithms.
- **Instance segmentation** based tasks identify the objects in the data, extract their geometric data and segments instances of the same category apart. A well-known algorithm for instance segmentation on images is the Mask R-CNN [83]. Instance segmentation can also be obtained by combining semantic segmentation with standalone segmentation.

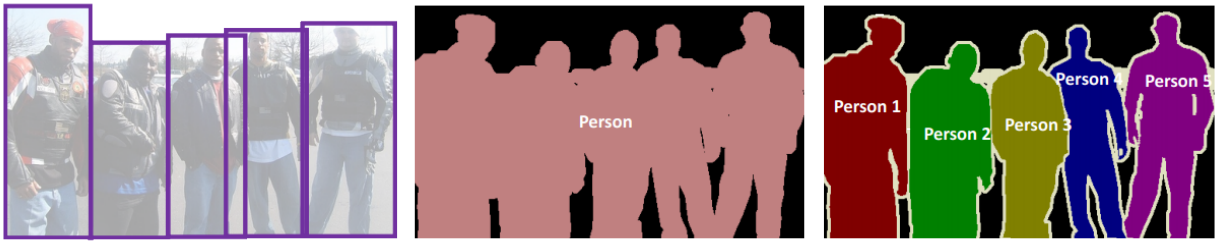


Figure 2.27: Left: Object localisation. Middle: Semantic segmentation. Right: Instance segmentation.

2.6.4 Alternative forwarding

For high-end performance, especially two ideas are shown to enhance the performance of convolution network by partially going away from feedforwards information propagation [84]: Residual connections [85] and Inception architectures [86]. Both are shown to be important in learning on deep neural networks [84], and both have pushed the performance in the ImageNet benchmark [79].

Residual connections

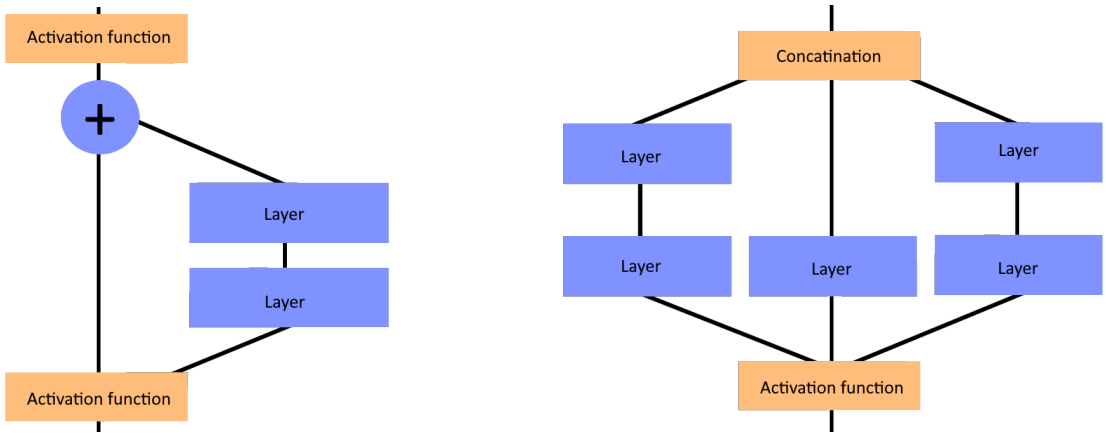
The concept of residual connections, also known as highway networks, is "shortcuts" that skips some layers to forward low-level information (illustrated in figure 2.28a). The low-level information is then fused with high-level processed information, as shown in equation 2.26, the upcoming layer can then extract complex information consisting of features from the high-level and low-level information. The shortcut connections do not add extra parameters nor computational complexity [85], thus the network can still be optimised with SGD-based methods. For convolution-based layers, it is important to maintain the shape of the data to fuse the output and the shortcut data.

$$Y = \phi(X) + X \tag{2.26}$$

ResNet was created by implementing the residual connections in a convolutional network. This allowed for deeper architectures with over 100 layers that perform strongly in various image recognition tasks [85].

Inception architecture

A traditional convolution network is forced to optimise its performance based on the architects choice of kernel size and pooling layers, although it is not known what the optimal size is and when to perform pooling. The optimal architecture might not be directly related to the type of task, but rather with the type of data. In other words, one type of architecture might be optimal for classification on some data, but not all classification tasks. The inception architecture [86], in its simplest form, is blocks that perform parallel forwarding where different kernel sizes and poling layers operate on the input data, and their output is concatenated and filtered. This allows the network to in some sense learn the optimal architecture. The blocks themselves take many different forms as the inception architecture is advanced [84]. This architecture, and iterations of it, show good performance in both image classification and detection on established benchmarks [84].



(a) A residual module.

(b) An inception-inspired architecture.

Figure 2.28: Illustrations of different architecture elements for neural network-based pipelines.

2.7 Deep learning on point clouds

The field of deep learning on point clouds started with PointNet [22], and its successor PointNet++ [87] that have formed the fundament that most modern high-end algorithms are based on. They present the idea of acting directly in the point clouds rather than transforming it into voxels or a stack of images, thus avoiding unnecessary memory usage or information loss [88].

Selected algorithms

The most common algorithms for semantic segmentation on point clouds (PCSS) is often based on one of the following concepts: Multi-view CNN, voxel-based 3D CNN, or Graph convolution neural networks [14], and Set-based [89]. All methods are based on CNNs, but the point clouds have to be transformed due to their unstructured nature in contrast to traditional 2D images that operate in a grid structure. The transformation of data leads to a loss of information that influences each of the methods.

Set-based

Set-based propagation are the basis for PointNet [22] and PointNet++ [87] and the origin method for point cloud processing. This is the only method that operates directly on the point clouds, thus no transformation. Each point is processed by an MLP to obtain a feature vector. The point features are then aggregated with pooling until a global feature vector is obtained. The set-based algorithm benefits form direct processing of the point cloud but are an approach that is not popular as it is outperformed on benchmarks [90] where the strongest set-based contender is Deep sets [91].

Multi-view

First proposed by [92] that used the technique for classification of point clouds, multi-view based algorithms project the point cloud onto artificial plains, thus creating 2D representations. The 2D projections are then individually processed to perform feature extraction. The features form all the projections are then collected (concatenated) and a separate algorithm is used to perform the desired takes. By introducing more views, more of the point clouds information is forwarded, but the 2D representation will not capture all aspects of the 3D model as it is just an approximation, thus geometric data will be lost. For more entangled point clouds with overlaps, the multi-view fall short as it

does not capture the "hidden" points. For large and more complex point clouds it is hard to select viewpoints that best captures the data [14], thus introducing a new optimisation problem and complexity. Never the less, the multi-view based algorithms benefits from being closely related to 2D CNN as several of the devoted concepts within image convolutions are easily transferable. SnapNet [93], performs PCSS by transferring well known 2D techniques and concepts, like SegNet [94], U-net [95] and residual connections, into multi-view based networks.

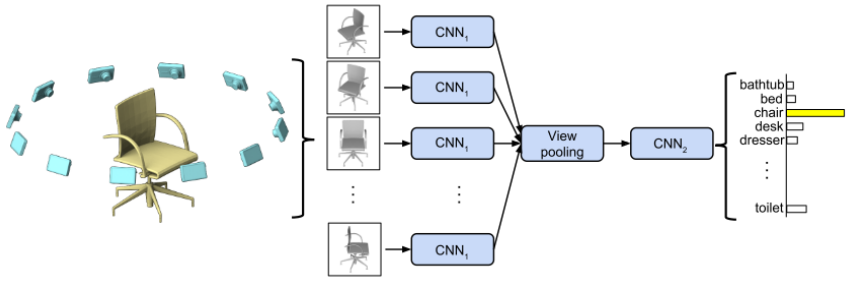


Figure 2.29: From the original paper [92]. An illustration of a general multi-view based system for classification.

Voxel-based 3D CNN

First proposed by [96], the voxel-based methods transform the point clouds into voxel grids, thus ordering the points. A 3D kernel is then swiped over the grid, in the same fashion as in traditional 2D CNNs, to extract features. This method of voxalization solves the problem of point clouds being unstructured, but it comes with three major drawbacks: First, when voxalizing geometric data is lost, especially surface details are easily lost. Second, the voxel grid stores empty space, thus becoming memory expensive when applying high-resolution voxelization. Third, gliding a kernel over three dimensions rather than two increases the computational complexity of each convolution from $O(N^2)$ to $O(N^3)$. Thereby it inevitably becomes a trade-off between having a high-resolution voxel grid to preserve details or having low-resolution voxel gird to reduce the memory usage and computational complexity.

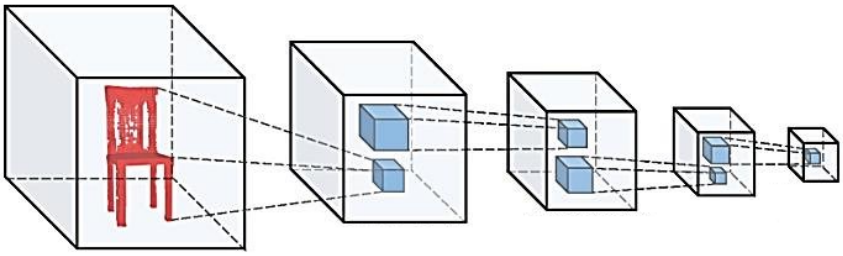


Figure 2.30: An illustration of a voxel-based 3D CNN.

The original VoxNet [96] only performed object classification, while newer models like SegCloud [94] uses 3D convolution to perform PCSS on the voxel grid and trilinear interpolation to assign predictions to the original point cloud, thus performing semantic segmentation. Never the less, SegCloud still suffer from high computational and memory problems [14]. Voxel-based CNN benefits, in the same way as multi-view, from being closely related to traditional 2D CNNs. VV-Net [21] uses a variable autoencoder to embed more local information when voxelising the point clouds, while OcTree-based solutions like OctNet [19] helps improve on memory usage and computational complexity.

Graph convolutions neural networks

By converting the point cloud into a homogeneous graph the neighbourhood properties are conserved in the graph's edges. Closely located points in the point cloud are now connected, thus there spacial relations are preserved. The edges can be both directional or unidirectional, thus allowing for more accurate preservation of relation-data. The most common edge value is the distance between the two original points in the point cloud. Each node in the graph contains the attributes associated with the original point, typically position, colour, curvature, or some sort of engineered feature.

A graph acts as a generalisation of relation-based data, not a specialisation. For instance, an image is just an underacted graph where every node has eight neighbours, and time-series data can be seen as a directed graph where every node has only one incoming and outgoing edge. By converting the point cloud into a graph, the data gets structured, thus allowing for convolution-like operations.

There are two major ways of performing convolution-like operations on graphs: spectral-based convolution and spatial-based convolution [58], [97]. They both work on an update rule for updating the node values thru some learnable parameters. The spectral-based convolution works on a graph level and originates from traditional graph theory. It focuses on optimising a parameterised filter g_θ that is multiplied with the input signal x in the Fourier domain (2.27). $\mathscr{F}$ denotes the Fourier transformation. Implementations of the spectral-based convolution are often computational heavy as they rely on calculating eigenvectors of large matrices [58]. By having a fixed size filter, the spectral-based convolution methods are bounded to a fixed-sized graph, thus strictly limiting their usage.

$$x^{(l+1)} = x^{(l)} \star g_\theta = \mathscr{F}^{-1}(\mathscr{F}(x^{(l)}) \cdot \mathscr{F}(g_\theta)) \tag{2.27}$$

Spatial-based convolution works on a node level and is based on messages passing along edges. At each instance l each node $v_i \in V$ passes its last value x_i^{l-1} to all connected nodes. The node values are updated as a weighted function of the neighbouring nodes. In its simplest form, the update rule becomes the weighted sum of all neighbors passed true an activation function $\phi(\cdot)$ (2.28).

$$x_i^{(l)} = \phi\Big(x_i^{(l-1)} + \sum_{j \in N(i)} x_j^{(l-1)} \cdot \theta_{i,j}^{(l-1)}\Big) \tag{2.28}$$

A small neural network is then added at the end to predict the class for each node $v_i \in V$ based on the final feature vector $x_i^{(L)}$. By acting only on a local scale, the spatial-based methods preserve local features within the graphs and allow for various sized graphs. The different implementations of spatial-based methods differ in their implementation of the messaging and update function [58].

Regardless of spectral-based or spatial-based convolution, the update function should inherit the two flowing properties to be used for semantic segmentation:

- **Permutation-invariance**. When converting the point cloud into a graph, the nodes are assigned a position to form the adjacency matrix A. The ordering of the nodes should not influence the output of the algorithm.
- **Translation-invariance**. Moving a feature in the point cloud (edge, curve, object, etc.) or the entire point cloud the result should not make any changes.

For point cloud applications the proposed Dynamic Graph CNN [98] is a spatial-based graph algorithm for semantic segmentation. It sticks out by recomputing the graph after

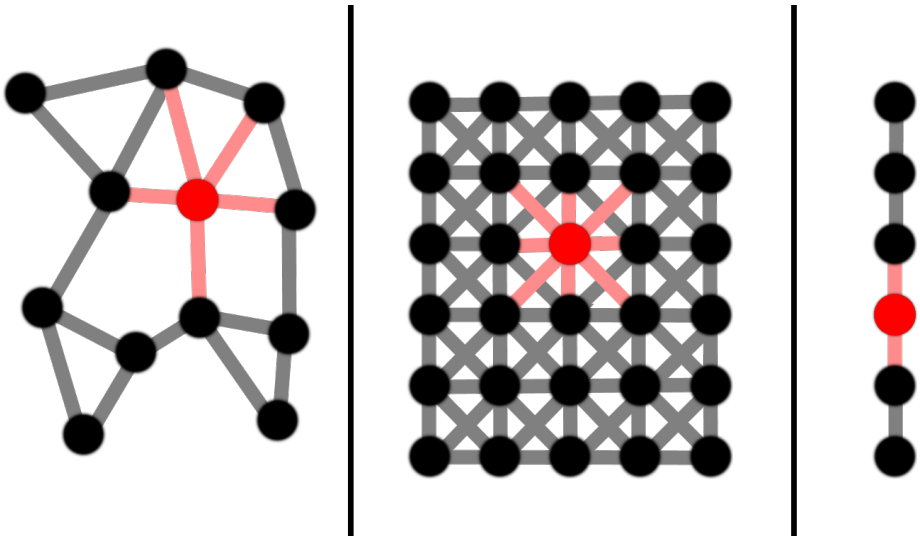


Figure 2.31: An illustration of different instances of graphs with a focused node in red with related edges in pink. Left: A general homogeneous graph. Middle: An image in a graph representation. Right: Time series data represented as a directed graph.

each convolution layer based on the newly achieved feature space (hence making it dynamic). By redrawing the graph similar featured points are connected at each layer of the network. Inspired by residual and inception networks, the output is determined by an MLP working on a concatenation of the achieved features. Another leading algorithm is the Regularized Graph CNN [25]. It is a spectral-based algorithm that intends to update the graph Laplacian matrix.

2.8 Non-deep learning algorithms

Traditional non-deep learning algorithms are in some situations more applicable than machine learning-based ones as they do not require expensive hardware and large amounts of training data. In addition, they are more interpretable, thus making them easier to debug and avoid unexpected behaviour. Depending on the algorithm and problem to be solved, the non-deep learning algorithms can often be faster as well.

2.8.1 Classification algorithms

Classification algorithms intend to assign a label or class to any data point based on its initial attributes. Classification problems are commonly separated into binary and multilabel classification problems that intend to assign a single label to each data point, and multi-label classification problems which allows data points to belong to multiple classes. The most commonly used and highest performing classifiers are Random Forrest and Support-Vector Machines [99].

K-NN

K-nearest neighbours (K-NN) works by finding the k data points in a reference data set and then determining the class by a majority voting. The metric for determining the k-nearest neighbours is commonly euclidean distance. The value for k is the only hyperparameter in a K-NN. The performance is dependent on how general the reference set is and what kind of features are engineered.

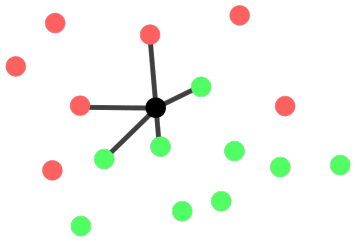


Figure 2.32: An illustration of a K-NN algorithm with $k = 5$.

Random forest

A desertion tree is generated by repeatedly splitting a taring set in two until the new subset only consists of a single class, or it is impossible to split further. The splits are done by choosing one of the features and setting a border value. Choice of attribute and border value is done in such a way that the two subsets become as pure as possible, thus points with the same label goes to the same subset. The formation of the tree is dependent on the training data, and a single tree will struggle to pick up more special cases. Therefore, the training data is randomly split up, and separate trees are made from each part. This collection of trees is whats referred to as a random forest. When performing classification, the data point is run thru all of the trees in the forest, and the predicted class is determined by a majority voting among all the trees.

SVM

A support-vector machine (SVM) uses kernels and hyperplanes to separate data sets. With similarities to linear regression, hyper-planes are created in such a way that they best possible separate the data into two sets by optimising for creating pure sets and maximising the distance to the nearest data point. Kernels are also used to move the data points into higher dimensions, thus allowing for hyper-planes to more easily split the data. A single SVM performs binary classification but can be chained together to perform multilabel classification.

The choice of kernel type must correspond with the shape of the underlying data. The simplest form of a kernel is the linear kernel, but this requires that the data is linearly separable [100]. For more complex data, the kernel type must be able to separate the data. A common kernel, in addition to the linear kernel, is the polynomial based and radial basis function (RBF).

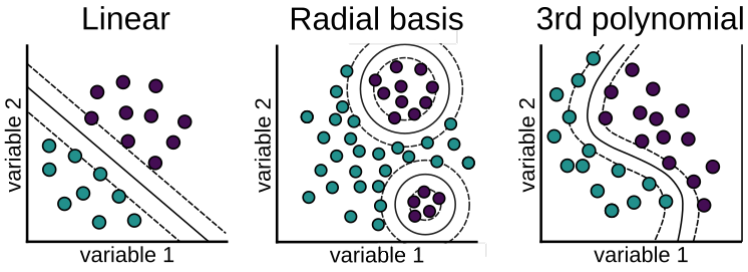


Figure 2.33: Three different kernels. Left: a linear kernel working on linearly seperatebule data. Middle: Radial basis kernel working on clustered data. Right: polynomial kernel of third degree working on non-linearly seperatebule data.

Boosting algorithms

A weak learner refers to a classifier (of any algorithm) that performs slightly better than a random guess, while a strong learner classifies correctly in most instances. For instance, an algorithm that could correctly predict the outcome of a coin flip in 55% of the instances is considered a weak classifier, while a face recognition system on a mobile phone that gets the correct result in almost every situation is considered a strong classifier. Similar to the random forest, boosting algorithms rely on a collective decision from many weak learners to for a strong learner [101]. This work as long as the weak learners fail on non-overlapping parts of the data space as illustrated in figure 2.34.

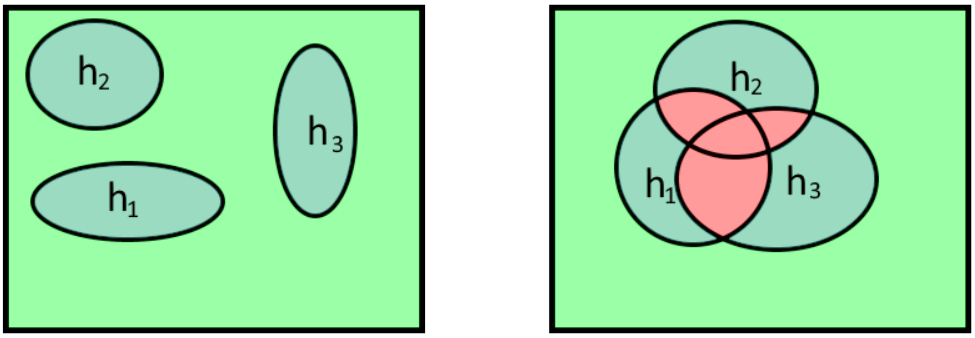


Figure 2.34: An illustration of the voting possess inn a boosting algorithm. The part of the data space where three weak learners h_1, h_2 and h_3 are incorrect is shown with circles. The parts where two or three of the learners vote correctly, thus the boosting algorithm guess correctly is shown in cyan and green. Red areas indicate that two or three of the weak learners were incorrect, thus the boosting algorithm was incorrect. Left: an ideal situation where non of the weak learners overlap, thus the boosting algorithm performs well. Right: the three weak learners overlap thus forming parts of the data space where the boosting algorithm gives the incorrect answer.

The problem of overlapping miss-classification is dealt with by boosting algorithms by recursively constructing the weak learners so that they learn of each other's weaknesses [102]. The first weak learner h_1 learns directly from the training set and is then evaluated whit the same data. When training the second weak learner h_2, the data points are assigned weights scaled according to how well they were classified by h_1. This ensures that h_2 focuses on learning the parts of the data set that were difficult for h_1 (illustrated with circles in figure 2.34) thus preventing them from overlapping. The process is then continued, adding more and more weak learners, until a target is reached.

2.8.2 Clustering algorithms

The task of any clustering algorithm is to group similar data points into clusters, either in a supervised, unsupervised manner or semi-supervised manner. The choice of algorithms are large, and they are based in a variety of different concepts [103], [104]. Some of the most relevant clustering algorithms are:

- **Partition-based.** This kind of algorithms assumes that the centre of data points is the centre of the corresponding cluster. The algorithms are computationally efficient with relatively low time-complexity, nevertheless, they suffer sensitivity to outliers and the need for the number of clusters to be known [103]. Iteratively, algorithms like K-means [105] moves the cluster centre and range until some convergence criteria are met.

- **Hierarchy-based.** By first considering all data points as individual clusters, and then repeatedly merging them until the final clusters are reached. The hierarchy-based algorithms are highly scalable, but suffer from poor time-complexity and need to know the number of clusters [103]. The most known hierarchy-based algorithms are BIRCH [106].
- **Density-based.** The idea is that data in a region of high density within the data-space most likely belong to the same cluster. The algorithms are flexible to the shape of the data but have high memory-complexity [103]. The most common density-based clustering algorithms are DBSCAN [107] and its preciser OPTICS [108].
- **Fractal theory-based**. Based on the mathematics of fractals, the stand for geometry can be divided into several parts which share a common character with the whole [103]. The only commonly used algorithm is fractal clustering [109] (FC).

As described in section 2.4, the time complexity of algorithms becomes relevant when the input data is of large orders of magnitude, as is the case with point clouds. The chosen algorithms are therefore chosen with time-complexity of $O(N \log N)$ or smaller.

The previously talked about algorithms are compared in table 2.3.

	K-means	BIRCH	DBSCAN	OPTICS	FC
Time complexity	$O(N)$	$O(N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$
Fixed number of clusters	Yes	Yes	No	No	No
Scalability	Middle	High	Middle	Middle	High
Sensitive to noise/outliers	Highly	Little	Little	Little	Little
Sensitive to the sequence of the inputting data	Highly	Moderately	Moderately	Little	Highly
Shape of the data	Convex	Convex	Arbitrary	Arbitrary	Arbitrary

Table 2.3: An overview of important aspects of selected clustering algorithms [103].

Clustering metric

Pairs of data points must be evaluated by a metric to determine their similarity or dissimilarity, this forms the basis for how the clustering process intends to separate the data [103]. The choice of metric depends on the application and is, therefore, a design choice. The computational complexity is high for some of the more complex metrics, like the Mahalanobis distance, therefore, only a few selected metrics are included. Since the goal of the thesis is to cluster quantitative data, only distance is used as a metric (dissimilarity), hence similarity-based metrics are not included.

The most common dissimilarity metric is the euclidean distance (2.29), more formally known as the Minkowski distance with a coefficient of 2. The euclidean distance is relatively easy to compute and is intuitive to use as it is the metric used in day-to-day applications. The simpler instance of Minkowski distance (coefficient of 1), the city-block distance (2.30), also known as the taxi distance and the Manhattan distance, operates in a grid-based manner as it compares the two points by summing the element-vice difference. The city-block distance is the simplest form of dissimilarity and therefore also easy to compute. The cosine distance (2.31), also known as the angular distance is widely used in loss functions for unsupervised learning [97]. It is computed by treating the data as vectors, rather than points, and finding the angle between two vectors. The cosine distance only cares for the angel, thus neglecting the magnitude of the vectors. This gives advantages in high-dimensional or unsealed data. The three mentioned metrics are illustrated in figure 2.35.

$$d_{euclidean}(a,b) = \sqrt{\sum_{i=1}^{j} |a_i - b_i|^2} \tag{2.29}$$

$$d_{city-block}(a,b) = \sum_{i=1}^{j} |a_i - b_i| \tag{2.30}$$

$$d_{cosine}(a,b) = \frac{a^T \cdot b}{||a|| \cdot ||b||} = 1 - \cos\alpha \tag{2.31}$$

j = the number of attributes in the data $a,b \in \mathbb{R}^j$

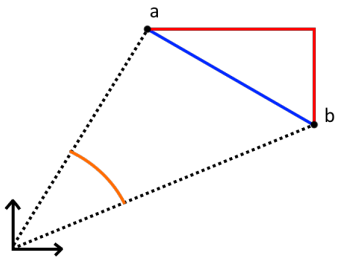


Figure 2.35: An illustration of different distance metrics between point "a" and point "b". Red: the city-block distance. Blue: the euclidean distance. Orange: the cosine distance.

2.8.3 Interpolation

Similar to K-NN, interpolation intends to assign a semantic value to data points based on the distance to data points in a reference set. K-NN is discreet and aims for a majority voting, while interpolation gives a weighted average of the neighbouring points.

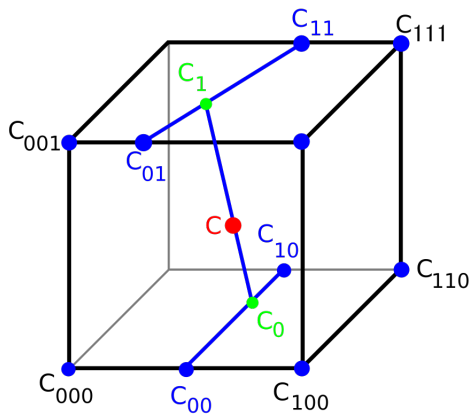


Figure 2.36: An illustration of trilinear interpolation. First fore linear interpolations are done along the first dimension to form C_{00}, C_{01}, C_{10} and C_{11}. The process is then repeated for the second axis to form C_0 and C_1, and third to acquire the value of C.

Classical linear interpolation assumes that two neighbouring points are linearly related, thus the value of a midpoint can be determined by a linear function (2.32). Where p_x is the value of a point at position x, p_a and p_b is the neighboring points, and $\Delta(\cdot)$ denotes the distance to x.

$$p_x = p_a + \Delta a \cdot \frac{p_b - p_a}{\Delta a + \Delta b} \tag{2.32}$$

The relation between the data can follow any mathematical function like polynomial or logarithmic relations. The concept of linear interpolation can be transferred into three dimensions by using the standard Euclidean distance in 3D-pace to interpolate between two neighbouring points. Alternatively, for data in a grid-form, trilinear interpolation can be used. In trilinear interpolation, the value is a point is determined by eight neighbouring points, one from each octet, that is pairwise linearly interpolated along each of the respective dimensions.

Chapter 3

Proposed solution

This chapter describes the working of the cable detection system, including hardware and software setup, and design criteria and design choices.

3.1 Problem definition

Assumptions

The following assumptions are made when developing the system.

- The global position and the outer dimensions of the battery is known. Thus no need to locate the battery.
- The ambient conditions are fixed to that of an indoor industrial warehouse or similar.

Choice of medium

Since this project is a part of a longer chain of possesses for the autonomous dismantling of EV batteries and not a stand-alone process, the input and output of the project must harmonise with the rest of the project. Although several parts of the greater project are not started yet, the ones that are should focus on making the transition from one stage of the disassembly process to another as seamlessly as possible. Hence, the input of this project must be the result of the previous one, and the output of this project should be the basis for the next part of the process.

Figure 3.1: An exploded view of battery A.

As shown in figure 3.1, the logical order of disassembling a battery is to start with the top cover. Before removing the top cover, there can not be any avoidable and relevant information about cables as they are all covered. Hence, there is no other relevant data to pass down from the previous step than the outer dimensions of the battery and its position in the global reference frame.

The logical process to follow after detecting the cables is to disconnect/grab the cables. Therefore, the output of this project should be suitable for model-free grasping. From previews work on the LIBRES project it is stated that a point cloud is the ideal representation for implementing model-free grasping [110]. Therefore, the output should be a point cloud representation of the cables, segmented from the battery.

3.2 Hardware setup

The system developed in this project was implemented in a physical system located in a robot laboratory at the University of Agder. The robot lab was set up with a ROS network connected to two active robots. Only one of the robots was used in this thesis, but the way the network was set up with two robots influenced the project design. The network used a PC running ROScore and a separate PC, connected by DNS-server, running the system used for this thesis. The experimental setup is composed of IRB4400 robot (ABB, Zürich, Switzerland), IRBT4004 track (ABB, Zürich, Switzerland), and Zivid One 3D camera (Zivid, Oslo, Norway) A schematic representation of the setup is showed in figure 3.2.

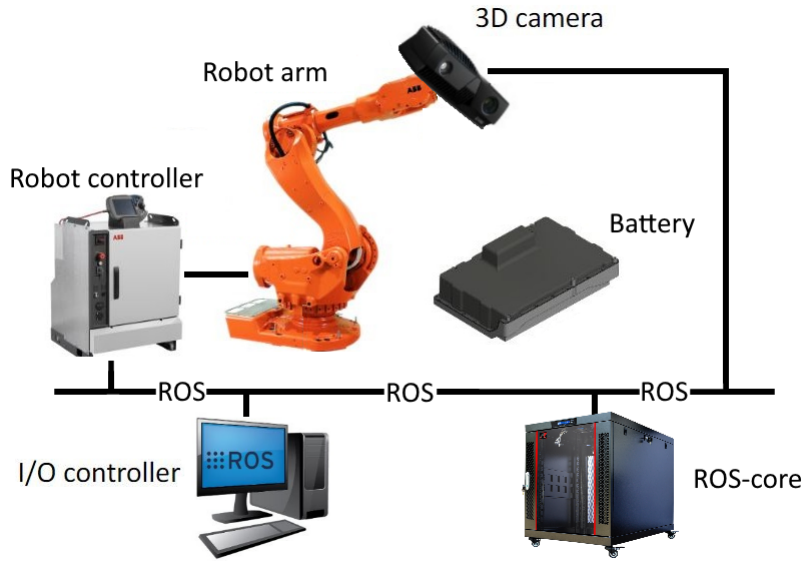


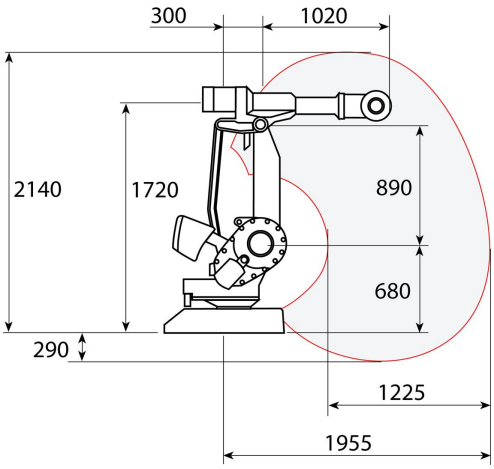
Figure 3.2: Schematic diagram of the lab setup.

3.2.1 Robot manipulator

The robot arm used in this project is an ABB IRB 4400/60 manipulator. The manipulator is a 6DoF robot designed for medium to heavy handling, with a maximum load capacity of 60kg. The IRB 4400 is designed to be versatile, with a focus on precision and smoothness[111]. The robot is a good fit for the project, due to its precision and workspace. It is indeed able to realise precise tasks like unscrewing fasteners while having the necessary working range to access the whole battery.

The robot comes with a pre-made control system (IRC-5 controller). The controller gives ROS the ability to give commands in the form of joint positions. An illustration of the robot and its range of movement is shown in figure 3.3.

(a) ABB IRB 4400.



(b) Dimensions and move range.

Figure 3.3: ABB IRB 4400 dimensions and working range in millimeters.

The IRB 4400 manipulator is placed on an IRBT 4004 track (ABB, Zürich, Switzerland), greatly extending the robot's work area. This workspace extension is essential for the robot to be able to service the largest batteries like Battery B depicted in figure 3.5[112].

3.2.2 Zivid camera

The camera used in this project was a Zivid One+ M (Zivid, Oslo, Norway). The Zivid One+ is capable of capturing at 2.3 megapixels, generating 2.3 million points for every capture. The point cloud has a total of seven values per point, i.e. 3D position(XYZ), colour (RGB) and contrast (C)[33]. The Zivid camera was mounted on the end effector of the ABB robot and worked as an eye in hand system.

The zivid camera uses a ROS service node to communicate with the ROS system. When the capture service is called, the Zivid camera captures a point cloud and publishes the point cloud on the `zivid/points` topic with the `Pointcloud2` message type. The point cloud is then transformed and modified to be ready for the segmentation process.

Figure 3.4: A Zivid One camera.

3.2.3 Test batteries

For this thesis, two battery packs were provided as test battery packs. One smaller Volkswagen battery pack for an Audi A3 e-Tron Sportback (Battery A) depicted in figure 3.6 and one larger Volkswagen battery pack for an E-golf (Battery B) depicted in figure 3.5. The two batteries were used as training material for the object segmentation software and in the physical demo. The reason for having two battery packs was to test how well the model-free callable segmentation worked.

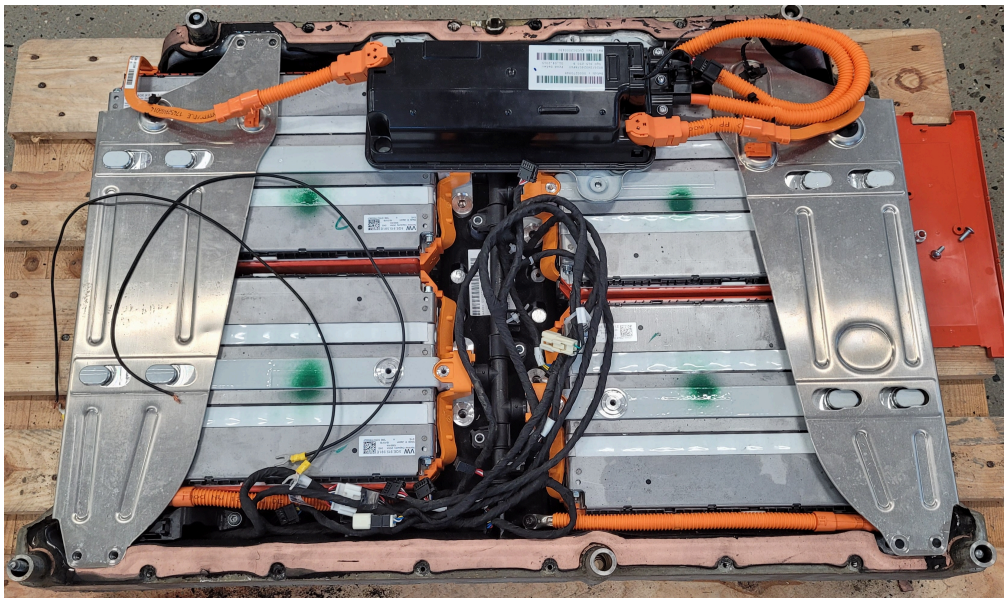


Figure 3.5: The small test battery (Battery A).

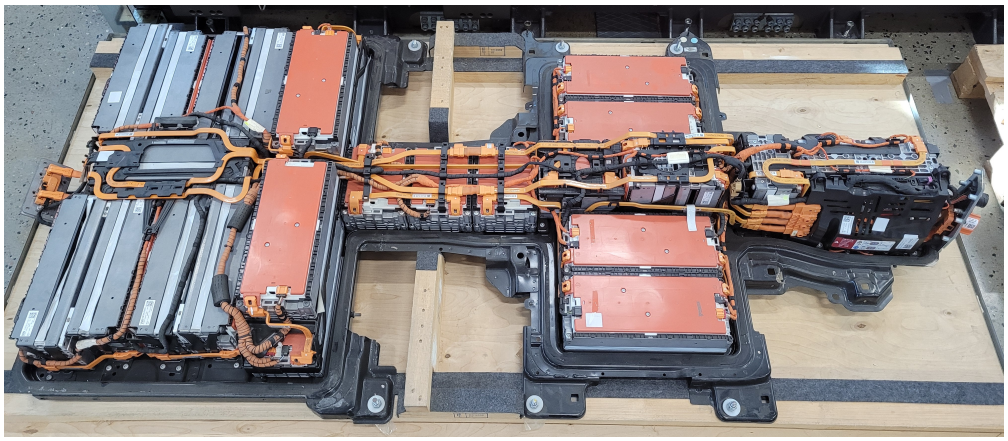


Figure 3.6: The large test battery (Battery B).

Battery dismantling process

The process for recycling a lithium-ion Battery (LIB) starts with removing the battery pack from the vehicle, while the remaining vehicle is sent to its recycling process. The battery pack then needs to be dismantled into its components and modules. The component layout is different for each battery model, making the detailed sequence of dismantling different. The dismantling process described in this thesis refers to the battery packs described in chapter: 3.2.3, but it is generally applicable to other battery packs. The battery packs featured in this thesis can generally be dismantled by the following procedure.

The protective casings are the first components to be removed, exposing the battery modules and electronics. The battery packs can also have plating securing the battery modules, like with the smaller Volkswagen battery pack used in this thesis, which needs to be removed to access the battery modules. After the casing is removed, a process of removing cables and their associated electronics and fasteners can begin. This process can be complicated with many small components that can be hard to manipulate. The components removed from the battery by this step, are likely to be sent to plastic recycling and metal recycling. The next step in the dismantling would be to remove the battery modules. The modules would need to be discharged, to safely dismantle further. The

battery modules contain a large majority of the LIB packs valuable resources, such as cobalt, nickel and lithium.

3.3 Evaluation of path-planning algorithm

When choosing the path planning algorithm for this project, three different solvers were evaluated. The solvers were part of the Open Motion Library (OMPL), which is a path planning library providing sample-based planners, OMPL also comes as the default library in MoveIt. The three evaluated planners were Rapidly-exploring Random Trees (RRT), Rapidly-exploring Random Trees Connect (RRTConnect) and Probabilistic Road Map (PRM).

The way the planners were evaluated was by running the same planning goal ten times for the three planners. The goal pose was a random valid position since there were found no noticeable difference in planning time for different goal poeses. Therefore the tests can be generally applied to the specific goal poses used in this thesis. How long time the algorithms use is dependent on how much time they have available. The tree-based solvers would use a certain percentage of the time, but PRM would use the time available to generate a better road map. Due to this, the planning time was adjusted down to a time where PRM would use more than what was ordered, giving a planning time of 0.05 seconds as the testing time. To then look at the solving times for all the solvers were assumed to be a fair way of comparison The mean solver time from of the evaluation can be found in Table 3.1.

Solver	Mean time @ 0.05 s
PRM	0.0707 s
RRT	0.0734 s
RRTC	0.1388 s

Table 3.1: Mean solver times from testing.

3.4 Data acquisition

The system is designed to be able to generate a high-quality 3D representation of the batteries by utilising the eye-in-hand concept. The resulting point clouds must capture details in both position and colour that later will be used for the perception system. Consistency and good quality of data are key elements for good performance of any machine learning system as the computed gradients then point more towards the optimal solution [57], thus a more convex loss-surface.

The data acquisition system is divided into sub-systems that perform their dedicated task. The sub-systems are developed with the intent of being fast and general to not be limited to only be used by robots. In an industrial setting, the data acquisition system can likely serve several dismantling stations and not be limited to a stationary working space. The workflow of the data acquisition system is shown in figure 3.7.

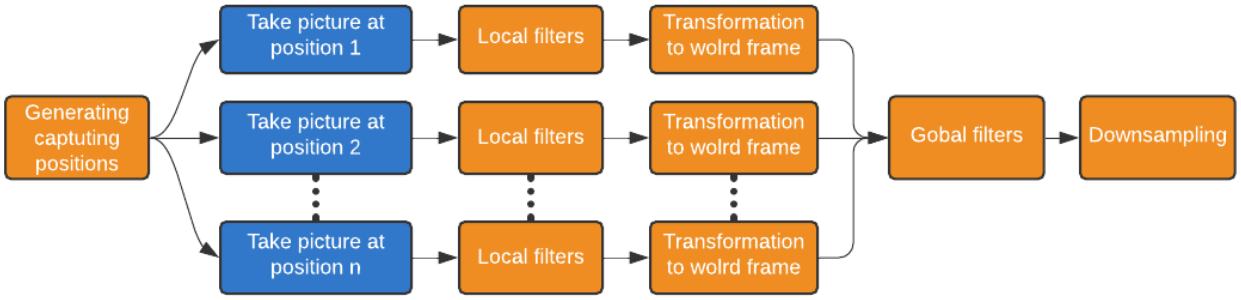


Figure 3.7: The work flow of the data acquisition system.

3.4.1 Point cloud representation

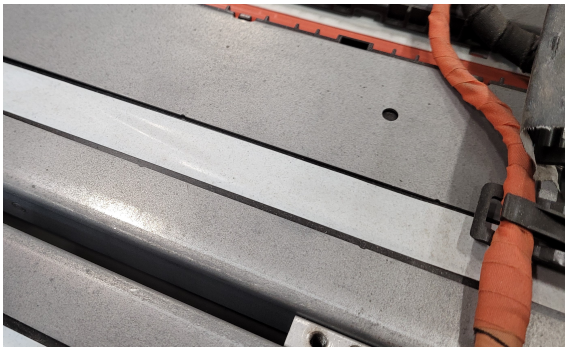
The process of good quality point cloud representations of the battery packs can be split into three problems. The first problem is how to correctly configure a structured light camera to capture an as complete point cloud as possible. The second problem is how to position the camera to get a good representation of a 3D feature. The last problem is how to guaranty that the point cloud that have been generated is a "close to complete" representation of the battery pack, with no regions missing.

Capture settings

The battery packs consist of parts that are affected by the structured light technology in different ways. When taking a 2D picture, the settings that are used can affect the quality of the picture. A picture can be too dark, making it hard to see the black plastic parts or it can be too bright, saturating the brighter parts of the battery. Since a structured light sensor uses visible light as input, the same problems seen in 2D capturing is applied to 3D capturing as well. Due to the camera not having an infinite dynamic range, information is lost when a region becomes too dark or too bright, the structured light system is not able to extract accurate information of the placement of the pixel, giving instead a "Not a Number" value (NaN). These NaN pixels can be a considerable problem for a model-free battery dismantling plant, due to the robot colliding with parts invisible to the system.

The battery packs have components that span a wide range of brightness. As can be seen in figure 3.2.3 the battery has components such as the orange and black cables, the white stickers and markings, the metallic platings and the black plastic. To give a complete representation all of these components would need to be given special attention. The dynamic range of the Zivid camera has a good dynamic range equal to 23 stops[45], however, the range is not extensive enough to cover the whole dynamic range present in capture of the battery packs, to focus on components in a certain range is therefore necessary.

Upper range features The brightest parts visible on the battery packs are the informative stickers and the white strips along with some components, as can be seen in figure 3.8. To represent such parts the dynamic range would have to be adjusted down in overall brightness, giving a dark image, where non of the white parts is saturated. The black plastic and wires would however be poorly represented by such settings, largely not appearing in a point cloud. The orange cables would likely still be visible since they are mostly higher in the dynamic range.



(a) White tape placed along battery modules.

(b) Stickers with component information.

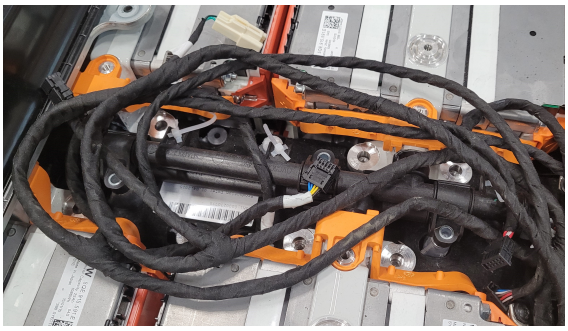
Figure 3.8: Example of features which demands a low to not be saturated.

Mid range features As seen from figure 3.9, many cables come in various shades of orange, as well as some coverings. The cables encompass most of the mid-range of the dynamic spectre, therefore to get good representations, one should also capture with settings where these parts are well exposed. An image captured with this range would likely be the most "normal" looking image.

Figure 3.9: Example of the dynamic range of various orange components.

Lower range features The components which are in the lower dynamic range, as depicted in figure 3.10 need a high brightness capture to be visible. However, when the brightness is increased, it is to be expected that most other features higher on the dynamic scale, will be saturated or overexposed.

(a) Black plastic box.



(b) Bundle of black cables.

Figure 3.10: Example of features which demands a high brightness to be represented.

HDR

High Dynamic Range (HDR) capturing is a technique used to extend the dynamic range of an image. By combining the three captures into one capture. The results give a capture that covers the whole range of exposure in the test region. With HDR one can therefore combine the three dynamic ranges discussed in the previous chapter into one capture, with non of the drawbacks of focusing on a set of features.

3.4.2 Camera pivot script

For easy camera positioning a script was created to pivot the camera around the battery, positioning it such that it is always pointing towards the battery when specifying a certain angular position. The script needs to have the position of the battery, aka the pivot point of the camera, then it requires the desired distance from the pivot point. Lastly inputting the angle from the vertical axis, and the rotation around the vertical axis outputs the corresponding tool position and orientation. Figure 3.11 is an illustration of what the Camera pivot script is intended to do.

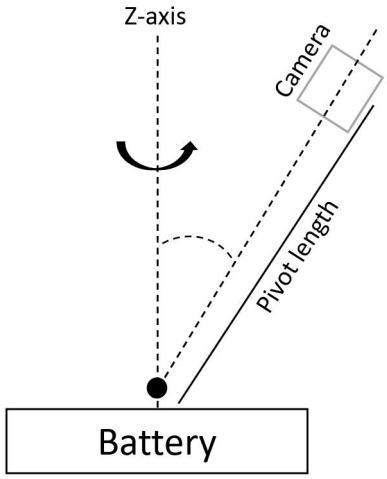


Figure 3.11: Battery pivot illustration.

3.4.3 Point cloud stitching

To get an accurate representation of the battery, a point cloud from only one point of view is not enough. With one point of view, there will always be surfaces of the battery which are not being included. The batteries also have divots, gaps and tightly packed cables, which can be hard to detect if the vision system only has access to one point of view. The

Zivid camera data sheets also give an optimal distance between 600 to 1600 millimetre, which can make it hard to have the whole battery in the frame, especially when working with larger battery packs. Since the setup in this thesis is an eye-in-hand system, the possibility of generating point clouds from several different points-of-views is an option. By using the camera pivoting system from Chapter 3.4.2, one can systematically represent a feature from several different angles and then stitch the point clouds together into one.

To stitch the point clouds together the point clouds are transformed to fit in a common world frame. ROS implements this transformation thru the library TF which keep track of positions and orientation of the position of all the frames in the system, including the camera frame. The system acquires the transformations by setting up a transform listener and requesting the lookup of the transformation between the `"world"` frame to the `"zivid_optical_frame"`. The point cloud is then multiplied, pointwise by the transformation matrix. The transformed points are then added to the list of points from the previously transformed point cloud, fusing them to one point cloud. All point cloud used in the training data and the point clouds showed in the perception and clustering results of Chapter 4.3 and 4.2.

3.4.4 Capture sequencing

When designing an end-to-end model-free dismantling plant the problem of incomplete information is one of the main challenges of the system. If the capturing service were not able to cover the whole battery pack, the system may not be able to complete the dismantling, due to the system's lack of knowledge about certain components, or vital parts of the dismantling process for the component. A consistent way of acquiring an as close to complete representation of the battery pack is essential for a consistently functioning LIB dismantling plant.

Cable representation

When judging what good cable representation is, there are two main considerations. The first consideration is how suited it is for the cable detection and segmentation software. The second considerations are to the grasping software since it is desirable to use the same point cloud for the grasping as was used for the detection and segmentation.

For the detection and segmentation software to work well, it's important for the point cloud to have continuous representations of the cables. This is so the segmentation software can define the whole cable as one. The cloud also needs to be fine and precise enough to detect small details and variations in the cloud, features such as slight colour changes and small fasteners can be important. The grasping software will be more sensitive to how complete the representation of the feature of interest is. To work optimally it can be critical for the grasping software to have a model of the whole surface of the cable as part of the point cloud. The whole surface would give the grasping software the widest range of grasping options, and faulty grasping plans would not occur as often. To capture a fully complete representation of the cables is for the most part impractical, the system would therefore need to work with some form of "as near to complete as possible" representation of the cables.

Point-of-view

In this thesis, it is assumed that capturing from a single point-of-view (POV) is not enough to get a desirable representation of the cables. The main reason being that a potential

grasping software would have problems, at best only working with a half-cylinder representation of the cable. The detection software would also be likely to suffer a performance decrease, due to the reduced amount of cable surface it would use as the basis for detection. It is natural to think that with more POV's the representation would be better, but due to the time needed to capture, the optimal number of POV's is an optimization problem between point cloud quality versus capture time used.

Full battery coverage

By extending the POV line of thinking to the whole battery pack means that every surface should likewise be covered by three POV's. Since the shape of the battery is not known beforehand, one can make a rough estimate of how many captures are needed, by assuming the battery to be ap plane with a certain area. One can then systematically capture the surface to a certain degree of completeness, by pivoting around one point, capturing several times, then move over and repeat the same process for a new pivot point. To not miss any regions of the battery pack, the system would need to know what distance it can move without skipping over any region. However, the system should not take too small steps either, wasting time capturing regions already captured.

The Zivid One+ M camera, which is used in this thesis, gives a field of view of $691 \times 432 [mm]$ at a $1000\,mm$ distance. Which gives a surface coverage of approximately $0.29\,m^2$ per capture. It would however be a large variation in how much surface one capture acquires, due to the angle and height of the camera, as well as the elevation of the battery pack feature. Therefore one can likely not trust the whole field of view to give consistent coverage. To estimate how many captures are needed, the following equation was devised.

$$N_{cap} = \left\lceil \frac{l_{bat} \cdot w_{bat}}{(l_{cap} \cdot w_{cap}) \cdot \eta_{cap}} \right\rceil \cdot N_{POV} \tag{3.1}$$

Where the l_{bat} and w_{bat} is the length and width of the the battery pack, l_{cap} and w_{cap} is the length and width of the capture surface coverage. η_{cap} represents how much of the coverage can be expected to be consistent. N_{cap} is the required captures and N_{POV} is the desired POV's. The ratio between battery pack and capture surface is rounded up to nearest full number, representing the number of pivot points is needed.

Next-best-scan

A general approach is not guaranteed to be able to handle all the battery pack geometries necessary. Features like cracks and partially obstructed features can be overlooked. It would therefore likely be necessary to focus on problematic regions as they show up. An approach that might be more successful at covering the whole battery pack surface is to use a next-best-scan algorithm. Next-best-scan is an algorithm that calculates what POV would be best for uncovering the largest amount of unknown space. Such a system could be used more efficiently cover, by covering more battery pack surface per capture and be able to focus in on problematic features.

3.4.5 Position accuracy

When generating the point clouds, there is a certain limit of how detailed the point cloud can be, before the inaccuracy of the hardware interferes. In the setup created for this thesis, there are three sources of inaccuracy. The robot, the camera and the sensor noise. In this thesis, a hypothetical worst case, based on the specs from the datasheets, was used to find this limit.

- The angular sensors of the ABB robot and track always have some amount of noise. However, this noise is accounted for in how the TF library handles all transformations. When TF looks up the transformation between the camera frame and the world frame, it records the average signal over a time defined in the script, effectively removing the majority of signal noise.

- The positional accuracy of the ABB robots is reported in the datasheet provided by ABB. The worst-case absolute accuracy is reported to be $0.70\,mm$[113]. The positional inaccuracy of the robot is a product of the compiling angular inaccuracy of each joint. The orientation inaccuracy of the robot amplifies the positional inaccuracy of the points in the point cloud, due to the distance between the camera and the battery.

  A simple approximation of the robots angular inaccuracy is done in this thesis by using the maximum reach of the robot as the adjacent side of a triangle and the positional inaccuracy as the opposing side. By some simple trigonometry, showed in equation 3.2, an angle can be found. θ_E is the angular error, P_E^R is the worst-case position accuracy of the robot and L_{max}^R is the maximum reach of the robot, given as $1.9550\,mm$ in the datasheet [111].

$$\theta_E = tan^{-1}\left(\frac{P_E^R}{L_{max}^R}\right) = 3.58e-04[rad] \tag{3.2}$$

  With the assumption that all of the joints have the same angular accuracy, the accuracy of the robot is calculated as if all the inaccuracy is in the last joint. The added inaccuracy of the camera to battery distance can be calculated with the following trigonometric function. Where $\hat{P}_E^R$ is the approximated accuracy of the robot and camera setup. L_{Bat} is the length from the camera to the battery, in this calculation, it was set to $1600\,mm$.

$$\hat{P}_E^R = L_{Bat} \cdot tan(\theta_E) = 0.57[mm] \tag{3.3}$$

- The camera also gives the system some inaccuracy. From the data sheet of the Zivid One+, the accuracy at a distance of $1600\,mm$ is given as $1030\,\mu m$ [33]. This is added to the accuracy approximation to give the following accuracy, where P_E^C represents the inaccuracy of the camera.

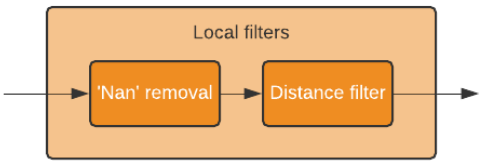
$$\hat{P}_E^{tot} = \hat{P}_E^R + P_E^C = 1.57[mm] \tag{3.4}$$

Due to how the inaccuracy of the robot manipulator has been calculated, the result can not be expected to be an accurate assessment. The given inaccuracy of $0.7\,mm$ is a worst-case, something which is only assumed to be accruing at the edges of the robots working range. The accuracy could therefore be lowered, but in the worst case, a voxel grid with approximately $1.6\,mm$ sized elements is small enough to give minimal information loss. To reduce the size further would therefore likely have little benefit and reduce the margin of error in the voxel grid.

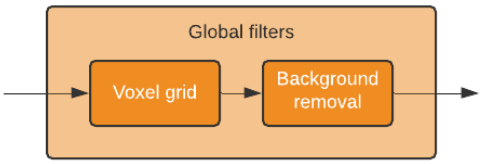
3.4.6 Filtering

Filters are applied to remove unwanted data and transforming the reaming data into one common format. The filtering process is divided into two parts: local filtering, shown in figure 3.12a, and global filtering, shown in figure 3.12b. Due to the nature of structured light cameras, every point is put into a grid format, some of the points will therefore come out without a value because the camera did not receive any reflection. Hence, the point cloud contains a substantial amount of empty points stored with the value 'Nan'. These

points do not have any attributes, thus are not to any use, but takes up space in the memory. From testing, around 30% to 50% of the points in the point clouds captured are empty. These points are excluded when reading from the `zivid_camera/points` topic in the ROS network and are the first step in the local filtering.



(a) The local filters.



(b) The global filters.

Figure 3.12: Flowchart for filters.

Only points within the recommended working range of the camera should be used in order to minimise the inaccuracy in position, thus the assumptions made in section 3.4.5 remains valid. For the zivid one camera used in this setting, the operating range is sett to $[0.6, 1.6]$ [33]. The filtering is preformed by a pass thru filter from the `PCL` library.

The local filters are done by a single loop over the data as both filters only check a single condition related to the intrinsic attributes of each point, thus they have a time complexity of $O(n)$.

After combining the individual point clouds there are a vast number of redundant points as several objects are appear in multiple of the raw point clouds. Since the same point from different frames will not have the exact same coordinate due to the inaccuracy of floating-point representations. In addition, there is inaccuracy in the calculation of the transformation matrix as described in section 3.4.5. A voxel grid with small voxels is therefore applied as closely related point are represented as a single point. Performing a voxelization of a point cloud involves looping over the entire point cloud and checking one condition related to the intrinsic attributes of each point, thus being linear with a time-complexity of $O(n)$.

The background is removed since the outer dimensions of the battery is assumed to be known. This is done in the same way as the distance filter, namely by using the `PCL` library's pass-thru filter. The point cloud is then cropped in all three spatial dimensions to fit inside a cuboid with side lengths equal to the outer dimensions of the battery. Especially pictures taken from the side tend to pick up a lot of the background. By the same arguments as previously used, this filter also has a time complexity of $O(n)$.

A statistical outlier filter could also have been used, but thees require extrinsic attributes like density to be calculated, thus having higher computational complexity and are therefore not used. Most outliers will be removed thru the background filtering, in addition, both the classification algorithm and the clustering algorithm will reduce the influence of outliers. It is therefore deemed as too costly to implement an outlier removal filter as the increase in computational time is substantial in comparison with the potential performance gain.

3.4.7 Labeling

Most labelling tools for 3D point clouds are intended for automotive usage where object detection is the goal of the algorithms. Therefore, software from well-known sources, like MATLAB's "Ground Truth Labeler", which are RoI-based (region of interest), is not feasible as they do not allow for individual point labelling. The solution is to use open-source software. In the end, a JavaScript-based software named the "Semantic segmentation editor" [114] was chosen as it fulfils the following requirements:

- Visualisation of large point clouds with colours.
- Support for `.PCL` file format.
- Allows single point labelling.

The labelled point clouds are exported as `.PCD` files with the optional field "label" in addition to "x", "y", "z" and "RGB". Each point in the exported point cloud is then assigned a binary label where 0 is associated with background and 1 is associated with a cable.

(a) Points with label "0". (b) Original point cloud. (c) Points with label "1".

Figure 3.13: An example of a labeled point cloud.

As shown in figure 3.13, the point cloud labeling is not perfect. This is a result of manually labelling, and complicated geometry to be labelled. The complicated geometry is a result of cables not being simple geometric shapes. Cables have the same colour as parts of the background, they go around corners and they are positioned adjacent to the background rather than floating in the air. As shown in image 3.13a and 3.13c the mislabeling is bidirectional. Since the mislabeling happens by human error and not by any external source, it is assumed that the rate of mislabelling is similar for both classes, thus not providing any bias towards any of the classes. Nevertheless, the mislabelling results in noisy training data that will increase the amount of training needed for any neural network-based model to converge, in addition, to reduce the true accuracy of any model.

3.4.8 Training data

When capturing point cloud data for the training, a total of 22 point cloud were captured, and 16 were generated by various transforms of the existing point cloud. Table B.1 lists the data talked about in this section. 38 point clouds may sound like a small amount coming from the perspective of 2D images detection, the total points in the data set were over 26 million points. The point clouds were captured with a varying amount of POV's, varying between two, to eight POV's. The capturing was also done with two sizes of the voxel grid. The first eight were captured with a $2.5\ mm$ voxel grid, whilst the rest were captured with $1.5\ mm$.

The reason for using the 2.5 voxel grid, was to see if the system were able to detect cables with less data. However, after a few captures, the rest were done with a 1.5-millimetre grid, since that is closer to what the camera can output. The transformed point clouds were generated to embed rotational invariance.

The quality of the point clouds was however not optimal during the first 16. This was due to them being captured with the HDR settings before the problem described in Chap-

ter 4.1 were discovered and worked around. The point clouds were however usable as training material, giving a large amount of variance in the point cloud quality.

3.5 Perception system

The purpose of the perception systems is to process the acquired point cloud to extract the information needed to perform disconnection of the cables. The system is composed of several blocks that perform designated operations on the initial data to reach the end goal.

3.5.1 Constrains and design criteria

Most applications for segmenting 3D-point clouds are towards navigation of either an indoor robot or a car driving in the street. Thereby the point clouds are taken from a single point of view, meaning that they only represent the environment from the sensors point of view, hence being incomplete and containing blind spots as illustrated in figure 3.14. Methods like complex YOLO [81] and other real-time orientation methods, are meant to work on this first person-view data as they are intended to sense the world through a sensor (GRB-D or LiDAR) mounted on the moving construction. These algorithms are developed to constantly update the surroundings and are therefore developed with a focus on processing speed (Commonly $0.1 - 0.03[s]$) rather than high classification accuracy ($< 83\%$ [28]). The low accuracy is compensated for with a high update rate [81].

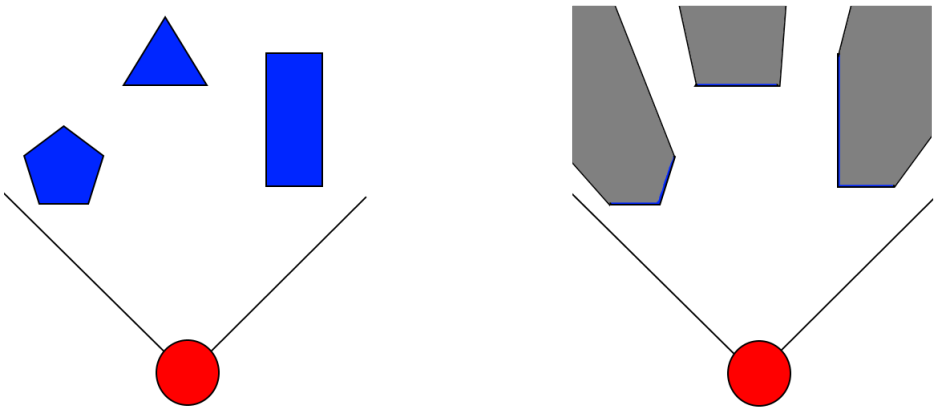


Figure 3.14: An illustration of first person-view. Red: an observer with a fixed field of view. Left: the ground truth shape of the observed objects. Right: the objects observed form a first person-view with blind-spots shaded in gray.

In this application, the robot will use some time to disassemble the battery due to the number of parts and the complexity of the disassembly process. There will not appear any new object within the workspace by removing one detected part (all new parts will be under the removed part), thus a single representation is valid until all detected parts are removed. By constructing a single representation of the battery, the robot, with its encoders and known position, can blindly manoeuvre around the battery. Therefore computing a single representation of the battery is valid for a longer time, thus the processing time does not need to be considered real-time as a processing time of a few seconds does not drastically impact the overall duration time for disassembling a single battery. Therefore this approach will use the time to generate a third person-view-based point cloud, thus operating on a more complete representation with smaller and fewer blind spots. This more complete representation is also usefully later for grasping or interacting with

any of the objects.

When choosing the architecture for the perception system the following are set as design requirements to be applicable:

- The end product should be labelled point clouds of the cables as this is optimal for running grasping algorithms, hence the perception system should perform instance segmentation of the original point cloud.
- The systems should be permutation-invariant and translation-invariant to the input point cloud to be model-free and hardware-independent.
- Accuracy in classification is valued over computational speed as the system is intended to acquire only one 3D model before disconnecting cables. Acquiring a good model of the cables is valued over reaching real-time computational speed.
- Computational speed should be reasonable as it is intended for industrial usage. Hence less than 10 seconds.

Instance segmentation

The process of instance segmentation of point cloud is often reshaped into multi-stage problem [32], [115]. Ether the process is deconstructed into a clustering problem followed up by a classification of the clusters (cluster-class), a semantic segmentation problem followed up by a clustering problem (PCSS-cluster), or simplified to a localization problem and setting all points within the cuboid as the same instance.

Simplifying the problem to a localization problem works well when the instances are separated in space. Therefore, the Mask-MCNet [32] that is designed for identifying teeth can utilise this method. For point clouds containing more nested objects, or the objects are close in space, this method is not feasible as several cuboids might overlap.

The two other methods are mere opposites of each other, differing in whether to cluster or classify first. By using cluster-label the algorithm gets drastically less training data as the number of clusters is significantly less than the number of points. The classification process also becomes harder as it must account for more complex features related to the entire point cloud rather than single points. A PCSS-cluster system does not need to have the same level of accuracy because mislabeling edge cases does not stop the forming of clusters. Thus the resulting instances will be more robust with a PCSS-cluster-based pipeline.

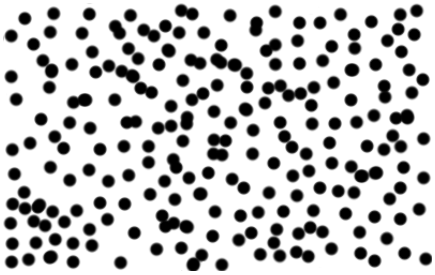
Due to the limited amount of training data, thus challenging the training, the PCSS-cluster pipeline is chosen. This is similar to what is done in other papers [115].

3.5.2 System work-flow

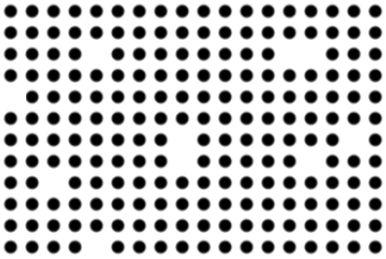
The overall workflow of the perception system is illustrated in figure 3.15 and is composed of four blocks that start with a raw point cloud and have the end goal of instance segmentation.

Filtering and downsampling

As shown in figure 3.15a, the raw data is noisy and bias, and will therefore make it more difficult for any algorithm to perform classification [57]. Therefore, the point cloud is first

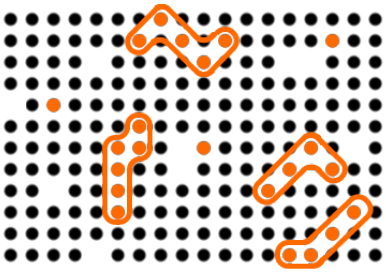


(a) The raw point cloud.



(b) The point cloud is filtered and downsampled.

(c) The points are assigned a semantic value.



(d) Clusters are formed.

Figure 3.15: The workings of the perception system.

filtered to remove invalid points, background and redundant points by the processes described in section 3.4.6.

The Zivid camera provides over two million points, which is more than a standard LiDAR and magnitudes more than what most algorithms are designed to work on (Commonly $< 10,000$) [14], [28], [30], [31]. Although algorithms are scaleable, they have at best a time-complexity of $O(n^2)$, thus the input size must be reduced to meet time requirements described in section 3.5.1.

Downsampling results in information loss, thus the choice of method for downsampling affects the performance. Downsampling is either done by random sampling, voxel-grids or set-pooling as done by [89]. Random sampling is the most computationally efficient, but will not give a uniform distribution of the output, thus some regions will be over-represented while others get underrepresented. Small objects and surface details may be lost if the downsampling is of substantial magnitudes. Random downsampling also introduces uncertainty into the algorithm as downsampling the same set several times will result in a different outcome. This increases the amount of training data but makes it harder to train as the algorithms might not be exposed to important details.

The system uses a hybrid solution where first a voxel-grid is applied to remove redundant points. The point cloud is also downsampled as a result of the voxel grid, but the voxel grid alone can not give the desired size. To reach the desired size, a random reduction filter is applied after the point cloud is voxelised. The data now take a grid-like shape with some empty squares as shown in figure 3.15b. The usage of a random reduction filter is justified by having the voxel-grid performing the majority of the downsampling, thus the disadvantages of a random reduction filter are minimized.

Semantic segmentation

As the first step in instance segmentation, semantic segmentation aims to label the points in the downsampled point cloud as shown in figure 3.15c. The semantic segmentation is likely to be the most time-consuming part of the pipeline as most popular techniques have a high time complexity. Voxel grid-based methods grow cubical with respect to the

resolution [22], and graph-based, in the best case, grows quadratic, [58]. More novel and traditional approaches like Random Forrest, SVM and PointNet have drastically lower time complexities [22]. Nevertheless, the novel approaches are outperformed by more complex ones in the large benchmarks [28], [31], [90]. Boosting algorithms have been used for semantic segmentation of point clouds from outdoor areal data with promising results [116], [117]. Therefore, gradient boosting is included as one of the proposed methods.

Benchmarks are directed towards accuracy, and not time usage, therefore there does not exists any known comparison of time complexity or forward time versus accuracy across several approaches. Algorithms based on the principles of the following promising architectures are implemented and tested to determine a stubble algorithm:

- Dynamic Graph CNN (DGCNN)[24]
- VoxelNet [118]
- PointNet [22]
- Random Forrest
- SVM
- Gradient boosting [119]

In some of the chosen algorithms, only a subset of the points is labelled. Therefore, it is needed to assign a label to the remaining points based on a subset of labelled points. The traditional way of doing this is a K-NN approach as described in section 2.8.1.

Inspired by SEGCloud [120], Mask-MCNet [32] and Point-GNN [89], trilinear interpolation, as described in section 2.8.3, is proposed to be used to transfer the learned label back to the original filtered point cloud. Trilinear interpolation allows the data to be a weighted average of the neighbouring points, thus providing a more continuous interpolation than the alternative, K-NN.

The well-known successor of PointNet, Pointnett++, is excluded as it is shown to have a vastly poorer time usage than DGCNN [98]. The two well-performing algorithms SEGCloud [120] and RandLA-Net [121] are not included. The choice of algorithms and there implementation are described further in detail in section 3.5.5 and section 3.6.

Clustering

After the points are assigned a semantic label, they must be grouped, also known as clustering, so that a collection of points can represent an object. In addition, a semantic segmentation algorithm will never truly reach 100% accuracy, hence there are mislabeled points that now become outliers and thus should not be included in any cluster. In other words, the clustering block, as shown in figure 3.15d, should form collections of points that represent cables and leave out mislabeled points. The choice of clustering algorithm is described further in section 3.5.6.

3.5.3 Graph forming

A k-NN graph based on coordinates is the most intuitive way of representing a point cloud as a graph. The graph will then be formed based on the euclidean distance between the

points in 3D, thus embed the local geometry in the forming of edges. The graph is then homogeneous and undetected as two nodes are not necessarily mutually within each other top k nearest nodes. The graph is commonly also self looping, thus each node has an edge looping back at each self. Self looping is included to simplify the message passing process for convolutional graph neural networks. Optionally, the graph can be formed by a radial search where nodes that have corresponding points in the point cloud which are less apart than a given reference r form an edge. For both cases, forming the graph involves finding the distance between all points in the point cloud, thus being computationally equal. In some cases, it is desirable to have a fixed number of edges related to each node, therefore a k-NN graph is used in this thesis.

Forming graphs is computationally heavy as finding the top k nearest neighbours of any points involves looping over all the entire point cloud. Repeating this process for the entire point cloud then ends in a time-complexity of $O(n^2)$, which is a large time complexity, thus making it challenging to form graphs from large point clouds. By using an octree, a small neighbourhood around each point can be generated so that each point does not have to be compared with every other point, but only the local neighbourhood. Searching an octree have a time complexity of $O(\log n)$, thus reducing the overall time complexity of forming a k-NN graph from a point cloud down to $O(n \log n)$.

3.5.4 Feature engineering

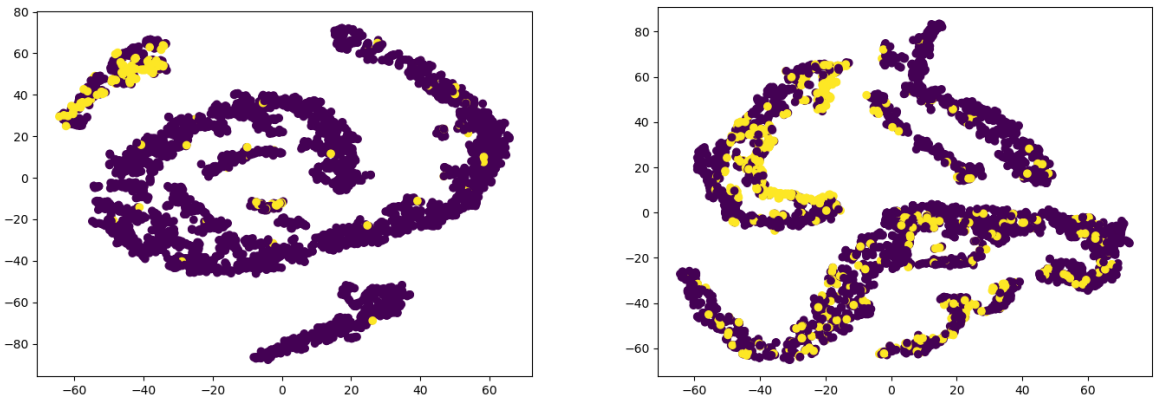
To obtain translation invariance, most of the algorithms can not work directly on the intrinsic features of the points. In addition, for the pure classification algorithms, more features result in more dimensions to distinguish the instances. Using the local surface curvature for points is suggested as an alternative to the absolute coordinates as it is a translation-invariant feature, is easy to implement and have relatively low time complexity ($O(N \log N)$. Since cables tend to have round shapes, they have by nature a lot of curvature, therefor it assumed that using the local surface curvature will improve results while maintaining translation invariance.

As suggested and argued by [57] the features used to train neural network should not be intercorrelated as this makes it harder to determine the impact of any feature. Thus calculating good gradients and training becomes harder. Therefore, for the neural network-based algorithms, only the intrinsic features are used, or the coordinates are replaced with the local surface curvature.

3.5.5 Semantic segmentation algorithm

As described in section 3.5.2 six different algorithms will be tested up against each other. Two of which are non-deep learning-based. The data is reduced to two dimensions to visualise any potential relation between features of the points. t-SNE is chosen over PCA since it preserves the shape of the data points thru out the dimensional reduction [62], [63]. The resulting 2-dimensional embedding are shown in figure 3.16. Clusters of same-labelled points in the embedding indicate that there is some combination of the original point features that can directly be used by a classifier to separate the cables from the point cloud. In other words, the local point features alone contains sufficient information to determine the semantic properties of any point. On the other hand, most geometric data have global features that are greater than the sum of the local features.

From the small battery (Battery A) it appears to be a cluster of labelled points as shown in the upper left corner of figure 3.16a. There are still some yellow points appearing in the rest of the embedding, but these are sparse and does most likely come from noise and



(a) Battery A

(b) Battery B

Figure 3.16: The t-SNE representation of two point clouds, one from each of the batteries. Each embedding is done with the six intrinsic attributes and the three attributes related to the local surface curvature ($\mathbb{R}^9 \rightarrow \mathbb{R}^2$). Yellow dots indicates points labeled as 'cable', while purple indicates 'not cable'.

miss-labelling. For battery B there is a week cluster, but no clear destination as there a lot of yellow dots in all of the embedding. This might be a result of poor labelling as battery B is more challenging to label than battery A.

Traditional classifiers

Although not distinguishable, there are tendencies in the embedding of both batteries A and battery B that there exists some combination of the nine-point attributes that allows for segmentation by using a traditional classifier. The Random Forrest algorithm together with the Support-Vector Machines algorithms are suggested due to their well-known performance [99]. Since the data is not linear distinguishable, several different kernel types of SVM is tested, including the polynomial kernel and the radial basis kernel.

To obtain translation invariance and rotation invariance, the classifiers can not work on the position attributes. A rotated version of any point cloud would have different coordinates for all points, while still representing the same objects. Therefore, the traditional classifiers are trained on the three colour features and the three curvature features.

Gradient boosting

Similar to the random forest algoritm, gradient boosting utilises decision trees to form a collective desertion [122]. It differs in that the decision process is not a majority voting, but rather a sum of votes that is compared to a threshold δ as shown in equation 3.6. For binary classification, the threshold is commonly set to $\delta = 0.5$ to not favour any of the classes. Similar to other boosting algorithms, the gradient boost works by having week classifiers recursively generated based on the miss-classification of the previous classifiers. Similar to other boosting algorithms like AdaBoost [17], the week classifiers are weighted based on their impact as shown in equation 3.5. Gradient descent is then used to determine the weights, similar to how it is done in deep learning, thereby giving the origin to the "gradient" part of gradient boosting.

$$F(x) = h_0 + w_1 \cdot h_1(x) + w_2 \cdot h_2(x) + \cdots + w_n \cdot h_n(x) \tag{3.5}$$

$$\hat{y} = \begin{cases} 0, & F(x) < \delta, \\ 1, & F(x) \geq \delta. \end{cases} \tag{3.6}$$

The initial estimate h_0 is the probability of getting class one in the training set. The number of leaves in each decision tree and the number of threes are the most important hyper-parameters for gradient boosting. A faster version of gradient boosting, named LightGBM [119], exclude a significant amount of the training data to remove small gradients. As shown in table B.1, there are over 20 million data points in the training data. Many of these data points will be redundant, and therefore provide little contribution while impacting the computational speed. Therefore are the LightGBM version of gradient boost used.

LightGBM is not trained on the coordinate features for the same reason as the traditional classifiers. It is therefore only trained on the colour features and the curvature features. The hyper-parameters are not optimised, but rather set to standard values of 1000 decision trees and a maximum 32 leafs per tree.

Deep learning-based classifiers

Deep learning has been shown by several other projects to have a good performance on semantic segmentation of point clouds from real wold data [32], [121], [123]–[126]. The results seem to vary in both fields of application and representation of the environment. As proposed by [14], [88], there are three main categories of semantic segmentation of point clouds using deep learning: Multi-view, 3D-CNN and Graph-based.

At the time of writing, no known implementations are using deep learning-based semantic segmentation on point clouds to grasp or dismantle an object. In addition, deep learning-based algorithms for semantic segmentation or instance segmentation of point clouds are commonly only compared by accuracy and not by processing time. This is probably a result of the field of deep learning on point cloud being quite fresh [14], [124]. Several promising algorithms, rooted in different concepts, are tested and compared with a focus on classification performance and forward processing time.

PointNet [22] and its successor, PointNet++ [87], are the fundament for most deep learning-based algorithms for working on point clouds [14], [124]. By using a computed global representation and a point-vice representation to classify each point, as shown in figure 3.17, the algorithms can learn both local and global features. Except form utilising local and global features are the workings PointNet novel in that it only uses the well-known feed-forward neural network architecture for aggregation. Since PointNet is the backbone for most deep learning-based algorithms, it becomes natural to include it in any comparison.

Graph-based neural networks (GNN) is a field within neural networks that only recently gained popularity. Nevertheless, the graph-based neural networks are viewed as a promising way of processing point clouds [14]. The fractal-based GNNs that revolves around the Laplacian matrix and eigenvectors are computational complex by nature [58]. Therefore, promising approaches like RGCNN [127] and other spectral-based GNNs are considered not applicable. GNNs have their strengths as they can for an understanding of the local geometry by utilising message passing, thus expanding the idea of PointNet that only looks at single points for local features.

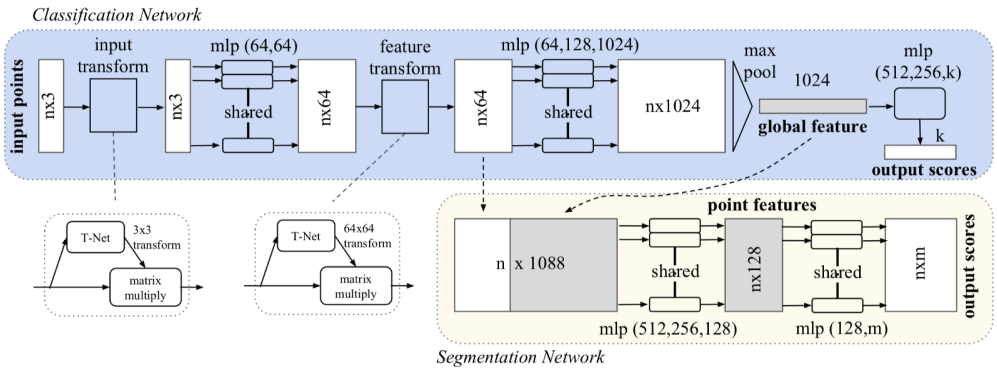


Figure 3.17: The workings of PointNet. Taken form the original paper [22].

The spatial-based GNN, Dynamic Graph CNN (DGCNN) [98], performers well on the ModelNet40 benchmark [31] with an accuracy of 92.9%. DGCNN have two core architectures that make it suitable for point cloud semantic segmentation: The message passing method "EdgeConv", and the dynamic re-drawing of the graph. EdgeConv allows for convolution-like operations similar to those in traditional 2D-CNN, while still maintaining permutation invariance, thus overcoming some of the shortcomings of PointNet. By dynamically re-drawing the graph based on the k-nearest neighbours in the feature space of hidden layers several times along the pipeline, the points with similar characteristic get closer. This allows for harder training as the information is exchanged between similar nodes, thus allowing the algorithm to focus on the nodes that are hard to distinguish. Nevertheless, as talked about in subsection 3.5.3, the formation of k-NN graphs are computationally heavy.

The architecture of DGCNN, as shown in figure 3.18, utilises PointNets idea of using a computed global representation together with local features to classify points. Although DGCNN does not use the same global representation for graph classification and node classification, they are computed similarly. DGCNN also utilises the principles of residual networks by concatenating features of the lower layers with those of the higher layers. This allows for features related to the local neighbourhood to be forwarded while the later layers pick up features for separating similar-looking points. DGCNN, like many other GNNs, are shallow compared to their 2D equivalents that can go to tens and hundreds of layers [128]. By being shallow, they do not get the performance boost that traditional deep 2D CNNs get, but on the other hand, they avoid the vanishing gradient problem that the deeper networks struggle with. Shallower network structures also allow for faster training as the gradients are smaller, thus can be computed faster.

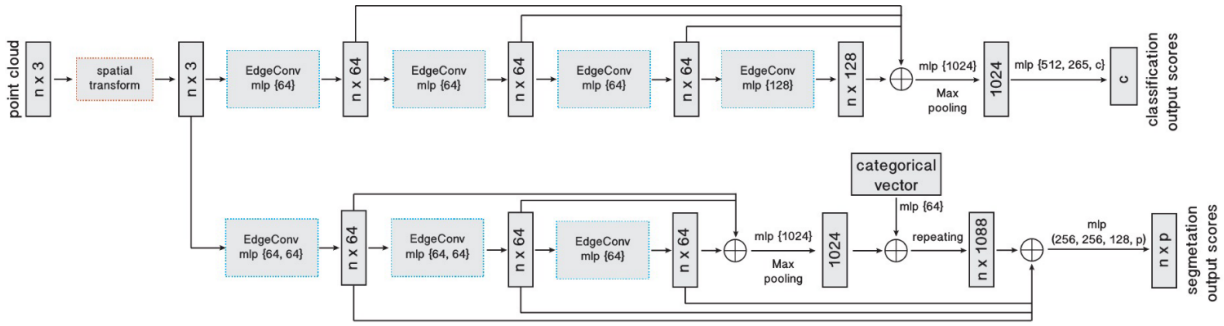


Figure 3.18: The workings of DGCNN. The upper branch is for graph classification, while the lower one is for node classification and is the one used in this thesis. Taken from the original paper [98].

3D convolutional neural networks (3D-CNN), also referred to as volumetric convolutional neural networks, have the advantage of being directly transferable from the 2D equivalent as they operate on a similar grid-based structure. By first converting the point cloud to a voxel grid occupancy map, 3D-CNN and trilinear interpolation as shown by SEGNet [120]. The model of SEGNet is not described in the paper, nor published, thus its results can not be reconstructed in this thesis. Never the less VoxelNet [118] present a model where first voxel features are learned based on the containing points in their "feature learning network", as shown in figure 3.19. This allows each voxel to learn its local features of the point clouds. Afterwards, in the "convolutional middle layer," the voxels are aggregated using 3D-CNN to learn global features.

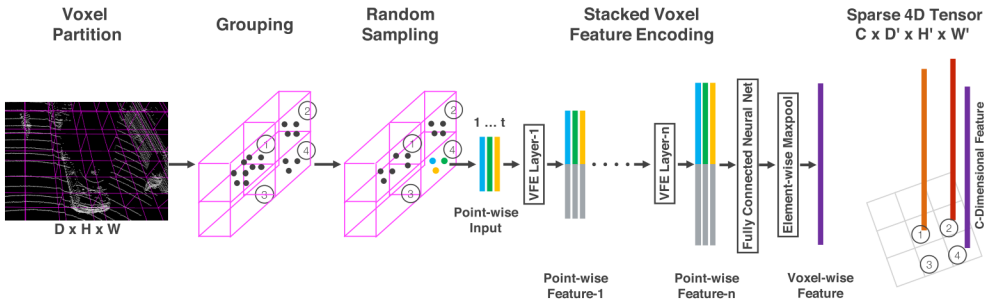


Figure 3.19: The working of the feature learning network of VoxelNet. "VFE" refers to the Voxel Feature Embedding module of VoxelNet. Taken from the original paper [118].

In the original paper, a third stage proposes regions based on the aggregated voxel features. This third layer is excluded and replaced with a trilinear interpolation layer to determine the point features as suggested by [32], [120]. The semantic value of each point is then determined by a simple MLP where the 64 point features are concatenated with the three-point coordinates that serve as the input. The structure of the MLP is shown in table 3.20. Using 3D-CNN is more computationally heavy, but is dependent on the number of voxels rather than the number of points. In other words, a good embedding from points to voxels that allows for large voxels can compensate for high time complexity.

Layer	Input features	Output features	Activation function
fc1	64+3	32	ReLU
fc2	32	32	ReLU
fc3	32	2	Softmax

Figure 3.20: MLP for VoxelNet. fc = fully conected linear layer.

Excluded algorithms

RandLA-Net [121] is based on the auto-encoder architecture, as shown in figure 3.21. The point cloud is first compressed down to a small high-dimensional latency space by the encoder part and then expanded back to the original shape by the decoder. RandLA-Net also uses attention-based mechanisms in combination with neighbour sampling to forward local features. In the Semantic3D benchmark [26], RandLA-Net has, at the moment of writing, the best score out of all algorithms with a mIoU (mean intersection over union) of 76.0%. But the algorithms are developed towards navigation of indoor environments, and not semantic segmentation of objects and is therefore not used. Nevertheless, its promising results in combination with the success of attention-based and autoencoder-based architectures

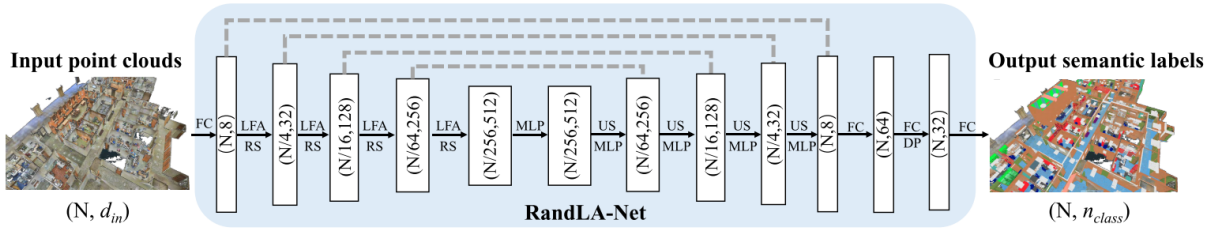


Figure 3.21: The workings of RandLA-Net. Taken from the original paper [121].

SEGCloud [120] is based in 3D-CNN on a voxelized version of the input point cloud. The processed features of the voxels are then transferred back to the point cloud by using trilinear interpolation. Although looking promising, there is no description of the architecture of the 3D-CNN, and it is therefore not feasible to reproduce the network. On the other hand, the pipeline of SEGCloud, as shown in figure 3.22, is used with VoxelNet [118] as the backbone as mentioned earlier.

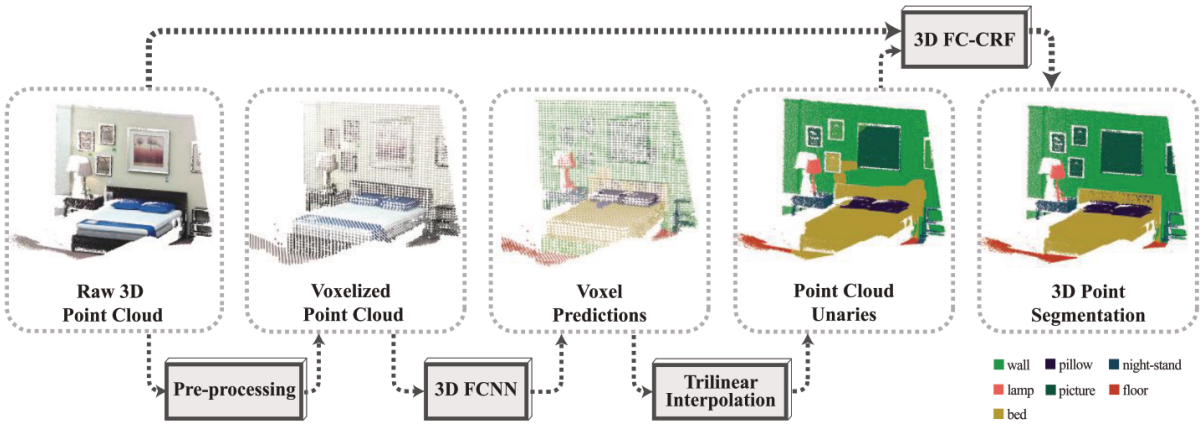


Figure 3.22: The workings of SEGCloud. Taken from the original paper [120].

Performance metric

It is seen as equally important to correctly label cables (true positive) as background (negative). False positives can lead to the formation of false clusters, while false negatives can lead to discontinuous clusters or too low density to form clusters. Accuracy (3.7) is used as a performance metric for the semantic segmentation system because it represents the classification of both positive and negative observations.

$$\text{Accuracy} \overset{\text{def}}{=} \frac{TP + TN}{TP + FP + TN + FN} \tag{3.7}$$

TP = true positive, FP = false positive, TN = true negative, FN = false negative

The disadvantage of the accuracy metric is its sensitivity to unbalanced data sets. The data sets used in this thesis, listed in table B.1, are highly imbalanced and therefore needs to be converted before training. The F-score (3.8) is an alternate to accuracy but is not used as the amount of imbalance varies in the data set and it is difficult to balance the importance of precision and recall.

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \tag{3.8}$$

Training

The imbalance in the data sets which are bias towards the background label (negative) is counteracted by creating a subset of each point cloud. The subsets consist of equally many positive and negative samples thus it is not biased towards any of the classes. All the positive samples are directly transferred to the new subset, while the negative samples are randomly sampled to conserve the natural variance in the background point features. This variance is important to conserve because background points represent several types of objects, and the system must be able to distinguish all of them. The variance in background features can also be seen in the t-SNE embeddings of the point clouds feature space as shown in figure 3.16. The inputs are all normalised in most of the training to prevent the algorithms from being bias towards a single attribute. The only exception is the graph-based algorithm (DGCNN) where the original coordinates are preserved to not interfering with the forming of the graphs. By normalising, the axis would be scaled with different factors, thus influencing the calculation of distances used to form the graphs.

The non-deep learning-based algorithms are not directly trained as they do not have any form of backpropagation, they are instead fitted to the training data. First, the training data is concatenated into one large point cloud, and then the algorithms are fitted to the new point cloud.

For the deep-learning-based algorithms, the CUDA framework is used to speed up training time. This limits the memory available to that of the GPU, hence 8 GB C.1. To overcome this limitation, the point clouds are divided into 8 smaller chunks that are individually processed. This has the additional benefit of reducing forward time with a factor of up to 64 as some of the algorithms have time complexities of $O(n^2)$. The large input data do not allow for training in mini-batches which would otherwise help to compute better gradients [57]. The learning rate is scheduled with cosine annealing and momentum, as done by [98], to avoid local minimums. The hyperparameters used during the standard training are shown in table 3.2.

	DGCNN	PointNet	VoxelNet
Training/Validation split	80%/20%	80%/20%	80%/20%
Epochs	200	200	160
Learning rate	0.1	0.01	0.01
Scheduler type	Cosine annealing	Cosine annealing	Step ($\gamma = 0.1$)
Scheduler period	50	50	150
Momentum	0.9	N/a	-
Batch size	One chunk	One chunk	One chunk
Dropout probability	50%	50%	50%
Optimiser	SGD	Adam	SGD

Table 3.2: Base value for hyper-parameters used during training.

Commonly the performance of point cloud semantic segmentation models are trained and compared with region-based metrics, where the mean IoU (3.9) (intersection over union) is the most common metric [124]. In the popular benchmarks [26], [28], [31] the objects are mostly separated in space. In other words, they are not entangled and rarely have overlapping boundary boxes. EV battery cables on the other hand are long and slim while being entangled. Boundary boxes and region-based loss would therefore not be feasible for this application. The boundary boxes would contain ether be too large thus including points that are not a part of the cables, or each cable would have to be divided into several smaller boundary boxes. The same arguments make boundary-based loss

also impractical. In addition, when training on chunks of the point cloud, a boundary box may overlap into several chunks thus making it more difficult to determine the loss.

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{3.9}$$

The loss function is therefore distribution-based, hence every point is individually evaluated and compared to the ground truth. The standard cross-entropy loss, described in section 2.6.1, is chosen as the primary loss function, as this is the standard for binary classification tasks. The weighted cross-entropy loss is not used due to the varying frequency of cable labels between the point clouds. As seen in table B.1, some point clouds have around 15% of the points labelled cable, while others have as low as 1.9%. With a weighted cross-entropy loss, the ratio between labelled and unlabeled points would be used as weights to not be biassed towards any class. This would lead to the positive samples from some point clouds would be weighted significantly stronger, thus creating a bias. The focal loss, shown in equation (3.10), is also a viable option as it is known for dealing good with heavily imbalanced data [66], [67], [71]. On the other hand, focal loss introduces the hyperparameter γ, thus complicating the learning process. Focal loss is therefore seen as an alternative loss function to potentially be used later to increase performance or to be used when standard binary cross-entropy loss is not sufficient.

$$\mathcal{L}_{FL}(y, \hat{y}) = |y - \hat{y}|^{\gamma} \cdot \mathcal{L}_{BCE}(y, \hat{y}) \tag{3.10}$$

3.5.6 Clustering

As the number of cables is unknown the clustering algorithm can not be reliant on having a fixed number of clusters. Also, the clustering algorithm must be flexible towards the number of clusters, thus being able to work with different magnitudes of numbers of clusters. As pointed out several times in this thesis: cables are entangled and twisted. They do not take up a separate convex chunk of space, and therefore the clustering algorithm can not operate in a regression-based manner as partition-based clustering does. Hierarchy-based is not applicable as it also depends on a fixed number of clusters and convex data.

As stated previously in section 3.5.2, the clustering algorithm also serves as noise removal from the imperfect semantic segmentation algorithm. Therefore, the clustering algorithm should not be sensitive to noise or outliers. The algorithm should ignore outliers, thus not considering them as a part of any cluster. This again makes the partition-based clustering algorithms unsuitable. On the other hand hierarchy-based, density-based and fractal theory-based algorithms are more robust to noise because they work in a recursive manner where points are added or removed according to some given constrain.

Although the point clouds from the Zivid camera are considered ordered, this should not be a constrain for the workings of the perception system as this would violate the permutation invariance criteria set in section 3.5.1. Therefore, all point clouds are treated as unordered point clouds. Sensitivity to the ordering of the sequence of the points, therefore, becomes important. Fractal clustering (fractal theory-based) fall short as it relies upon looping over the input data only once. Optionally, the point cloud could be ordered before any clustering, but this process of ordering would add at best add a time complexity of $O(N \log N)$ [129], thus it is not viewed as a viable option due to the time constraints set in section 3.5.1.

The importance of having a clustering algorithm that manages to work on non-convex data is illustrated in figure 3.23 as algorithms like K-means (partition-based), and BIRCH

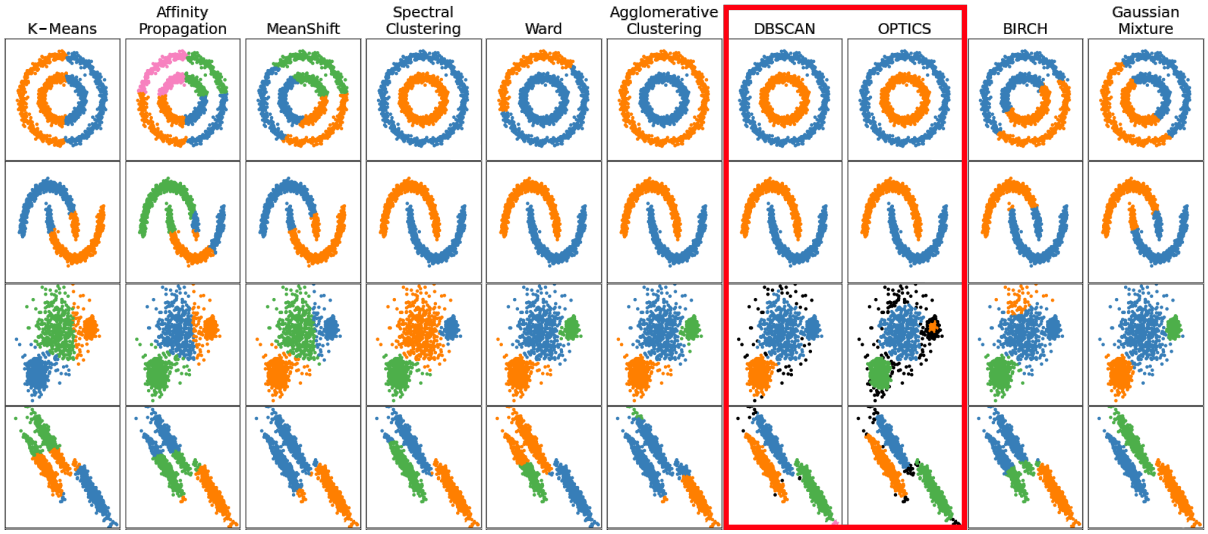


Figure 3.23: A comparison of clustering algorithms on dummy data to illustrate how different algorithms work on different shaped data [130].

and Agglomerative clustering (hierarchy-based) wrongly classifies the non-convex data shown in the first two rows. On the other hand, the density-based clustering algorithms DBSCAN and OPTICS manages the noon-convex data better.

Although the density-based mean-shift algorithm is used in similar projects [115], it is not used in this thesis due to its high time complexity [103]. The viable clustering algorithms are therefore DBSCAN and its predecessor OPTICS, both highlighted in figure 3.23. They both work by forming cluster centres from the dens part of the data and then recursively evaluating neighbouring points to determine if they should be included or not in the current cluster as illustrated in figure 3.24. In the original paper of OPTICS [108], the author claims an increase in time usage by a factor of 1.6. This factor must be determined empirically for this application as it depends on the search radius of OPTICS, ϵ_{max}.

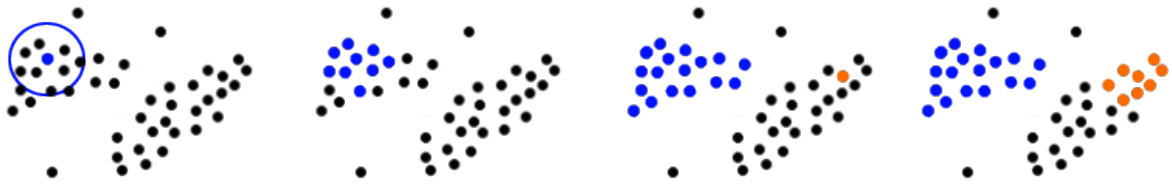


Figure 3.24: The iterative workflow of DBSCAN and OPTICS.

The hyper-parameters for the two methods, in addition to the choice of metric, are the neighbourhood distance ϵ and the minimum neighbour samples k. ϵ indicates how far two points can be apart, while still being considered neighbours. In the leftmost frame of figure 3.24, the size of ϵ is indicated with a blue circle around the centre point. ϵ takes the shape of a circle since the euclidean metric is used. The second hyper-parameter, k, indicates how large a neighbourhood is needed for a point to be considered a centre point, thus adding its neighbourhood to the current cluster. In OPTICS, only a max value for the parameter ϵ is required, as the optimal value is dynamically estimated. Nevertheless, ϵ_{max} can be set to infinite, thus including the entire point cloud, but this leads to quadratic time complexity ($O(n^2)$). OPTICS uses an additional hyperparameter, ξ, in its clustering process. How OPTICS calculates the average distance every point have to its neighbourhood and looks at how this distance changes to determine the clustering borders. ξ determines the threshold value for an increase in the average distance between

two neighbours.

The size of ϵ must be larger than the voxel side length l_{voxel} used in the global filtering process, described in section 3.4.6, to have any neighbours at all. Hence, the lower bound of ϵ is as described in equation 3.11.

$$\epsilon_{min} = l_{voxel} \tag{3.11}$$

The point cloud is a surface representation of the batteries, hence they do not have any volume. Therefore, when viewed locally, the points will lie on the surface. As a result of the voxel filter described in section 3.4.6, the points will be spaces apart with no less than the voxel size given by the inaccuracy (l_{voxel}). The upper bound of k can therefore be given by the proportion between the surface created by ϵ (the neighbourhood) and the point density as shown in equation 3.12.

$$k_{max} = \frac{\pi\epsilon^2}{l_{voxel}^2} \tag{3.12}$$

For the OPTICS algorithm, the hyperparameter ϵ_{max} is set to be $10 \cdot l_{voxel}$ as a conservative value to reduce the processing time.
The hyper-parameters for the two methods are empirically determined in chapter 4. Their performance is also compared to determine which one is the most ideal.

Clustering metric

The dissimilarity metric should preserve the rotational invariance and translational invariance of the perception system. Therefore, the city-block metric distance is influenced by the orientation of the reference frame, hence violating the rotational invariance. In the worst case, a 45^o rotation can change the distance between two points with a factor of $\sqrt{2}$. For the same reason, the cosine metric does not hold as it violators the translation invariance by changing the angle as shown in figure 3.25.

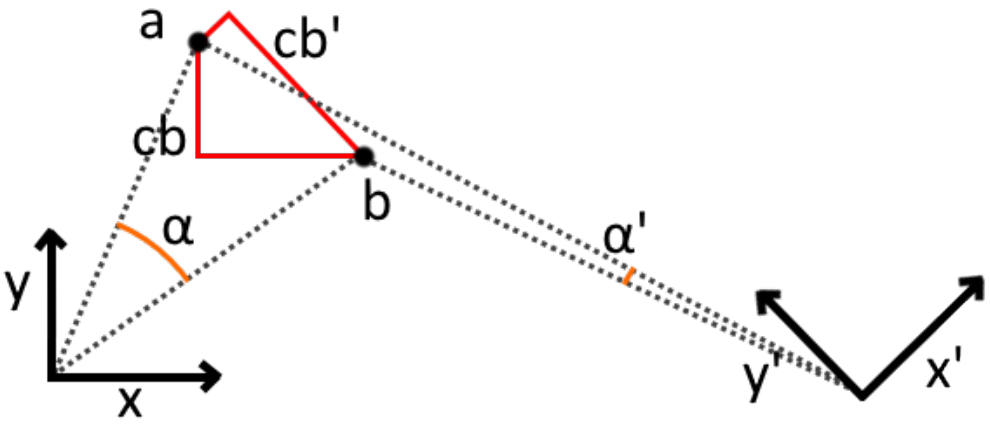


Figure 3.25: An illustration of how the cosine metric and the city-block metric violates the rotation and translation invariances. The cosine distance (orange) and the city-block distance (red) is shown for the original reference frame with axis X and Y, and the shifted and rotated reference frame with axis X' and Y'. cb = city-block, α = the angle related to the cosine distance.

The Euclidean distance is not dependent on the reference frame, thus being invariant of any translation or rotation. It is in addition relatively fast to compute [103] and is intuitive as it is the day-to-day metric for separating objects in 3D-space.

3.6 Software implementation

The implementation is done by using the Python and ROS frameworks because this project is a continuation of previews work, and therefore continuity is important. The project could also have been done in C++, but Python is favoured as it has good support for machine learning.

The code discussed in this section can be obtained thru the following link: https://github.com/HenrikBradland-Nor/IKT590-MAS500.

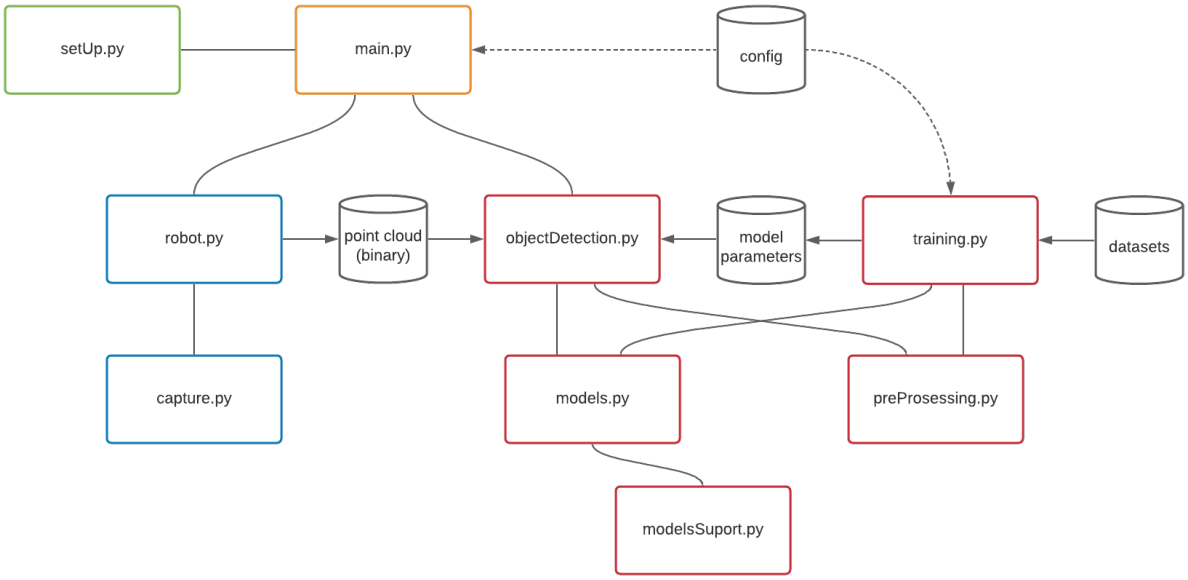


Figure 3.26: An overview of the software files and there relations. Yellow: the main executable file. Green: hardware setup file. Blue: the capturing system. Red: the perception system.

The project code is mainly divided into the following files:

- `main.py` is the executed python file and only contains the main program flow.
- `config` contains the class `config` that again hold every hyper-parameter and global variable in the entire code. The `config` class is instantiated in the main function and passed down to the classes and functions that rely on it, thus there exists only one instance of the config object.
- `robot.py` is called from `main.py`. It contains the interface with ROS, therefore does the operation of the robot and 3D camera. The main purpose of the file is to perform the capturing of the battery and filter the capered point clouds.
- `models.py` contains the implementation of all the semantic segmentation algorithms described in section 3.5.5 and clustering algorithms described in section 3.5.6.
- `training.py` loads the training data set and the chosen model to perform training. The model parameters are then stored in a separate file.
- `objectDetection.py` loads the chosen model from `model.py` with corresponding parameters stored in a separate file. Any passed input is filtered, downsampled, labelled and clustered in accordance with the system workflow described in section 3.5.2.

3.6.1 External libraries

The following external libraries are used to implement the software.

- **numpy** (v. 1.19.1) is used for efficiently managing matrices and vectors on the CPU. Mostly used for data transformation and data augmentation, and as an interface with other libraries.
- **pytorch** (v. 1.7.1)is used for neural networks processing as it makes interacting with the CUDA toolkit convenient, thus speeds up forward processing time and training time. In addition, PyTorch has good support for machine learning-based functionalities and is together with TensorFlow one of the most commonly used frameworks for machine learning. The PyTorch framework is chosen over TensorFlow because of the compatibility with DGL.
- **DGL** (v. 0.5.3) is short for Deep Graph Library which builds on the PyTorch framework to provide support for graph-based machine learning.
- **pclpy** (v. 0.11.0) is a binding of the c++ library PCL (Point Cloud Library) which provide an efficient framework for point cloud processing.
- **rospy** (v. 1.12.14) is the ROS interface for python and is used to communicate with the Zivid camera and the robot controller as shown in figure 3.2.
- **scikit-learn** (v. 0.23.2) provides implementations for non-deep learning-based algorithms like t-SNE, random forest and various clustering algorithms.
- **joblib** (v. 0.17.0) help save and load more complex data structures from files. It is used to save and load the parameters of the non-deep learning-based models.
- **LightGBM** (v. 3.2.1) is a framework for implementing and training boosting algorithms.

Camera positioning

The MoveIt move order system uses a set of position values (XYZ) and a set of quaternions to specify the pose of the end effector. A script designed to make the input values more intuitive was made. The script called `Batpiv` was therefore designed to generate the required goal position for the camera to be pointing towards the battery pack from a certain angle and distance, the `Batpiv` script is explained further in Chapter 3.4.2.

The required system inputs are the following:

- **Battery position:** The point at which the camera will pivot around, given as a set of XYZ values. Placed where the feature of interest is located.
- **Pivot arm length:** The radius from the battery position to the tool position. Gives the option to pivot around the battery at a further distance. Keep in mind the tool position in this thesis is placed 1.1 meters from the camera, where the centre of focus of the camera is located. Giving an actual radius of 1.1 meters when the pivot arm length is set to zero.
- **z-axis angle:** The angle between the Z-axis and the axis of the camera view. If the ZY-plane angle is zero, the camera will pivot in the negative X direction. and positive direction when negative.
- **ZY-plane angle:** The angle between the ZY-plane and the axis of the camera view. If a positive angle is given, the robot will twist the camera in a clockwise direction around the Z-axis, and counterclockwise if negative.

A set of pose goals can be defined in the `config.py` script by calling the `Batpiv` function and ordering the poses in an array. To use the set the set would need to be specified in `Robot.py` as well. This is done so `Config.py` can work as storage of several capturing sequences poses, and the desired sequence can then be specified in `Robot.py`.

`Robot.py` handles the process of sending move orders to the robot. The actual movement sequence is done by the `move_group` system in MoveIt. The `Robot.py` configures the settings for the move order and requests a move order to the given goal pose. The `move_group` then handles the path planning, collision avoidance, and robot control orders. When the robot has reached the desired location, the point cloud capture service is called.
Figure 3.27 is an illustration of how the system goes from input to calling the move robot action.

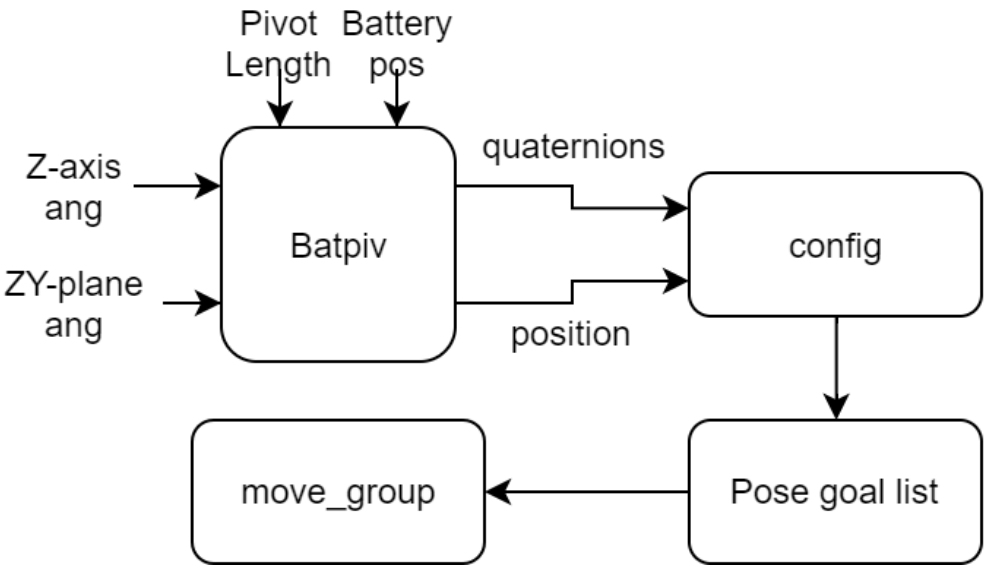


Figure 3.27: Input variables to robot pose execution.

Point cloud processing

When the Zivid capture service is called, the Zivid camera starts the process of capturing and generating a point cloud. This process also involves the filtering from the built-in filters described in Chapter 3.4.1. To process the generated point clouds two ROS topic channels are used. The captured point cloud is published to the ROS topic `/zivid_camera/points` and a listener to `/tf` is started to keep track of the point of view used when the point cloud was captured. The process of post-capture to exported point cloud is depicted in figure 3.28.

The process of transforming and exporting the point clouds consists of the following steps: The data from `/zivid_camera/points` is converted from the message type `Pointcloud2` into a list. The list is then processed through the pre-transformation filters, removing `NaN`-points and removing points which is outside of the desired range of the camera, in the case of this thesis, this filter removed all points which were beyond 1.6 meter from the camera, and nearer than 0.6 meters, eliminating points which was outside of the desired range of the camera.

The `/tf` topic is where all 3D-transform data of the ROS system is published. To get the associated transform of the camera frame (`zivid_optical_frame`) to the origin frame (`world`) the transform between the two frames is looked up. The results from the lookup are then converted to 4×4 matrix. The `XYZ` values in the list are then transformed point

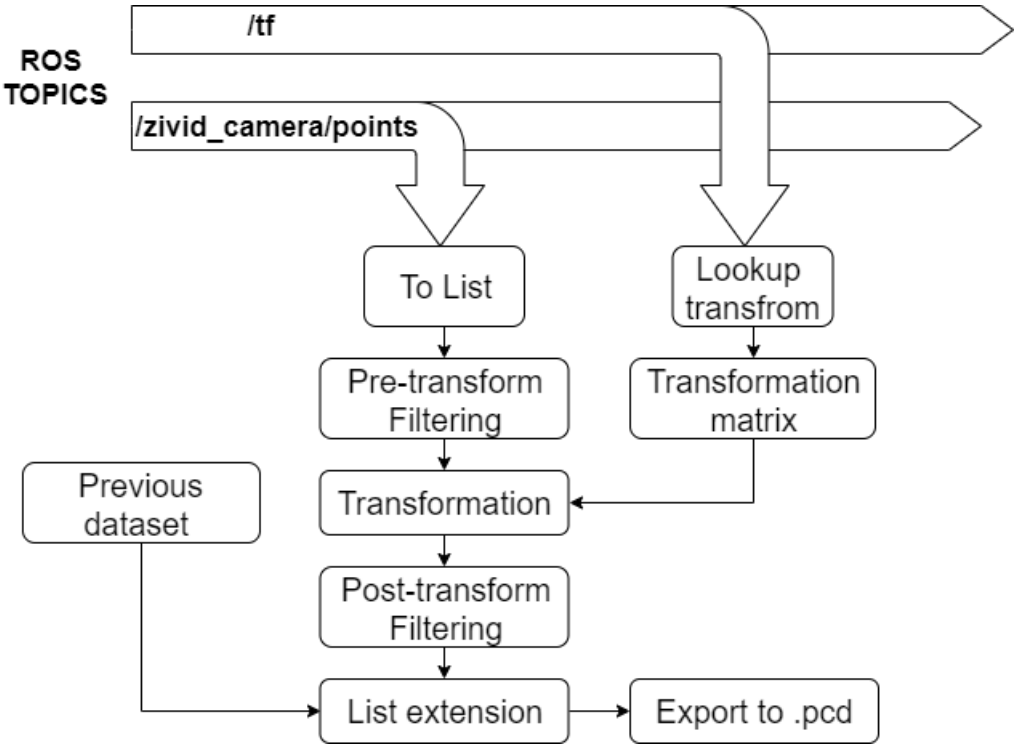


Figure 3.28: Diagram of how point cloud data is processed and exported.

by point to transform the point cloud to the `world` frame.

Post-transformation filters are then applied. The post-transform filters remove all points outside of a valid range, from the point of view of the world frame, like removing everything which is located below around the battery pack of interest.

Since the system is designed to capture from different angles, the point cloud data is then added to a larger list consisting of all point clouds that have been captured since the last time `main.py` was run. The whole point cloud process will then repeat until all requested capturing are completed. When all capture requests have been processed the combined point cloud is exported to a `.pcd` file.

3.6.2 Semantic segmentation

The algorithms of choice, as described in section 3.5.2, are implemented in Python 3.6 as this is the only distribution of Python that supports all libraries. The PyTorch framework is used for the deep learning-based algorithms because it allows for easy integration with the Nvidia CUDA Toolkit, thus allowing for GPU-based processing that heavily reduces processing time for deep neural networks.

All the code related to semantic segmentation are located in two files: `models.py` and `NN_suport.py`. In addition, there is two prepossessing files and `dataPreProsessor.py`.

Random forest

The model implementation itself is done by using the scikit-learn libraries `ensemble.RandomForestClassifier()` function. The data is first prepossessed with by `dataPreProsessor.py`. To make the implementation translation invariant, the training is done on the absolute position, but rather the local curvature represented by a normal vector. The normal vectors are constructed using the function `pclpy.compute_normals()`.

SVM

The model implementation itself is done by using the scikit-learn libraries `sklearn.svm.SVC()` function. The data is first prepossessed with by `dataPreProsessor.py`. To make the implementation translation invariant, the training is done on the absolute position, but rather the local curvature represented by a normal vector. The normal vectors are constructed using the function `pclpy.compute_normals()`. Including the z-coordinate as an attribute and normalising could improve predictions, but this would make the model sensitive to outliers as any outlier would affect the normalisation process.

The reference data set is reduced to speed up process time. The reduction is implemented by collecting random samples from all reference point clouds. The samples are taken uniformly from the two classes to avoid any bias.

LightGBM

The implementation of LightGBM is done by using the framework with the same name. The model is trained using the `lgb.train()` method, and predictions are made using the `lgb.predict()` method. The LightGBM framework has most of the functionality already set up, thus the implementation of the LigthGBM algorithm is straightforward.

Dynamic graph CNN

The segmentation versions (the lower branch shown in figure 3.18) of DGCNN is implemented using the DGL framework. DGLs class `dgl.nn.pytorch.conv.EdgeConv()` is used as it is a direct implementation of the EdgeConv block for message passing described in the original paper [98]. The `dgl.nn.pytorch.factory.KNNGraph()` class is used for forming the graphs before each EdgeConv block as shown in figure 3.18.

VoxelNet

The "Feature Learning Network" and "Convolutional Middle Layers" are implemented using the PyTorch framework as described in the original paper [118]. A trilinear interpolation layer is implemented to assign point attributes based on the convoluted voxels. The interpolation is constructed using the PyTorch framework to be done on a CPU and to allow for efficient backpropagation. The 64 point features in addition to the three-point coordinates are then aggregated using a small MLP (described in table 3.20) which is implemented using the `torch.nn` module.

PointNet

The implementation is done using the PyTorch framework as the backbone with the `torch.nn` module providing the required functionality. The implementation is done following the original paper [131], with the only adjustment being changing the model input from n to $n \times 6$ to include the colour attribute of the points.

3.6.3 Clustering

The two clustering algorithms DBSCAN and OPTICS are implemented using the scikit-learn libraries `sklearn.cluster.DBSCAN` and `sklearn.cluster.OPTICS` modules. The output of the clustering algorithm, which is a vector of $n \times 1$ that indicates the clustered index $(0, 1, \cdots c)$ of each point, is concatenated with the point coordinates $(n \times 3)$ to form a point cloud with four attributes: "x", "y", "z" and "cluster index". The points with cluster index -1 is discarded as they are seen as outliers and does not belong to any cluster. The remaining points are then grouped into separate point clouds $P_0, P_1 \cdots P_c$ based on their cluster index. The workings of the clustering pipeline are shown in figure 3.29.

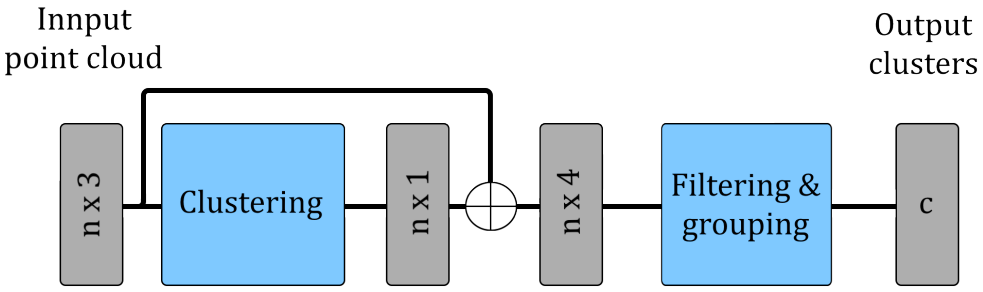


Figure 3.29: The clustering pipeline. $\oplus$ indicates concatenation.

Chapter 4

Results

In this chapter, the results from the capturing process is presented. Following with the performance of classification algorithms, with the clustering algorithms being presented last.

4.1 Capture quality

The results in reference to the quality of point cloud capturing are done on a basis of visual inspection. Therefore, the results may subject to a bias in the form of personal preference. The main feature used to define a point cloud as being of good quality in this thesis is how well the cables were represented. This is due to the cables being the main focus of this thesis.

Dynamic range testing

To test the capturing quality, a region of the smaller battery pack (Battery A) was chosen for comparison. The region depicted in figures 4.1 to 4.4 was chosen since the region contains a distribution of key components which need spatial consideration to capture correctly. Namely the orange and black cables, the white stickers and markings, the metallic plating's, and the black plastic. The subfigures on the left (Sub figure a) depicts the point cloud generated with the setting in the corresponding table. The right subfigure (subfigure b) is the depth map from the point cloud, included to make it easier to identify the regions where the system failed to generate a point.

The test captures also used the built in filters of the Zivid camera. Table 4.1 lists the filters and the filter settings.

Outlier	Gaussian	Reflection	Saturation	Contrast
Threshold = 5.00	Sigma = 1.5	True	True	True

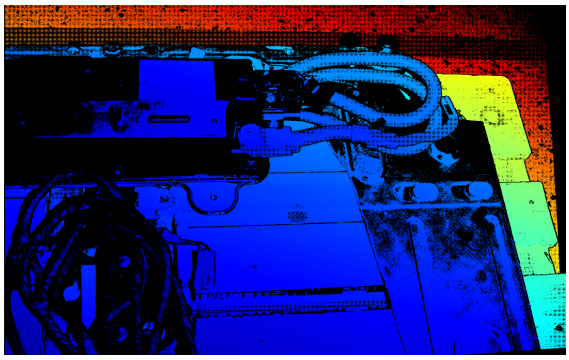
Table 4.1: Built inn Zivid filters used in capture testing.

Upper range capture settings The settings listed in Table 4.2 are found by reducing the stops to a point where no clipping is occurring at the most exposed regions. As seen in figure 4.1a the resulting point cloud is quite dark, but none of the whiter regions, like the sticker on the black plastic box, is removed by the saturation filter. The black plastic and wires are detected poorly by the system, showing up as black regions in the depth map of figure 4.1b. The orange cables are however quite well defined with these setting.

Exposure time	Iris	Brightness	Gain
10 000	27	1.00	1.00

Table 4.2: Capture settings for the upper-exposures.

(a) Point cloud from upper-exposure



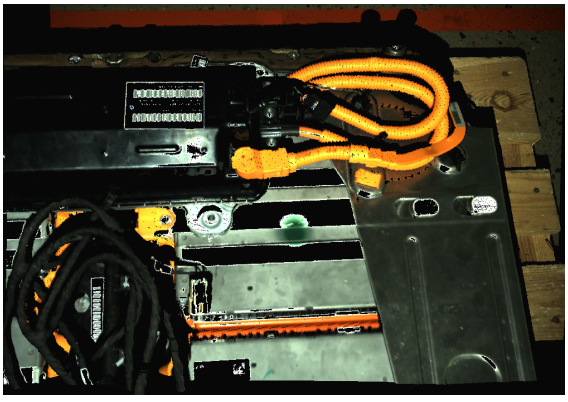
(b) Depth from upper-exposure

Figure 4.1: The point cloud and depth map captured with the settings of Table 4.2.

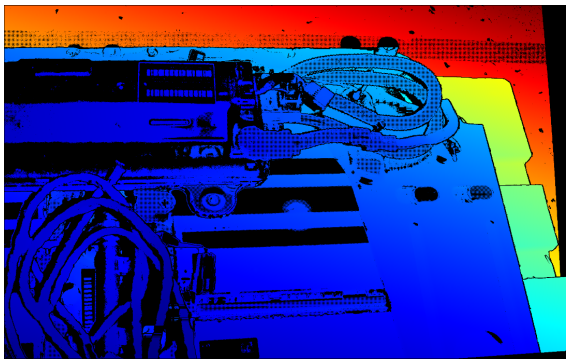
Mid range capture settings The settings listed in Table 4.3 are found by increasing the stops from the upper range capture settings until the mid-tones is well exposed. As seen from figure 4.2a, the settings give a point cloud where only the brightest parts, like the sticker on the black box, is saturated. The black box and cables are more successful point generations. Tho the black cables are improved, the black box is large chunks missing. The settings seemed to work well for both the orange cables and for the metallic plating.

Exposure time	Iris	Brightness	Gain
10 000	46	1.00	1.00

Table 4.3: Capture settings for the mid-exposures.



(a) Point cloud from mid-exposure.



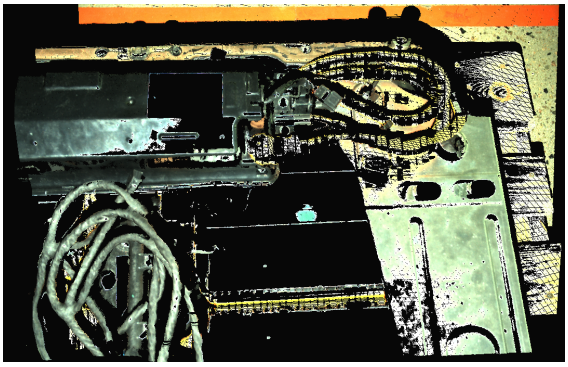
(b) Depth from mid-exposure.

Figure 4.2: The point cloud and depth map captured with the settings of Table 4.3.

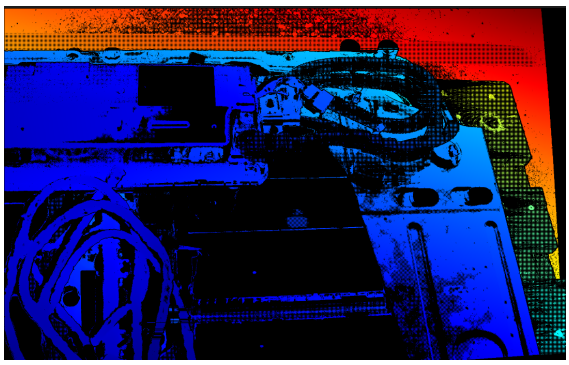
Lower range capture settings The settings listed in Table 4.4 were found by increasing the stops until the least exposed regions were detectable. As seen in figure 4.3, the modules behind the metal plating are all saturated. The orange cable is almost gone, only the edges remain. However, the black cables and box are in large part visible.

Exposure time	Iris	Brightness	Gain
20 000	46	1.00	3.00

Table 4.4: Capture settings for the lower-exposure.



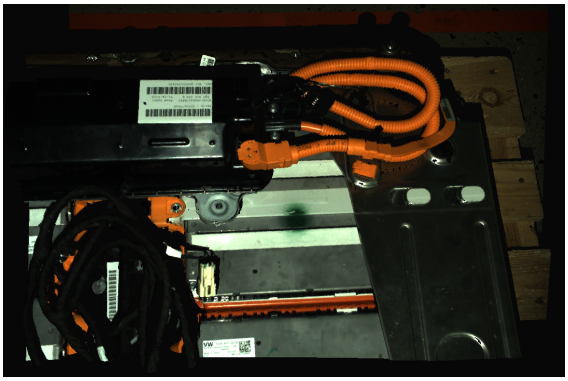
(a) Point cloud from lower-exposure.



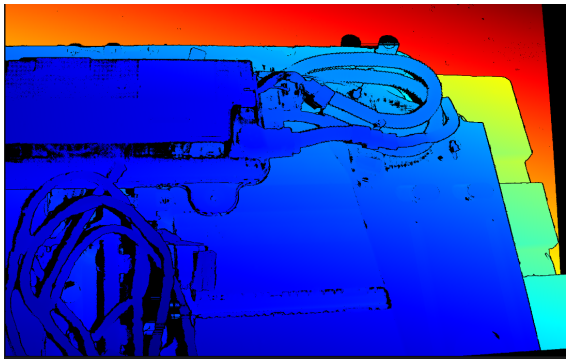
(b) Depth from lower-exposure.

Figure 4.3: The point cloud and depth map captured with the settings of Table 4.4.

HDR Figure 4.4 represents the combining of the three capture settings into an HDR capture mode, in essence combining the three captures into one capture. The results give a capture that covers the whole range of exposure in the test region. As can be seen in the depth map of figure 4.4b, there are next to no regions missing.



(a) Point cloud from HDR-exposure.



(b) Depth from HDR-exposure.

Figure 4.4: The point cloud quality from HDR capture.

Deviation from Zivid Studio

The three tested capture settings showed in this section are captured using the Zivid studio app. The app gives the option to customise capture settings and HDR functionality. The settings used is the same as what is used in the point cloud capturing script made for this thesis. But due to some unresolved problem in the capture code, the result does not match what is seen in Zivid studio. The captured script seems to set the saturation limit for the point cloud considerably lower, causing features visible in the studio to be saturated and removed in the point cloud export. The same effect would also be showing up whenever the HDR functionality is used.

A work around to this problem is tested by reducing the stops an additional level from the upper range capture setting, by reducing the iris one stop down, giving the following settings.

Exposure time	Iris	Brightness	Gain
10 000	21	1.00	1.00

Table 4.5: Capture settings with the required corrections.

Chosen settings

Table 4.6 shows the settings that are ultimately settled on as both desired settings and work around settings. The upper range, mid-range, and lower range is the desired setting, with the intention of combining into an HDR capture. Whilst the workaround settings are the ones that are ultimately decided upon as the workaround settings.

	Exposure time	Iris	Brightness	Gain
Upper range	10 000	27	1.00	1.00
Mid range	10 000	46	1.00	1.00
Lower range	20 000	46	1.00	3.00
Work around	10 000	21	1.00	1.00

Table 4.6: Potential and actual capture settings.

Point-of-views

To estimate the number of POV's necessary to represent the cables in a representative way is done by testing three sets of point cloud captures sequences. Consisting of sets of two, three and four POV sequences, with all captures done on a $30\,\mathrm{deg}$ tilt off the Z-axis. The test focus on a section of cables on the small battery pack (Battery A), as depicted in figure 4.5. The cable representation is judged on the basis of a visual inspection, looking at how much of the cable surface is captured.

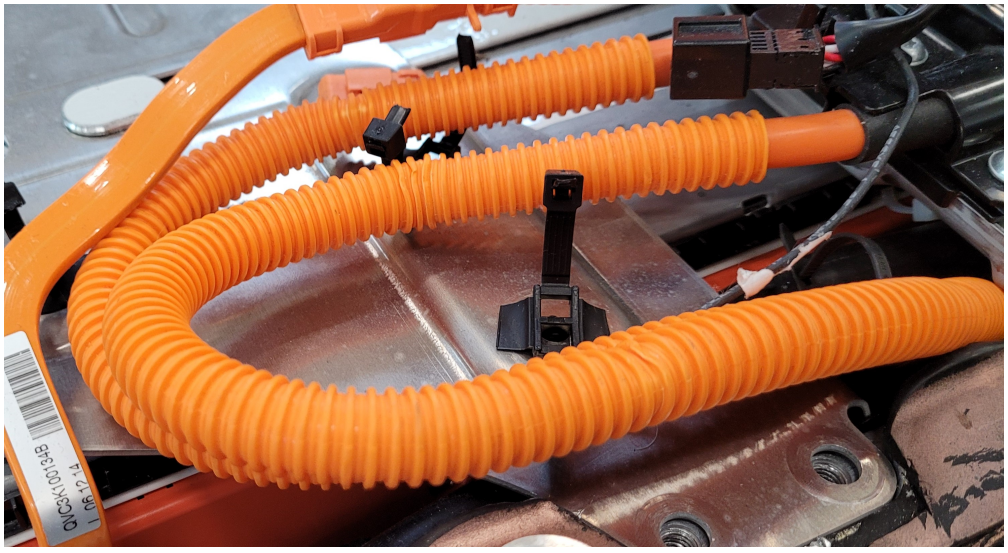


Figure 4.5: Reference image for point-of-view testing.

The test is judged based on how well the whole cable bundle is depicted. However, the quality of the full point cloud is difficult to illustrate in a productive way in a thesis format. Therefore a region that is deemed representative of the overall quality is singled out for illustration. The cable located in the forefront of figure 4.5 is chosen, due to it having a full $180\,\mathrm{deg}$ turn, with minimal obstructions covering its surface. The resulting point clouds are depicted in figure 4.6.

(a) Two POV results. (b) Three POV results. (c) Four POV results.

Figure 4.6: Point cloud results for POV testing.

Two POV results The two POV capturing sequence captured two point clouds each at an angle of $\pm 30\,\mathrm{deg}$ off the Z-axis, moving the camera along the X-axis. As can be seen in figure 4.6a, the resulting point cloud has good results in the regions closer to the ends, however, as the cable bend around $180\,\mathrm{deg}$ the cable gradually becomes more half-cylinder shaped, which can be problematic for a grasping software. The reason being that the camera is never in a position to see the underside of the cable.

Three POV results The three POV capturing sequence captures point clouds each at an angle of $\pm 30\,\mathrm{deg}$ off the Z-axis. However in this case the POV's are spread out by $66.6\,\mathrm{deg}$, with one POV along the negative X-axis. As seen in figure 4.6b, the surface coverage around the bend is considerably better than in figure 4.6a.

Four POV results The four POV capturing sequence captures point clouds each at an angle of $\pm 30\,\mathrm{deg}$ off the Z-axis. The four POV's are placed along the X and Y-axis, in the same fashion as the two POV configuration. As seen in figure 4.6c, the results are much the same as with three POV's. This indicates that using 3 POV's will not result in a better result, but more likely it means that adding an extra POV after three, will not give as much of an improvement, as going from two to three.

Full battery coverage

A set of estimates calculated using equation 3.1 can be seen in table 4.7. A range of camera distances and their associated field of view are tested against both battery pack areas. The ratio of how much of the field of view that can be used while still be consistent are varied between 1 and 0.6. However, from real-world testing experience, 4 pivots are needed to cover Battery A. To get the same results from the equation would require a coverage ratio (η_{cap}) of 0.6

Battery dimensions in [mm]	Surface coverage [mm] @ camera distance [m]	Coverage ratio η_{cap}	POV's N_{POV}	Captures N_{cap}
930×600	433×271 @ $0.6\,m$	1	3	15
930×600	433×271 @ $0.6\,m$	0.6	3	24
930×600	691×432 @ $1\,m$	1	3	6
930×600	691×432 @ $1\,m$	0.8	3	9
930×600	691×432 @ $1\,m$	0.6	3	12
930×600	1015×652 @ $1.6\,m$	1	3	3
930×600	1015×652 @ $1.6\,m$	0.6	3	6
2320×1400	691×432 @ $1\,m$	1	3	33
2320×1400	691×432 @ $1\,m$	0.6	3	57
2320×1400	1015×652 @ $1.6\,m$	0.6	3	27

Table 4.7: Capture requirements to cover battery pack by scenario.

The camera distance of 1 meter is of most interest for the estimation, since 1 meter is at the centre of the optimal working range of the camera. With a coverage ratio of 0.6, required captures reach 12 to reliably cover the small battery pack. Since the larger battery pack is a considerably larger area it would take a total of 57 captures to cover it in the same way. Which would likely be an impractical amount for an industrial setting. To get within a somewhat practical amount of captures, extending the camera distance to 1.6 meters could be an option. This would reduce the required captures to 27.

Time usage estimate

The system used to capture point clouds is not optimized for time usage. Therefore the actual time used to capture is not realistic if the test were to be developed to a complete capturing system. Mainly, the inefficiencies are due to how the point clouds are processed after capture, the capturing process is forced to wait for the processing to complete before it can continue. However, in an industrial version, the processing can be done in parallel with the capturing. When estimating the time for completing the capturing process, the processing is therefore not included.

The time to position the robot for one set of three POV's is timed at 27 seconds. This involves path generation, path execution and a pause to dampen vibrations. The robot is started at approximately the same distance from the first goal pose as the distance to the second goal pose, representing the time of transit between every POV sett. The capture time for the Zivid camera is assumed to be 1 second, though the exposure time in the test system is only 0.1 seconds the estimate leaves room for more extensive captures, with HRD functionality. The resulting time to capture the set is, therefore, estimated to be approximately 30 seconds.

The time required to complete the various capture sequences is listed in Table 4.8. The coverage ratio η_{cap} is assumed to be 0.6.

	POV's N_{cap}	Completion time t_{POV}
Battery A @ $1\,m$	12	120 s
Battery A @ $0.6\,m$	24	120 s
Battery B @ $1\,m$	57	570 s
Battery B @ $1.6\,m$	27	270 s

Table 4.8: Estimate of time to complete a surface covering capture.

4.2 Performance of classification algorithms

As described in section 3.5.1, the perception system needs to be both fast and accurate. The selected algorithms are compared on a custom benchmark consisting of only EV batteries. This results in a fair and realistic comparison as neither of the algorithms are tailored to perform on the custom benchmark, and any results on the benchmark are directly relatable to real-world performance. The detail of the data used in the benchmark is listed as "validation" in table B.1.

Set up

All the algorithms are trained/fitted in the way as described in section 3.5.5 on the training data-set consisting of 30 labelled point clouds. Their accuracy and forward time is then measured when performing on the validation data-set. The accuracy is computed

with equation 3.7 as described in section 3.5.5. The forward time does not include any prepossessing of the data, only the semantic segmentation.

For the two algorithms Random Forest (RF) and SVM, several versions are implemented and tested because there is no reference on what hyper-parameters to use for semantic segmentation.

Result

The accuracy of each if the algorithm is shown in table 4.9, while the forward time is shown in table 4.10.

Algorithm	point cloud								Mean
	3	4	13	14	40	42c	44c	45c	
RF 40	53.6	52.6	51.2	54.2	51.9	49.4	50.5	49.9	51.7
RF 100	54.4	52.7	52.4	52.9	52.7	48.7	50.1	50.1	51.8
RF 400	53.8	52.7	52.0	53.2	53.0	49.4	49.9	50.3	51.8
RF 1000	52.6	53.0	52.1	54.2	52.8	49.4	49.8	50.3	51.8
SVM Linear	57.7	54.3	52.7	52.3	50.8	49.4	50.0	50.4	52.2
SVM Polynomial	57.6	54.9	53.2	53.2	50.8	49.8	50.7	49.8	52.5
SVM Radial basis	61.9	60.7	55.3	61.3	51.1	49.4	49.5	50.4	55.0
LightGBM	81.4	85.6	77.4	87.4	92.6	50.0	50.0	73.2	74.7
DGCNN	**88.4**	**91.7**	**89.4**	**92.0**	**94.2**	**92.7**	**93.4**	**89.4**	**91.4**
PointNet	-	-	-	-	-	-	-	-	-
VoxelNet	-	-	-	-	-	-	-	-	-

Table 4.9: Accuracy of selected algorithms performed on the test set. All values in %. The accuracy of PointNet and VoxelNet is not included as the algorithms did not manage to converge, thus the results do not reflect the performance of the algorithms.

Algorithm	point cloud								Mean
	3	4	13	14	40	42c	44c	45c	
RF 40[1]	.117	.105	.112	.120	.109	.113	.112	.117	.113
RF 100[1]	.119	.119	.105	.107	.116	.117	.108	.118	.114
RF 400[1]	.119	.219	.227	.108	.219	.341	.329	.332	.237
RF 1000[1]	.334	.334	.455	.218	.337	.674	.775	.675	.475
SVM Linear[1]	10.0	9.09	31.0	4.79	21.1	56.2	64.1	56.4	31.6
SVM Polynomial[1]	9.73	9.47	31.4	4.98	22.0	56.1	65.8	57.7	32.1
SVM Radial basis[1]	33.8	30.7	108	15.6	72.5	188	220	193	107
LightGBM[2]	**.013**	**0.01**	**.011**	**.010**	**.009**	**.009**	**.010**	**.011**	**.010**
DGCNN[2]	.791	.844	.750	.716	.737	.758	.786	.773	.769
PointNet[2]	.075	.090	.093	.073	.073	.074	.072	.075	.078
VoxelNet[2]	22.6	24.5	24.8	22.6	21.6	21.7	22.2	22.5	22.8

Table 4.10: Forward time of selected algorithms performed on the test set. All algorithms are run on the "Private PC" with the hardware as specified by table C.1. All values are in seconds. [1] running on CPU. [2] running on GPU.

In order to compare the algorithms, their accuracy and forward time are plotted against each other in figure 4.8. The F1-score (fig. 4.7) and the confusion matrix (tab. 4.11) for the algorithms are also included to make the results for this thesis easier to compare with other future results.

SVM rbf	RF 100	LightGBM	DGCNN	PointNet	VoxelNet
$\begin{bmatrix} 40131 & 13104 \\ 38409 & 14826 \end{bmatrix}$	$\begin{bmatrix} 26888 & 26337 \\ 26234 & 26991 \end{bmatrix}$	$\begin{bmatrix} 8011 & 1349 \\ 9270 & 22627 \end{bmatrix}$	$\begin{bmatrix} 37951 & 3982 \\ 3043 & 38890 \end{bmatrix}$	$\begin{bmatrix} 12036 & 10404 \\ 10149 & 12291 \end{bmatrix}$	$\begin{bmatrix} 5 & 42094 \\ 5 & 42094 \end{bmatrix}$

Table 4.11: The confusion matrix of the selected algorithms when performing on the entire validation set. For random forest, the version with 100 threes and for SVM, the radial basis-based algorithm is selected to be displayed based on there performance as shown in table 4.9. Positive is defines as cable, while negative is background.

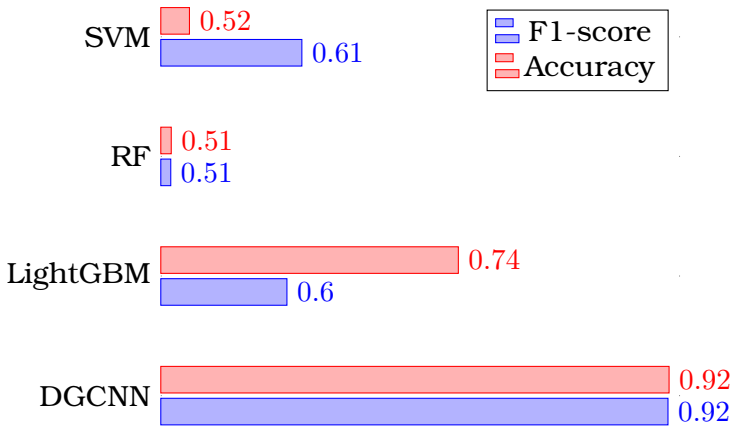


Figure 4.7: The F1-score and the accuracy of the algorithms over when evaluating the entire validation set. PointNet and VoxelNet is not included as the algorithms did not manage to converge, thus the results do not reflect the performance of the algorithms.

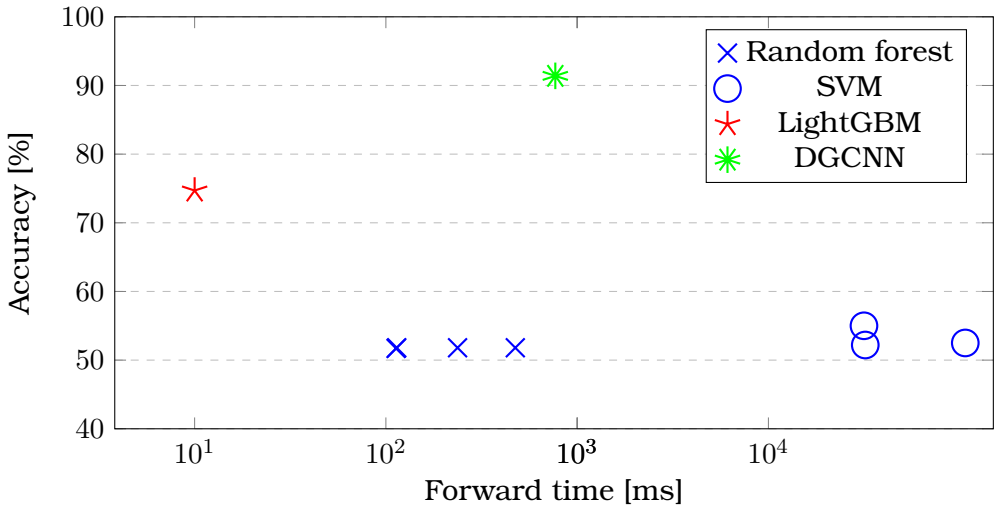
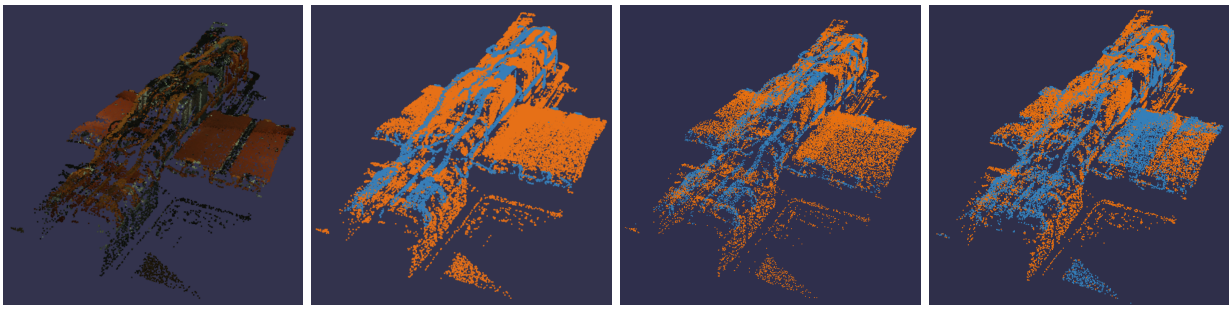
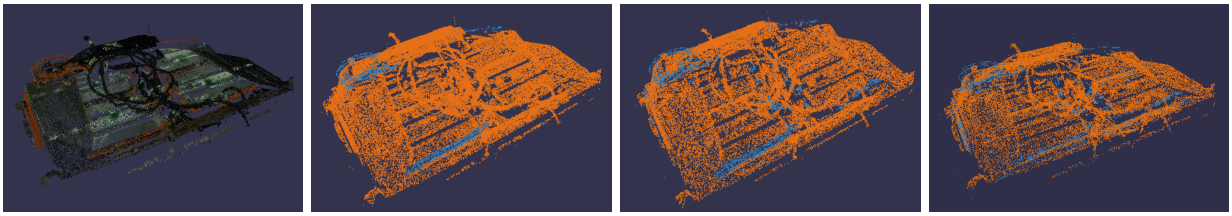


Figure 4.8: A plot of the mean accuracy (table 4.9) vs the log of the mean forwards time (table 4.10). PointNet and VoxelNet is not included as the algorithms did not manage to converge, thus the results do not reflect the performance of the algorithms.

The result of the semantic segmentation is compared to the ground truth in figure 4.9 for both batteries.



(a) Input point cloud (b) Ground truth (c) DGCNN (d) LightGBM



(e) Input point cloud (f) Ground truth (g) DGCNN (h) LightGBM

Figure 4.9: The result of the semantic segmentation of battery A (point cloud nr. 3) and battery B (point cloud nr. 45c).

4.3 Hyper-parameter selection for clustering algorithms

The correct hyper-parameters are crucial for the clustering algorithms to work properly. Wrong hyper-parameters can lead to too many or too few clusters.

Set up

A linear swipe is used in order to determine the optimal hyper-parameters for the two selected clustering algorithms. As discussed in section 3.5.6, the hyper-parameter ϵ must be larger than the voxel size of $l_{voxel} = 1.6[mm]$ as discussed in chapter 3.4.5. For the final product, the voxel size is rounded down to $l_{voxel} = 1.5[mm]$, this Therefore becomes the lower bound on ϵ. The sweep is performed with an upper bound of $\epsilon_{max} = 3 \cdot \epsilon_{min}$, and an increment of $0.1 \cdot \epsilon_{min}$.

The hyper-parameter k is linearly swiped form a minimum of $k_{min} = 10$, with an increment of 2, up to the upper bound detriment by equation 3.12 as described in section 3.5.6. For OPTICS, the hyper-parameter ξ is also linearly swiped between $\xi_{min} = 0.1$ and $\xi_{max} = 0.5$ with an increment of 0.1.

The data set is set up for semantic segmentation, thus there is no parameter to determine the instance (cluster) any point belongs to. Therefore, there is no mathematical way of evaluating the performance of the clustering algorithm. The only exception is the number of clusters, but this is far from a proper metric, and rather a filter to remove horrific attempts. The swipe is set to filter and only save the result of algorithms that produces between 5 and 30 clusters. The performance of the algorithms is then evaluated by manual inspection. The evaluation of the clustering is done on battery B as it has more cables and is therefore considered more challenging.

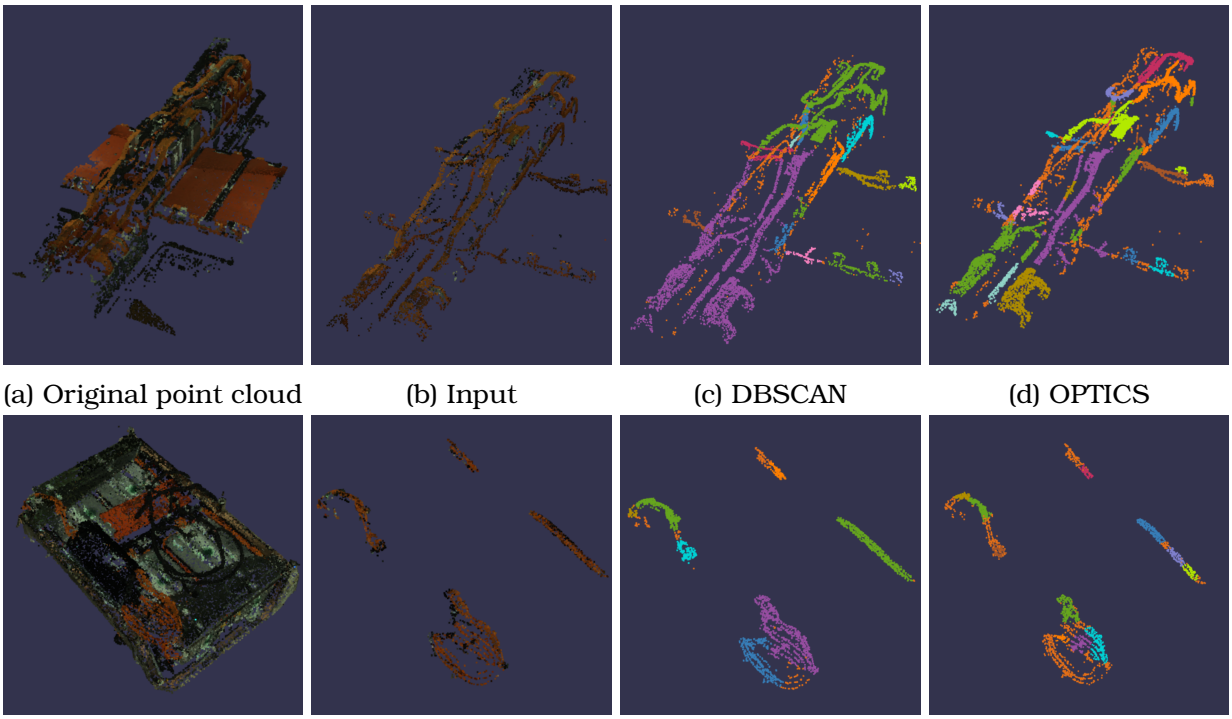
Result

The best result from the linear swipes is displayed in table 4.12.

		DBSCAN	OPTICS
Hyper-parameters	ϵ / ϵ_{max}	0.01935	0.01725
	k	31	40
	ξ	n/a	0.3
Forward time	Battery A	0.068	1.41
	Battery B	0.337	10.5

Table 4.12: Hyper-parameters and forward time for DBSCAN and OPTICS. The forward times are in seconds, and ϵ is in meters.

The clustered point clouds produced from the two algorithms with the hyper-parameters described in table 4.12, are illustrated in figure 4.10.



(a) Original point cloud (b) Input (c) DBSCAN (d) OPTICS

(e) Original point cloud (f) Input (g) DBSCAN (h) OPTICS

Figure 4.10: The result of the two clustering algorithms operating on battery A (point cloud nr. 0) and battery B (point cloud nr. 45). The two algorithms are run with the derived hyper-parameters listed in table 4.12.

Chapter 5

Discussions

5.1 Data quality

The quality of the data is a key aspect in developing a dismantling system viable for industrial usage. This section describes the aspects of the capturing system and how to repeatedly acquire good data. An example of a point cloud captured and filtered with the capturing system is shown in figure 5.1.

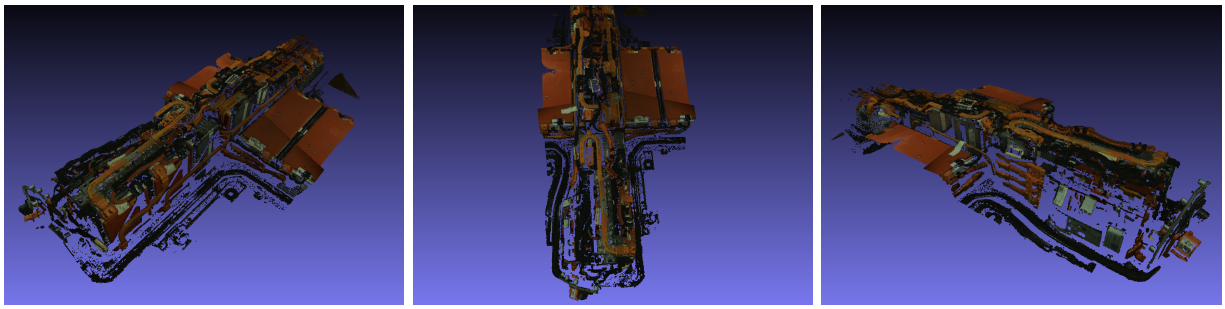


Figure 5.1: A stitched and filtered point clouds displayed from three different views (before downsampling).

5.1.1 Capture settings

Finding the proper settings for the camera were done with the intention of using an HDR solution. However, due to the unresolved problem with the capturing code, the HDR function is not available. The problem also caused the saturation limit to be considerably lower than expected. Therefore two versions of the capturing settings are found. One set of the desired setting that is optimal if not for the problem. The second being the settings found to optimal whilst working around the problem.

Due to this error, some of the data used for training data is oversaturated and having problems capturing certain features. The workaround is configured to give a good result for the orange cables since the orange cables is the intended segmentation problem. The workaround is however affecting the systems ability to register the darker components of the battery, a problem which may lead to collisions in a dismantling scenario.

A capture system using HDR functionality with the three capture settings is deemed to be a "close to optimal" solution for the point cloud capture process. One could split the mid-range capture into upper and lower mid-range captures, but from the visual inspections, the effect is minimal, while increasing the processing time. The workaround solution is however a good compromise, giving a clear representation of the cables which are the main focus of this thesis. For the detection software, the workaround solution is likely

sufficient. A problem may arise when a grasping software is to be implemented, due to the poor representation of the darker components. However, in a potential industrial setting, it would be natural to assume the saturation problem to be solved. For such a situation, an HDR solution using the three capture setting would give the best result.

5.1.2 Point-of-views

A battery pack geometry is complex on two different scales, the large scale configuration of different models and components, and the small scale with cable layouts and fasteners. To get a desirable representation of the whole battery pack, a system to handle both scales of complexity is needed. The testing from chapter 4.1, is done to address the small scale complexity. Showing that three POV's are required to give a desirable level of completeness. When increasing the number of POV's beyond three, the improvement in the point cloud is less notable.

This thesis does not guaranty the three POV approach to be applicable to all geometric features the system can encounter. The three POV system is only evaluated on how complete the cable models are, and features such as divots and gaps will likely still be a problem. However, adding more POV's are not guaranteed to solve the problem. Solving the problem by generally increasing the required POV's would be an inefficient way of increasing consistency. Increasing the POV's used on simple features while not guarantying that the camera is correctly positioned for the difficult feature.

5.1.3 Full battery coverage

As with the small scale geometric complexity, guaranteeing a "close to complete" approach is problematic. By applying the three POV approach from the small scale testing, a generalized full battery pack coverage sequence is devised. Due to this, the problem of difficult geometries would be found in the large scale approach as well. To calculate the required captures necessary to cover a whole battery pack, the geometries are simplified to a 2D plane, thus making the estimate highly generalized. The system is therefore not intended to be a "close to complete" representation, rather covering the majority of the battery surface. The rest then needs to be handled by a second capture service that can take care of the problematic areas.

The results from the number of captures estimation are also used in estimating a time to capture. As was seen from the capture estimates, how far the camera is located affects the number of capture to a large degree. By extending the distance from 1 meter to 1.6 meters, the estimate for captures goes from 57 to 27 on Battery B. However, likely, extending the distance will negatively affect the accuracy of the point cloud since 1.6 meters is the limit of the optimal working range of the Zivid One+ M. A similar effect is seen when the camera is moved closer. Since the camera generates one point per pixel, moving closer gives a more detailed and denser point cloud. However, this also increases the required number of captures.

The way the battery pack should be captured ends up being a compromise between detail and time. However, to what extent time should be prioritized over detail, is application dependent. Therefore, it is realistic to assume that a functioning dismantling plant will have separate robots for data capturing and dismantling. In such a scenario, one capture robot could service several dismantling robots. Reducing the capture time by increasing the distance to 1.6 gives the capture robot the ability to service additional dismantling robots. On the other hand, it is not known what level of detail is required for such a system to function consistently. To find the answer to how detailed a captured point

cloud needs to be for the system to function consistently, a more complete dismantling system is needed.

5.1.4 Quality of the data-set

The constructed data-set, as displayed in table B.1, consisting of 38 labelled point clouds is quite narrow as there were only two batteries available for capturing. This lead to very similar data in most point clouds, although they are captured from different viewpoints, and for battery B, are different segments of the battery. Never the less by using a distribution-based loss function, a gradient can be computed for every single point, thus making the number of computable gradients sufficient as there are over 26 million points in the data set. Some of the data is augmented with rotation to embed rotational invariance. The usage of augmented data is recommended as it increases performance on geometric data in 2D [132] and is, therefore, likely to do the same in 3D where several newer papers on the topic have promising indications [133]–[135]. Alternatively, a more trendy solution of using a generative adversarial network for generating augmented training data, something that has shown success for 2D images [136].

The size of the current point clouds might be too high, thus overwhelming models developed to work on smaller point clouds with $2000 - 4000$ points which is used in the benchmarks [31]. On the other side, reducing the point clouds leads to information loss. In a future version of the system that will distinguish smaller objects like clips and bolts, in addition to cables, this information loss might be devastating. Therefore, the current system should be developed to work on large point clouds. Hence, the models should be adapted to fit the problem, and not the other way around. In the extreme case of there not being any suitable model for large point clouds, there should be developed new algorithms. Optimally, a hybrid algorithm that can work on a heavy downsampled version of the point cloud for detecting large elements while also working on small sections of the original point cloud to find small local features.

Ideally, and for full-scale implementation, there should be a more diverse data-set with no more than two or three instances of each battery. A new data set should also account for more classes, thus setting up for a transition into a multi-label classification system. Ideally, the points should also be grouped and labelled with an object id to provide a metric for evaluating the hyper-parameters of the clustering algorithms.

Nevertheless, the usage of per-point labelling tools [114] showed to be an efficient way of labelling data as the entire data set was captured and labelled in under one week. The current data set also serves its purpose as this thesis only intend to compare and evaluate the performance between algorithms, and thus do not need to reach industry levels. The low variance of the data set is likely to affect the performance of all algorithms to more or less the same degree, thus allowing for a fear comparison between the algorithms.

5.2 Perception system

The two parts of the perception system have individual aspects that influence their performance as well as each others performance. Design choices and test results of the semantic segmentation algorithms and the clustering algorithms, and how they relate to each other is discussed in this section.

5.2.1 Semantic segmentation algorithms

From the results shown in table 4.9 it is clear that the traditional algorithms (random forest and SVM) do not manage to provide any relabel results. Throughout runs with several different hyper-parameters, none of the random forest algorithms managed to reach over 55% accuracy on any of the data in the validation set. Keep in mind that a random guess would result in an accuracy of 50%, thus making the random forest a week classifier in this context. The random forest algorithms do not appear to be affected by the number of trees as the average accuracy of all versions are stable around $51.7-51.8\%$. Nevertheless, the random forest algorithms score well on forward time, as shown in table 4.10.

The SVM algorithms performed marginally better than the random forest and are also seen as week classifiers. From the three implementations of SVM, radial based, linear and polynomial (2. order), the radial basis-based one performs notably better on test point cloud 3, 4 and 14. This is likely to be a consequence of the simpler geometry of battery A and there is a combination of curvature and colour in a smaller region that indicates cable. This supports the assumptions that were made in section 3.5.5 about the possibility of segmenting the cables with classical approaches based on the t-SNE transformations shown in figure 3.16a and 3.16b. On the other side, all three implementations of SVM show extremely poor time usage with time upwards of several minutes.

As expected, the traditional classifiers show an overall poor performance. Neither managing to generalise as none of them performs sufficiently better than a pure guess (50%) on the augmented point clouds 42c, 44c and 45c as shown in table 4.9. Therefore, neither random forest nor SVM can in their simplest form be used for semantic segmentation of point clouds of this kind of setting. On the other side, they constantly manage to perform slightly better than a random guess, thus fulfilling the requirements for being weak classifiers in a boosting algorithm [101].

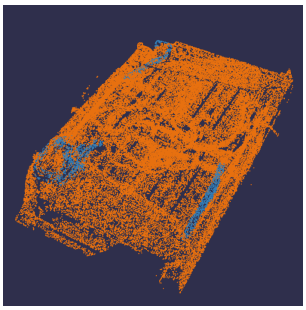
The boosting algorithm, LightGBM, show surprisingly good performance both within accuracy and forward time. With accuracy up towards 90% and an average forward time of 10 [ms], the algorithm clearly outperforms the traditional classifiers, thus showing the potential of boosting-based algorithms. Nevertheless, the algorithm fails to generalise as the augmented versions of the data 42c and 44c only have a 50% accuracy. The good results on the rest of the data is a result of too little variance between the training data and the validation data, thus the expels are hard learned. Gradient boosting is therefore nether a good algorithm for semantic segmentation.

The fact that neither the boosting algorithm nor the traditional classifiers manage to generalise supports the idea of PointNet [22] saying that the semantic value of any single point depends on more than the point features alone. Thus to classify any point, one needs knowledge about the global features in combination with the local features. In other words, a point cloud is more than the sum of its parts.

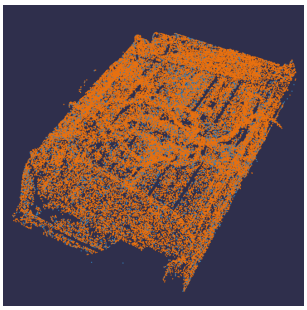
Finding the correct hyperparameters for training turned difficult as no other known projects are working on point clouds of the same size. The parameters used in the original papers of DGCNN [98], PointNet [22] and VoxelNet [118] were meant for smaller point clouds. This lead to several runs not converging and ending up with a nonsense prediction as shown in figure 5.3c. For the DGCNN algorithm, the original value for k, the number of neighbours in a k-NN graph, was set to 20 and 40. As shown in figure 5.3, this did not lead to any usefully results in this application as the loss settled around 0.24 (converting go down to 0.05) with an accuracy of around 50%. The inability to converge is a result of each node not reaching far enough at each stage of message passing. Thereby they do not manage to capture broad enough features, and the algorithm does

not manage to distinguish points. In other words, the algorithm is supposed to grasp local features at each stage of the message passing, but the features become too local. Instead of capturing more complex features like curves, the algorithm only manages to capture more primitive shapes like plane surfaces. In general, a deeper neural network picks up more complex features than a shallow one, so adding additional layers of message passing could solve the problem. Graph convolution is shown to dramatically drop in performance when the number of layers increases [58], [137]. This is a result of the nodes features averaging out, thus all nodes become equal, and not an effect of the vanishing gradient problem [58], [137]. Increasing k, as suggested by the original paper [98], to then include a larger amount of features showed to help. The value of $k = 100$ turned out to work well and is what is been used for the thesis. Due to the size of the input point clouds, adding an additional layer could be justified, but is not done in this thesis.

(a) The loss curves for DGCNN.



(b) The ground truth.



(c) Output for $k = 20$.

Figure 5.3: The result of the DGCNN algorithm running with original hyper-parameters.

Initially, the binary cross-entropy loss function was used, as described in section 3.5.5. The model did not manage to learn due to the extreme value of the loss function. This is explained by the way the binary cross-entropy loss function punishes wrong guesses, namely in an exponential fashion. As shown by equation 2.21, the value of the binary cross-entropy function tends towards infinity when the error reaches one. This is normally a good feature as it punishes wrong estimations harder. However, when the loss is calculated over hundreds of points at a time, and the network is in the early phases of learning, at least one prediction will likely by chance be far off, thus resulting in an extreme loss. Although the loss is averaged over the entire point cloud, at least one point will be too strong of an influence due to the exponential nature of the binary cross-entropy loss function. Alternatively, the training could be conducted by mini-batch training. This was tested for batches of size 32 and 64 points, but this failed to produce proper gradients. The previously proposed focal-loss (chapter 3.5.5) builds on the binary cross-entropy loss function, thus it also fails for the same reasons. The mean square error loss function (2.19) act in a quadratic way, thus do not suffer from the venerability of a single point having a large error while still punishing wrong guesses progressively harder in a non-linear way. Although the mean square error loss function is most commonly used in regression problems, it showed a good effect when applied in this thesis. The characteristic of the binary cross-entropy loss function and the mean square error loss function are potted in figure 5.2

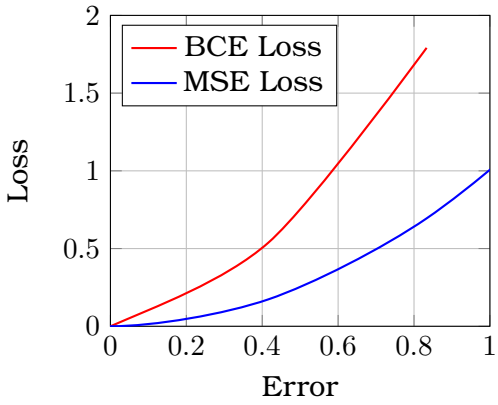


Figure 5.2: A plot of error vs loss for the binary cross-entropy loss function (BCE Loss) and the mean square error loss function (MSE Loss).

(a) The ground truth. (b) PointNet prediction.

Figure 5.4: The results of PointNet.

Niter the PointNet model nor the VoxelNet model managed to converge to a useful result. A comprehensive amount of hyper-parameters are tested, as shown in table 5.1, but neither manages to make any of the two algorithms produce any useful results. The output prediction of PointNet is shown as an example in figure 5.4. The reason why neither of the two algorithms manages to converge is not known, but this is an indication that the algorithms are intended for smaller point clouds. The PointNet algorithm intends to produce a global feature vector that is later used for determining the point semantic value. When training on chunks rather than the entire point cloud, this global feature vector might not be able to provide a good representation. The VoxelNet algorithm is an attempt to reconstruct SEGCloud, but its training is based on the hyper-parameters of the original VoxelNet algorithm. This might be a source of error as the original VoxelNet algorithm is trained with a region-based loss function rather than a distribution-based one.

On the other side, a solution to PointNet is to produce a global feature vector before the point cloud is divided into chunks. Thereby allowing the algorithm to have a proper representation of the entire point cloud when performing semantic segmentation. The SEGCloud algorithm could be reconstructed with another backbone than VoxelNet. It is not certain that another backbone like VoxNet [96], would enhance the performance, but it is a possible source of error. Neither to say, a larger training set might have helped on improving the results.

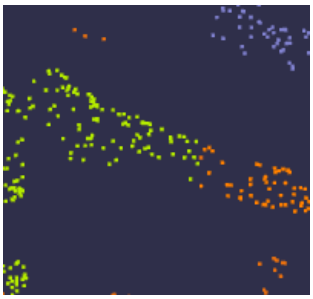
Parameter	Values
Learning rate	0.1, 0.01, 0.001
Batch size	16, 32, 512, 1024, One chunk
Number of epochs	200, 300, 500
Loss function	BCE, MSE, Focal-Loss, NLLL
Output layer activation function	Soft-Max, Log Soft-Max, Sigmoid, Tanh
Scheduler type	Cosine annealing, step ($\gamma = 0.9$)
Scheduler period	30, 50, 200
Solver	SGD, Adam

Table 5.1: The following configurations were tested when attempting to make PointNet and VoxelNet converge to satisfying performance.

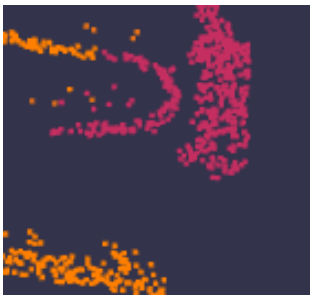
5.2.2 Clustering algorithms

Neither of the two proposed clustering algorithms, DBSCAN and OPTICS, is successful as neither manages to separate all cables. As seen in figure 4.10, both algorithms struggle with working on both batteries and following the cable paths all the way. Finding the optimal parameters for density-based clustering turned hard as the cables are located

too close in space so that cluster "jump" from one to another as shown in figure 5.5b. On the other side, the clusters do not always manage to follow the cables all the way due to a small less dense section. This leads to the same cable being included in several clusters as shown in figure 5.5a. These two problems of "jumping" clusters and discontinues clusters make adjusting the parameter ϵ ambiguous. Since there is no other good alternative stand-alone clustering algorithm, as discussed in section 3.5.6, there is also no good working clustering algorithm for this application with the current circumstances.



(a) The clustering is discontinuous.



(b) Gaps are too small.

Figure 5.5: Cutout form the figure 4.10d.

Chancing ether the downsampling strategy or scale would most likely allow for DBSCAN and OPTICS to work better. Currently, random sampling is used due to the time efficiency, but a triad off could be made to get a more uniform distribution in the point cloud. This would reduce the size of gaps, thus allowing for a smaller value of ϵ as the formation of clusters would not be stopped by sparse regions caused by random processes. It is unclear what the trade-off in time would be to get a more uniform point cloud, but is a potential solution to the clustering problem.

Operating on larger point clouds would lead to more dense regions without chaining the gaps between cables, thus allowing for a lower value of ϵ. The cluster forming process would then take small steps, but the higher density point cloud would allow for the cluster forming to work. The immediate disadvantage is the increase in forward time and memory usage. This solution, therefore, becomes application dependent as the exact cost of increased time and memory usage depends on industrial application. Nevertheless, the filtering process and the density of the point cloud are important factors for making the density-based clustering algorithms work.

Alternatively, setting more constraining hyper-parameters to form many small clusters, thus avoid "jumping" in the clustering forming process. As long as the clusters have some sort of length, a separate algorithm can estimate a centre line within the cluster. Clusters can then be merged if their centre lines are within some constraints. Similar approaches are done for 2D-images [138], and it is, therefore, a promising solution for 3D point clouds. This is a hypothetical solution, and the time and memory usage is not looked into.

Nevertheless, voxelising the input point cloud gives a common ground for the clustering algorithms to work, regardless of the amount of down-sampling. By evenly spacing points with a known distance, the clustering algorithms get similar conditions, regardless of the captured object and the capturing hardware.

The semantic segmentation algorithm should be developed towards having high precision, as the two clustering algorithms show to be vulnerable to variance in density. Therefore, the semantic segmentation algorithm should provide a sable density regardless of the input. By expanding the data-set to include objects as described in section 5.1.4,

the semantic segmentation algorithms can be trained to have an even density within the objects. In other words, a loss function based that does not look at the overall accuracy of the algorithm, but at how well balanced the accuracy between the objects are. If all objects have the same accuracy, and a voxel grid is applied, the density should be more or less the same and the density-based clustering algorithms would work better.

Although the two clustering algorithms have the same time complexity, when comparing them based on the data from table 4.12 and figure 4.10, DBSCAN shows to be superior and the only algorithm that meets the design criteria described in section 3.5.1. Nevertheless, DBSCAN suffers more from "jumping" as OPTICS is better at discovering changes in density. This is shown in figure 4.10 where OPTICS manages to follow the path of the cables better than DBSCAN. On the other side, with the discussed solution of operating on larger point clouds, DBSCAN could operate with a significantly lower value of ϵ. OPTICS would then be too slow for any practical application, if not a GPU-based implementation can be used.

5.3 Industrial viability

For an autonomous dismantling plant to be practical and cost-effective, it is critical for the system to work with such consistency that it can be trusted to complete the task without supervision. For an end-to-end model-free autonomous LIB battery dismantling plant to reach this level, three key challenges would need to be solved, of which two have been a focus for this thesis.

The first challenge is how to guarantee that the generated point cloud from the capture process is an "as close to complete" representation of the battery pack. If the system can not do this with the desired consistency, it is expected that the system would fail when a component is left un-scanned. To solve this problem, a more complex capturing sequencing algorithm is likely needed, one which could adapt the battery pack model during capturing.

When the capturing problem is solved, the process of perceiving the components is the next challenge. If the perception system is not consistent, the system will fail in much the same way as with the capturing system, not "knowing" about the component the system is supposed to remove. The classification will need to handle a larger array of labels, not only "cable" and "not cable". It is also of critical importance that the software does not miss label components, since a miss labelled part can cause the dismantling tools to break when wrongly applied.

The last challenge, for an autonomous dismantling plant, is how to handle the wide array of dismantling sequences it will need to complete. Thou not a topic discussed in this thesis, some system of handling complex manipulations will likely be necessary if a model-free dismantling plant is desirable. Due to the large variation in both layout and component design in battery packs, making a custom dismantling sequence for every component would be impractical. Therefore some system that can adapt a dismantling sequence to its current problem is necessary.

5.3.1 Industrial capturing solution

A possible solution to reduce time requirement is to partially go away from an eye-in-hand approach for capturing, but instead installing three stationary cameras and moving the battery pack. Since the three cameras can capture essentially at the same time, only

using $10\,ms$. each, it can be thought of as one capture, reducing the required captures to a third. Since most of the time used in the capture sequence is likely to be the robot path execution, going away from an eye-in-hand system will likely save a considerable amount of time.

However general static camera solution is likely to be insufficient to guarantee an "as close to a complete" representation. Since the battery packs can have tight spaces, or partially obstructed, where a camera needs to specially aligned to get a good capture. A possible solution is to brute force the problem by increasing the number of pivot points (the point that all the cameras point towards), to a level where the chance of a camera not aligning correctly for a feature, is below the desired failure rate. But this approach leads to a too inefficient system, using most of the time on redundant captures.

Likely, a more efficient approach is to combine a static camera system with an eye-in-hand robot. Where the static system covers a large majority of the battery pack surface and the eye-in-hand robot handles the remaining problematic areas. The eye-in-hand robot needs to have some system of judging where the problematic areas are located, what position is best to capture that area and what level degree of detail is deemed sufficient. However, developing such an algorithm will probably require a considerable amount of work.

5.3.2 Industrial perception system

As stated in the design criteria (section 3.5.1) the perception system must be accurate but also fast. Only the two algorithms that provide good results, LightGBM and DGCNN, are compared as the others are outclassed as shown in figure 4.8. The true accuracy of any of the algorithm is truly higher than what is presented because of miss-labelling in the process of creating the data sets. Therefore, the quality of the semantic segmentation must also be evaluated by visual inspection. As shown in figure 4.9, the LightGBM algorithm does not manage to distinguish the orange panels from the orange cables in all cases. Also when looking at table 4.9 LightGBM looks to memorise the test set rather than learn as it performs poorly on the augmented data. Nevertheless, the LightGBM shows an impressive forward and outperforming all the other algorithms. With a larger and more diverse data set, the LightGBM algorithm might work in an industrial setting. It depends mainly on how diverge the geometry of the EV battery packs are. If most follow a standard look and design, the LightGBM could with a sufficient amount of training data in combination with more engineered features be viable in an industrial setting.

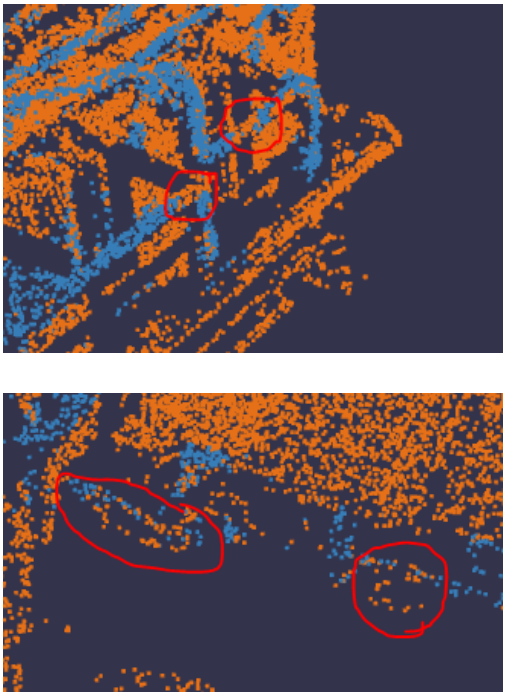


Figure 5.6: Cutouts from figure 4.9c showing examples of DGCNN miss-labeling.

Nevertheless, the DGCNN algorithm shows a significantly better quality of the semantic segmentation, both in terms of accuracy but by visual inspection. There are still in-perfections in DGCNN, as shown in figure 5.6 where the algorithm fails to find parts of the cable. The discontinuous segmentation causes problems for the clustering algo-

rithms as previously discussed. The algorithm used is prom to some overfitting that makes it perform slightly worse on the validation set. Also, due to the low dissimilarity within the training data, the algorithm was never challenged with hard examples that would help push the performance. In an industrial setting, the algorithm, therefore, needs to be trained on a more realistic data-set that contains a larger diversity of battery packs.

From a time perspective, a perception system consisting of a combination of a PCSS and a clustering system is well within the time requirements and is therefore viable for industrial applications. If using DBSCAN and DGCNN, the total time usage is 1.1 seconds for the large battery. With additional prepossessing and filtering added, the total time will not exceed two seconds. Depending on the application, this might allow for the usage of larger point clouds as would be beneficial for the clustering algorithm.

Chapter 6

Conclusions

To address the issue of obtaining a good digital representation of battery packs, this thesis divided the problem into three sub-problems: how to optimize point cloud quality for individual captures, how many points-of-view are required for the system to adequately represent cable features, and how to consistently capture the entire battery pack.

When using a structured light camera, it has been recommended to use HDR capturing to achieve a desirable point cloud quality. This thesis presented a three-capture HDR sequence that captures the full dynamic range of the test battery packs with minimal loss and artifacts. When HDR capturing is not available, a single capture system focused on saturation elimination is presented as an alternative. The alternate settings presented capture the cables and bright elements with a desirable level of detail but struggle with dark features.

Three points of view are thought to be required for a desirable representation of geometric features such as cables. The half-cylinder representation of the cables is not provided by the three POV solutions. Adding more point-of-views is deemed inefficient, as there is little improvement in the point cloud while adding time to the capturing process. Though there is a case to be made that having more points of view makes the system more reliable.

This thesis presented a general approach to covering the majority of the surface details of the battery pack. The battery packs are simplified to be 2D horizontal planes in order to estimate the required number of captures and time usage. The estimate determines how many captures are required to cover the area from three different perspectives. With the recommended setup, the time to capture is estimated to be 120 second for the smaller battery pack (Battery A), using 12 captures, and 570 seconds for the larger battery pack (Battery B), using a total of 57 captures. This thesis does not imply that the general approach would be consistent in covering any battery pack surface, but rather that an approach to cover the majority would be provided. To identify and cover potentially overlooked regions, an improved algorithm would be required.

Furthermore, the clustering algorithm has been shown to consume more time than the semantic segmentation algorithms. As a result, for instance segmentation of point clouds, the PCSS-cluster approach is preferred over the cluster-class approach.

This thesis demonstrated that it is possible to extend the DGCNN algorithm to large point clouds while still producing promising results by successfully implementing it on a point cloud of $50,000$ points. However, larger point clouds may be required to build a functional clustering system, so the algorithm must be extended further to operate on larger point clouds. Furthermore, the poor performance of traditional classifiers and the success of DGCNN point to PointNet-based architectures that embed local and global features as a promising method of performing PCSS.

The thesis also demonstrated how the preparation of the point cloud and the PCSS algorithms affects the performance of the clustering algorithms. The distance between the points in the point cloud should be much less than the smallest gap between any two cables in any battery pack. As a result, instance segmentation necessitates a minimum density in the point cloud. Other approaches that do not necessitate a significant increase in density are also discussed, but they need to be investigated further.
Overall, the thesis demonstrated that 3D-instance segmentation is a viable and promising alternative to traditional 2D segmentation. However, more work is needed to bring the field to a desirable level of robustness.

Further work

While working on this thesis, the scope of the project has been changing back and forth. But one thing has been clear, a lot more development is needed for a fully automated battery dismantling plant. If work on this thesis were to continue, the following topics would be a recommended ways forward.

Capture sequencing

If the project is continued, incorporating a system for finding the optimal sequence of points-of-view for the camera is advised. If an eye-in-hand approach is to be continued incorporating an algorithm like next-best-scan is a potential way forward. A next-best-scan system could ensure the reliability of the point cloud generation, and reduce the time needed to generate point clouds.

A potential way forward could be to focus on a hybrid static/eye-in-hand solution, where a static camera system do the rough surface coverage, whilst an eye-in-hand system covers problematic areas, with some kind of next-best-scan algorithm.

Stronger performance of the perception system

There are several algorithms and approaches to PCSS that are not tested in this thesis that have a good performance on the large benchmarks. Auto-encoder-based and attention-based architectures are interesting due to their success in the 2D-equivalent. The RandLA-Net [121] is a promising candidate that should be tested. In addition, the field of PCSS is growing and new algorithms and approaches are published regularly. As an example, five out of the top-10 algorithms on the ModelNet40 benchmark [31] were published during the writing of this thesis (as of 26.05.2021).

The field of semi-supervised learning where supervised and unsupervised learning is combined is also promising. This can be accomplished by first pre-training a PSCC algorithm in an unsupervised fashion on the large benchmarks to train a feature aggregation network, and then train a classification network in a supervised fashion on the battery-pack data set B.1. Similar projects are conducted by Kipf et al. [97].

A larger and more diverse data set would help to make the algorithms more robust. The data-set should also be divided into objects in addition to labels. This would allow for a mathematical evaluation of the clustering algorithms, thus making it easier to find good hyper-parameters.

A more purpose-engineered loss function that would help the PCSS algorithm and the clustering algorithm to work better together can help make the results more viable.

Appendix A

Loss curves

The focus of the thesis is not to reach high performance, but to show that training on large point clouds is possible. Therefore are the loss-curves not included in the main rapport, but included in the appendix for future comparisons.

The loss curve of DGCNN, as seen in figure A.1, show a tend to overfit, thus the training could have ended earlier and provided stronger results and a more robust algorithm.

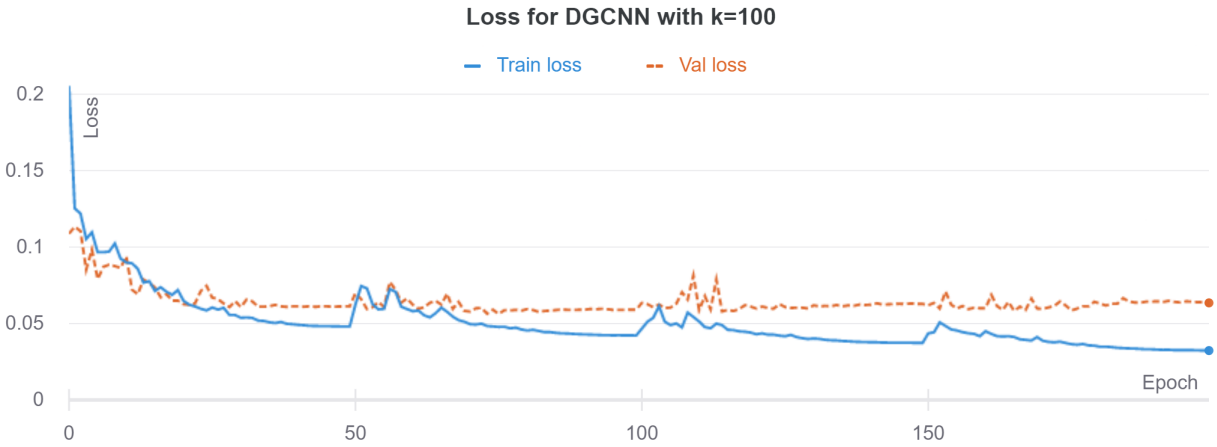


Figure A.1: The loss curve for DGCNN with k=100. This is the model used in the thesis.

Appendix B

The data-set

Name	Size	Amount cable	Voxel size	Battery	Rotation	Usage
0	$388,875$	4.28%	$2.5[mm]$	A	None	Training
1	$243,224$	4.39%	$2.5[mm]$	A	None	Training
2	$101,702$	2.23%	$2.5[mm]$	A	None	Training
3	$169,400$	3.67%	$2.5[mm]$	A	None	Validation
4	$273,907$	3.45%	$2.5[mm]$	A	None	Validation
5	$268,043$	13.8%	$2.5[mm]$	B	None	Training
6	$419,222$	13.2%	$2.5[mm]$	B	None	Training
7	$181,224$	14.4%	$2.5[mm]$	B	None	Training
8	$225,706$	15.4%	$2.5[mm]$	B	None	Training
9	$563,830$	13.4%	$1.5[mm]$	B	None	Training
10	$107,387$	6.19%	$1.5[mm]$	B	None	Training
11	$695,599$	14.3%	$1.5[mm]$	B	None	Training
12	$625,414$	13.1%	$1.5[mm]$	B	None	Training
13	$556,884$	12.9%	$1.5[mm]$	B	None	Validation
14	$422,214$	1.93%	$1.5[mm]$	A	None	Validation
15	$556,730$	3.02%	$1.5[mm]$	A	None	Training
16	$616,171$	10.8%	$1.5[mm]$	B	None	Training
40	$1,144,795$	8.75%	$1.5[mm]$	A	None	Validation
40a	$1,144,795$	8.75%	$1.5[mm]$	A	$20/0/0$	Training
40b	$1,144,795$	8.75%	$1.5[mm]$	A	$40/0/0$	Training
40c	$1,144,795$	8.75%	$1.5[mm]$	A	$40/40/0$	Training
40d	$1,144,795$	8.75%	$1.5[mm]$	A	$0/40/0$	Training
42	$967,833$	22.9%	$1.5[mm]$	B	None	Training
42a	$967,833$	22.9%	$1.5[mm]$	B	$20/30/90$	Training
42b	$967,833$	22.9%	$1.5[mm]$	B	$20/30/0$	Training
42c	$967,833$	22.9%	$1.5[mm]$	B	$90/10/0$	Validation
42d	$967,833$	22.9%	$1.5[mm]$	B	$90/20/20$	Training
43	$301,137$	26.1%	$1.5[mm]$	B	None	Training
43a	$301,137$	26.1%	$1.5[mm]$	B	$90/10/0$	Training
44	$890,997$	25.9%	$1.5[mm]$	B	None	Training
44a	$890,997$	25.9%	$1.5[mm]$	B	$60/20/40$	Training
44b	$890,997$	25.9%	$1.5[mm]$	B	$70/10/10$	Training
44c	$890,997$	25.9%	$1.5[mm]$	B	$90/0/0$	Validation
44d	$890,997$	25.9%	$1.5[mm]$	B	$90/0/40$	Training
45	$1,196,458$	22.9%	$1.5[mm]$	B	None	Training
45a	$1,196,458$	22.9%	$1.5[mm]$	B	$30/90/60$	Training
45b	$1,196,458$	22.9%	$1.5[mm]$	B	$50/100/20$	Training
45c	$1,196,458$	22.9%	$1.5[mm]$	B	$70/50/40$	Validation
Total	$26,821,763$	17.3%	N/a		N/a	8 Val, 30 Train

Table B.1: An overview of the data captured and used in this thesis.

Appendix C

Hardware specifications

Computer	Lab PC	Private PC
Operation system	Ubuntu 16	Windows 10
GPU	NVIDIA GeForce RTX 2080	NVIDIA GeForce RTX 2070
Video Memory	8GB GDDR6	8GB GDDR6
CPU	Intel core i7-9700	AMD Ryzen 5 3600 6-Core
Memory	16GB DRAM	16GB DDR4

Table C.1: Hardware specification.

Bibliography

[1] OFV, *OFV | Opplysningsrådet for veitrafikken*. [Online]. Available: https://ofv.no/.

[2] SSB, *Bilparken*. [Online]. Available: https://www.ssb.no/transport-og-reiseliv/landtransport/statistikk/bilparken.

[3] G. Harper, R. Sommerville, E. Kendrick, L. Driscoll, P. Slater, R. Stolkin, A. Walton, P. Christensen, O. Heidrich, S. Lambert, A. Abbott, K. Ryder, L. Gaines, and P. Anderson, "Recycling lithium-ion batteries from electric vehicles," *Nature*, vol. 575, no. 7781, pp. 75–86, 2019, ISSN: 0028-0836. DOI: 10.1038/s41586-019-1682-5. [Online]. Available: http://www.nature.com/articles/s41586-019-1682-5.

[4] R. Lib, "Li -ion battery (LIB) R ecycling," 2019.

[5] *The University of Agder dismantles electric car batteries for reuse - Universitetet i Agder*. [Online]. Available: https://www.uia.no/en/news/the-university-of-agder-dismantles-electric-car-batteries-for-reuse.

[6] H. Engel, R. Hensley, S. Knupfer, and S. Sahdev, "Charging Ahead : Electric-Vehicle Infrastructure," *McKinsey & Company*, no. Exhibit 1, pp. 1–8, 2018.

[7] K. A. Tsintotas, L. Bampis, and A. Gasteratos, "Modest-vocabulary loop-closure detection with incremental bag of tracked words," *Robotics and Autonomous Systems*, vol. 141, p. 103 782, 2021, ISSN: 09218890. DOI: 10.1016/j.robot.2021.103782. [Online]. Available: https://doi.org/10.1016/j.robot.2021.103782.

[8] A. Thomas, F. Mastrogiovanni, and M. Baglietto, "MPTP: Motion-planning-aware task planning for navigation in belief space," *Robotics and Autonomous Systems*, vol. 141, p. 103 786, 2021, ISSN: 09218890. DOI: 10.1016/j.robot.2021.103786. [Online]. Available: https://doi.org/10.1016/j.robot.2021.103786.

[9] *OpenAI's 'state-of-the-art' system gives robots humanlike dexterity | VentureBeat*. [Online]. Available: https://venturebeat.com/2018/07/30/openais-state-of-the-art-system-gives-robots-humanlike-dexterity/.

[10] *ROS.org | Powering the world's robots*. [Online]. Available: https://www.ros.org/.

[11] *MoveIt Motion Planning Framework*. [Online]. Available: https://moveit.ros.org/.

[12] *The Open Motion Planning Library*. [Online]. Available: https://ompl.kavrakilab.org/.

[13] H. Poschmann, H. Brüggemann, and D. Goldmann, "Disassembly 4.0: A Review on Using Robotics inăDisassembly Tasks as a Way of Automation," *Chemie-Ingenieur-Technik*, vol. 92, no. 4, pp. 341–359, 2020, ISSN: 15222640. DOI: 10.1002/cite.201900107.

[14] Y. Xie, J. TIAN, and X. X. Zhu, "Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation," *IEEE Geoscience and Remote Sensing Magazine*, pp. 1–20, 2020, ISSN: 21686831. DOI: 10.1109/MGRS.2019.2937630.

[15] J. Zhang, X. Lin, and X. Ning, "Svm-based classification of segmented airborne lidar point clouds in urban areas," *Remote. Sens.*, vol. 5, pp. 3749–3775, 2013.

[16] S. Canaz Sevgen, "Airborne lidar data classification in complex urban area using random forest: A case study of bergama, turkey," *International Journal of Engineering and Geosciences*, vol. 4, pp. 45–51, 2019. DOI: 10.26833/ijeg.440828.

[17] Z. Li, L. Zhang, R. Zhong, T. Fang, L. Zhang, and Z. Zhang, "Classification of urban point clouds: A robust supervised approach with automatically generating training data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, Mar. 2017. DOI: 10.1109/JSTARS.2016.2628399.

[18] J. F. Lalonde, R. Unnikrishnan, N. Vandapel, and M. Hebert, "Scale selection for classification of point-sampled 3d surfaces," pp. 285–292, 2005. DOI: 10.1109/3DIM.2005.71.

[19] G. Riegler, A. O. Ulusoy, and A. Geiger, "Octnet: Learning deep 3d representations at high resolutions," 2017. arXiv: 1611.05009 [cs.CV].

[20] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–11, Jul. 2017, ISSN: 1557-7368. DOI: 10.1145/3072959.3073608. [Online]. Available: http://dx.doi.org/10.1145/3072959.3073608.

[21] H.-Y. Meng, L. Gao, Y. Lai, and D. Manocha, "Vv-net: Voxel vae net with group convolutions for point cloud segmentation," 2019. arXiv: 1811.04337 [cs.GR].

[22] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pp. 601–610, 2016. DOI: 10.1109/3DV.2016.68.

[23] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," 2018. arXiv: 1711.09869 [cs.CV].

[24] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," 2019. arXiv: 1801.07829 [cs.CV].

[25] G. Te, W. Hu, Z. Guo, and A. Zheng, "Rgcnn: Regularized graph cnn for point cloud segmentation," 2018. arXiv: 1806.02952 [cs.CV].

[26] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, "Semantic3d.net: A new large-scale point cloud classification benchmark," 2017. arXiv: 1704.03847 [cs.CV].

[27] X. Roynard, J.-E. Deschaud, and F. Goulette, "Paris-lille-3d: A large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification," 2018. arXiv: 1712.00032 [cs.LG].

[28] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[29] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "3d semantic parsing of large-scale indoor spaces," pp. 1534–1543, 2016. DOI: 10.1109/CVPR.2016.170.

[30] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. NieSSner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," 2017. arXiv: 1702.04405 [cs.CV].

[31] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, *3D ShapeNets: A Deep Representation for Volumetric Shapes*, https://modelnet.cs.princeton.edu/download.html, (Accessed on 05/10/2021).

[32] F. G. Zanjani, A. Pourtaherian, S. Zinger, D. A. Moin, F. Claessen, T. Cherici, S. Parinussa, and P. H. N. de With, "Mask-MCNet: Tooth Instance Segmentation in 3D Point Clouds of Intra-Oral Scans," *Neurocomputing*, no. xxxx, 2021, ISSN: 09252312. DOI: 10.1016/j.neucom.2020.06.145. [Online]. Available: https://doi.org/10.1016/j.neucom.2020.06.145.

[33] Zivid, "Zivid One+ Technical spesification," pp. 1–24, 2021.

[34] M. Atzmon, H. Maron, and Y. Lipman, "Point convolutional neural networks by extension operators," *arXiv*, vol. 37, no. 4, 2018, ISSN: 23318422.

[35] *Spatial Transformation Matrices*. [Online]. Available: https://www.brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html.

[36] A. Taqi, H. M.A.A.Al-Assadi, and A. A. Mat Is, "An Improved Adaptive Kinematics Jacobian Trajectory Tracking of a Serial Robot Passing Through Singular Configurations," *Advanced Strategies for Robot Manipulators*, no. August, 2010. DOI: 10.5772/10200.

[37] C. Torras, *Reinforcement Learning of Bimanual Robot Skills*. 2020, vol. 134, ISBN: 978-3-030-26325-6 978-3-030-26326-3. [Online]. Available: http://link.springer.com/10.1007/978-3-030-26326-3.

[38] *Concepts | MoveIt*. [Online]. Available: https://moveit.ros.org/documentation/concepts/.

[39] S. M. LaValle, *Planning algorithms*. 2006, vol. 9780521862, pp. 1–826, ISBN: 9780511546877. DOI: 10.1017/CBO9780511546877.

[40] Q. Dai and Z. Feng, *Open Motion Planning Library: A Primer*, 2-3. 2011, vol. 190, pp. 206–211.

[41] M. A. Lopez, M. E. A. L, and S. T. Mar, "Ph . D . Dissertation - Reactive Evolutionary Path Planning for Autonomous Surface Vehicles in Lake Environments Reactive Evolutionary Path Planning for Autonomous Surface Vehicles in Lake Environments on," Ph.D. dissertation, 2019.

[42] Ø. Skotheim and F. Couweleers, "Structured light projection for accurate 3D shape determination," *Proc. 12th Int. Conf. Exp. Mech*, 2004. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Structured+light+projection+for+accurate+3D+shape+determination#0.

[43] S.-A. Dragly, *How Structured Light works - Part 1.* [Online]. Available: https://blog.zivid.com/how-structured-light-works-part-1.

[44] *Optical properties of materials - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/98828689/Optical+properties+of+materials.

[45] *Introduction to Stops - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/98763232/Introduction+to+Stops.

[46] *Exposure Time - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/99450881/Exposure+Time.

[47] *Aperture - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/451149880/Aperture.

[48] *Brightness - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/98796105/Brightness.

[49] *Gain - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/100007959/Gain.

[50] *Noise filter - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/484016139/Noise+filter.

[51] *Detectable Light Intensity in a Camera Capture - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/98795895/Detectable+Light+Intensity+in+a+Camera+Capture.

[52] *Outlier filter - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/482345089/Outlier+filter.

[53] *Contrast distortion artifact - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/502202755/Contrast+distortion+artifact.

[54] *Contrast Distortion filter - Zivid Knowledge Base - Confluence*. [Online]. Available: https://zivid.atlassian.net/wiki/spaces/ZividKB/pages/484081687/Contrast+Distortion+filter.

[55] *OpenCV: High Dynamic Range (HDR)*. [Online]. Available: https://docs.opencv.org/3.4/d2/df0/tutorial_py_hdr.html.

[56] *Understanding the importance of 3D hand-eye calibration*. [Online]. Available: https://blog.zivid.com/importance-of-3d-hand-eye-calibration.

[57] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient backprop," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700 LECTU, pp. 9–48, 2012, ISSN: 16113349. DOI: 10.1007/978-3-642-35289-8{_}3.

[58] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv*, vol. XX, no. Xx, pp. 1–22, 2019, ISSN: 23318422. DOI: 10.1109/tnnls.2020.2978386.

[59] L. Wu, F. Li, Y. Wu, and T. Zheng, "GGF: A graph-based method for programming language syntax error correction," *IEEE International Conference on Program Comprehension*, pp. 139–148, 2020. DOI: 10.1145/3387904.3389252.

[60] X. Wang, C. Chen, Y. Min, J. He, B. Yang, and Y. Zhang, "Efficient metropolitan traffic prediction based on graph recurrent neural network," *arXiv*, vol. 1, 2018, ISSN: 23318422.

[61] Z. Guo, C. Zhang, W. Yu, J. Herr, O. Wiest, M. Jiang, and N. V. Chawla, "Few-Shot Graph Learning for Molecular Property Prediction," 2021. DOI: 10.1145/3442381.3450112. [Online]. Available: http://arxiv.org/abs/2102.07916%0Ahttp://dx.doi.org/10.1145/3442381.3450112.

[62] F. Anowar, S. Sadaoui, and B. Selim, "Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE)," *Computer Science Review*, vol. 40, p. 100 378, 2021, ISSN: 15740137. DOI: 10.1016/j.cosrev.2021.100378. [Online]. Available: https://doi.org/10.1016/j.cosrev.2021.100378.

[63] L. V. U. Nl and G. Hinton, "Visualizing Data using t-SNE Laurens van der Maaten," *Journal of Machine Learning Research*, vol. 1, pp. 1–48, 2008, ISSN: 1532-4435. [Online]. Available: http://www.cs.toronto.edu/~hinton/absps/tsne.pdf.

[64] C. E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv*, pp. 1–20, 2018, ISSN: 23318422.

[65] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv*, pp. 1–13, 2017, ISSN: 23318422.

[66] S. Jadon, "A survey of loss functions for semantic segmentation," *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology, CIBCB 2020*, 2020. DOI: 10.1109/CIBCB48159.2020.9277638.

[67] J. Ma, "Segmentation Loss Odyssey," *arXiv*, 2020, ISSN: 23318422.

[68] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research*, vol. 9, pp. 249–256, 2010, ISSN: 15324435.

[69] W. Weng and X. Zhu, "INet: Convolutional Networks for Biomedical Image Segmentation," *IEEE Access*, vol. 9, no. June, pp. 16 591–16 603, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3053408.

[70] F. Calivá, C. Iriondo, A. M. Martinez, S. Majumdar, and V. Pedoia, "Distance map loss penalty term for semantic segmentation," *arXiv*, no. 1, pp. 1–5, 2019, ISSN: 23318422.

[71] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020, ISSN: 19393539. DOI: 10.1109/TPAMI.2018.2858826.

[72] F. Milletari, N. Navab, and S. A. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pp. 565–571, 2016. DOI: 10.1109/3DV.2016.79.

[73] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10072 LNCS, pp. 234–244, 2016, ISSN: 16113349. DOI: 10.1007/978-3-319-50835-1{_}22.

[74] L. Bottou, "Stochastic Gradient Learning," *Tutorial2*, 2012.

[75] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[76] P. Zhou, J. Feng, C. Ma, C. Xiong, H. O. Steven, and E. Weinan, "Towards theoretically understanding why SGD generalizes better than ADAM in deep learning," *arXiv*, no. 1, pp. 19–21, 2020, ISSN: 23318422.

[77] N. S. Keskar and R. Socher, "Improving generalization performance by switching from ADAM to SGD," *arXiv*, no. 1, 2017, ISSN: 23318422.

[78] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Journalism Practice*, vol. 10, no. 6, pp. 730–743, 2016, ISSN: 17512794. DOI: 10.1080/17512786.2015.1058180.

[79] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," pp. 248–255, 2009. DOI: 10.1109/cvprw.2009.5206848.

[80] *Imagenet classification leaderboard | computer-vision-leaderboard*, https://kobiso.github.io/Computer-Vision-Leaderboard/imagenet.html, (Accessed on 03/04/2021).

[81] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: Real-time 3D Object Detection on Point Clouds," pp. 1–14, 2018. [Online]. Available: http://arxiv.org/abs/1803.06199.

[82] J. Redmon, S. Divvali, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv*, 2016, ISSN: 23318422.

[83] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 386–397, 2018, ISSN: 19393539. DOI: 10.1109/TPAMI.2018.2844175.

[84] C. Szegedy, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *Pattern Recognition Letters*, vol. 42, no. 1, pp. 11–24, 2017, ISSN: 01678655. [Online]. Available: http://arxiv.org/abs/1512.00567.

[85] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Indian Journal of Chemistry - Section B Organic and Medicinal Chemistry*, vol. 45, no. 8, pp. 1951–1954, 2016, ISSN: 03764699. DOI: 10.1002/chin.200650130.

[86] C. Szegedy, "Going Deeper with Convolutions," *Journal of Chemical Technology and Biotechnology*, vol. 91, no. 8, pp. 2322–2330, 2015, ISSN: 10974660. DOI: 10.1002/jctb.4820.

[87] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 5100–5109, 2017, ISSN: 10495258.

[88] D. Griffiths and J. Boehm, "A Review on deep learning techniques for 3D sensed data classification," *Remote Sensing*, vol. 11, no. 12, 2019, ISSN: 20724292. DOI: 10.3390/rs11121499.

[89] W. Shi and R. R. Rajkumar, "Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud," *arXiv*, pp. 1711–1719, 2020, ISSN: 23318422.

[90] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[91] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," *arXiv*, no. ii, pp. 1–11, 2017, ISSN: 23318422.

[92] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 945–953, 2015, ISSN: 15505499. DOI: 10.1109/ICCV.2015.114.

[93] A. Boulch, J. Guerry, B. Le Saux, and N. Audebert, "SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks," *Computers and Graphics (Pergamon)*, vol. 71, pp. 189–198, 2018, ISSN: 00978493. DOI: 10.1016/j.cag.2017.11.010. [Online]. Available: https://doi.org/10.1016/j.cag.2017.11.010.

[94] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2016, ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2644615.

[95] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *IEEE Access*, vol. 9, pp. 16 591–16 603, 2015, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3053408.

[96] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition Daniel," *The Positive Encourager*, 2015, ISSN: 21530866. [Online]. Available: http://www.thepositiveencourager.global/the-mentoring-approach/.

[97] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–14, 2017.

[98] Y. U. E. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds," vol. 1, no. 1, 2019.

[99] C. Zhang, C. Liu, X. Zhang, and G. Almpanidis, "An up-to-date comparison of state-of-the-art classification algorithms," *Expert Systems with Applications*, vol. 82, pp. 128–150, 2017, ISSN: 09574174. DOI: 10.1016/j.eswa.2017.04.003. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2017.04.003.

[100] G. Mountrakis, J. Im, and C. Ogole, "Support vector machines in remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 3, pp. 247–259, 2010, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2010.11.001. [Online]. Available: http://dx.doi.org/10.1016/j.isprsjprs.2010.11.001.

[101] R. E. Schapire, "The Strength of Weak Learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990, ISSN: 15730565. DOI: 10.1023/A:1022648800760.

[102] ——, "A Brief Introduction to Boosting Generalization error," *Ijcai 99*, pp. 1401–1406, 1999, ISSN: 1045-0823. [Online]. Available: http://u.math.biu.ac.il/~louzouy/courses/seminar/boost1.pdf.

[103] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015, ISSN: 2198-5804. DOI: 10.1007/s40745-015-0040-1.

[104] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005, ISSN: 10459227. DOI: 10.1109/TNN.2005.845141.

[105] J. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14, pp. 281–297, 1967. [Online]. Available: http://books.google.de/books?hl=de&lr=&id=IC4Ku_7dBFUC&oi=fnd&pg=PA281&dq=MacQueen+some+methods+for+classification&ots=nNTcK1IdoQ&sig=fHzdVcbvmYJ-lTNHu1HncmOFOkM#v=onepage&q=MacQueen%20some%20methods%20for%20classification&f=false.

[106] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," *SIGMOD Record (ACM Special Interest Group on Management of Data)*, vol. 25, no. 2, pp. 103–114, 1996, ISSN: 01635808. DOI: 10.1145/235968.233324.

[107] M. Ester, H.-P. Kriegel, J. Sander, and X. Xiaowei, "A density-Based Algorithm for Discovering Clusters," *Comprehensive Chemometrics*, vol. 2, pp. 635–654, 1996. DOI: 10.1016/B978-044452701-1.00067-3.

[108] M. Ankerst, M. M. Breunig, H.-p. Kriegel, and J. Sander, "OPTICS: Ordering Points To Identify the Clustering Structure," *ACM SIGMOD Record*, vol. 28, no. 2, pp. 49–60, 1999.

[109] D. Barbará and P. Chen, "Using the fractal dimension to cluster datasets," *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, no. August, pp. 260–264, 2000. DOI: 10.1145/347090.347145.

[110] H. Brådland, "Implementation of model-free grasping of arbitrary shaped objects," 2020.

[111] ABB, "IRB 4400 Product Description," 2019.

[112] ——, *IRBT 4004 / 6004 / 7004 Data S*, 2016.

[113] ——, *Absolute Accuracy: Industrial Robot Option*, 2010. [Online]. Available: https://library.e.abb.com/public/931fcb281dbe7fecc1257b1300579a6a/AbsAccPR10072EN_R5.pdf.

[114] H. Automotive and I. Laboratory, *Semantic-segmentation-editor: Web labeling tool for bitmap images and point clouds*, https://github.com/Hitachi-Automotive-And-Industry-Lab/semantic-segmentation-editor, (Accessed on 04/13/2021), 2020.

[115] B. de Brabandere, D. Neven, and L. van Gool, "Semantic Instance Segmentation with a Discriminative Loss Function," *arXiv*, 2017, ISSN: 23318422.

[116] S. K. Lodha, D. M. Fitzpatrick, and D. P. Helmbold, "Aerial lidar data classification using AdaBoost," *3DIM 2007 - Proceedings 6th International Conference on 3-D Digital Imaging and Modeling*, pp. 435–442, 2007. DOI: 10.1109/3DIM.2007.10.

[117] Z. Wang, L. Zhang, T. Fang, P. T. Mathiopoulos, X. Tong, H. Qu, Z. Xiao, F. Li, and D. Chen, "A multiscale and hierarchical feature extraction method for terrestrial laser scanning point cloud classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 5, pp. 2409–2425, 2015, ISSN: 01962892. DOI: 10.1109/TGRS.2014.2359951.

[118] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," *Computers in Education Journal*, vol. 6, no. 3, pp. 46–48, 2018, ISSN: 10693769.

[119] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 3147–3155, 2017, ISSN: 10495258.

[120] L. Tchapmi, C. Choy, I. Armeni, J. Young Gwak, and S. Savares, "SEGCloud: Semantic Segmentation of 3D Point Clouds,"

[121] 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," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 11 105–11 114, 2020, ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.01112.

[122] J. Friedman, "Greedy Function Approximation : A Gradient Boosting Machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[123] G. Storey, L. Jiang, and Q. Meng, "A Point Cloud Semantic Segmentation Framework for Embedded Systems in Agricultural Robots," no. April, pp. 92–94, 2020. DOI: 10.31256/bk3wa5k.

[124] W. Liu, J. Sun, W. Li, T. Hu, and P. Wang, "Deep learning on point clouds and its application: A survey," *Sensors (Switzerland)*, vol. 19, no. 19, pp. 1–22, 2019, ISSN: 14248220. DOI: 10.3390/s19194188.

[125] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017, ISSN: 10535888. DOI: 10.1109/MSP.2017.2693418.

[126] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 2131–2136, 2010. DOI: 10.1109/IROS.2010.5650459.

[127] G. Te, "RGCNN : Regularized Graph CNN for Point Cloud Segmentation,"

[128] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *arXiv*, pp. 1–22, 2018, ISSN: 23318422.

[129] V. Shubham, S. Vishwas, and K. Anupama, "Analysis of Sorting Algorithms Using Time Complexity," vol. 5, no. 21, pp. 5–7, 2017.

[130] scikit-learn developers, *Comparing different clustering algorithms on toy datasets scikit-learn 0.24.2 documentation*, https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py, (Accessed on 05/11/2021).

[131] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017. arXiv: 1612.00593 [cs.CV].

[132] J. Wang and L. Perez, "The effectiveness of data augmentation in image classification using deep learning," *arXiv*, 2017, ISSN: 23318422.

[133] R. Li, X. Li, P. A. Heng, and C. W. Fu, "PointAugment: An Auto-Augmentation Framework for Point Cloud Classification," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 6377–6386, 2020, ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.00641.

[134] D. Griffiths and J. Boehm, "Weighted point cloud augmentation for neural network training data class-imbalance," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, vol. 42, no. 2/W13, pp. 981–987, 2019, ISSN: 16821750. DOI: 10.5194/isprs-archives-XLII-2-W13-981-2019.

[135] Y. Chen, V. T. Hu, E. Gavves, T. Mensink, P. Mettes, P. Yang, and C. G. Snoek, "PointMixup: Augmentation for Point Clouds," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12348 LNCS, pp. 330–345, 2020, ISSN: 16113349. DOI: 10.1007/978-3-030-58580-8{_}20.

[136] C. Bowles, L. Chen, R. Guerrero, P. Bentley, R. Gunn, A. Hammers, D. A. Dickie, M. V. Hernández, J. Wardlaw, and D. Rueckert, "GAN augmentation: Augmenting training data using generative adversarial networks," *arXiv*, 2018, ISSN: 23318422.

[137] A. R. Bhople, A. M. Shrivastava, and S. Prakash, "Point cloud based deep convolutional neural network for 3D face recognition," *Multimedia Tools and Applications*, 2020, ISSN: 15737721. DOI: 10.1007/s11042-020-09008-z.

[138] M. F. Silva, J. LuÃŋs Lima, L. P. Reis, A. Sanfeliu, and D. Tardioli, "Perception of Entangled Tubes for Automated Bin," *Advances in Intelligent Systems and Computing*, vol. 1092 AISC, no. February 2020, p. C1, 2020, ISSN: 21945365. DOI: 10.1007/978-3-030-35990-4.