

# DEVELOPMENT OF COMPUTER VISION SYSTEM TO DETECT AND IDENTIFY FLEXIBLE STRUCTURES

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

PHILIP LEIRFALL

## SUPERVISOR

Ilya Tyapin

Muhammad Talha Bilal

**University of Agder, 2022**

Faculty of Engineering and Science

Department of Engineering and Sciences

# Acknowledgements

I would like to thank the supervisors Ilya Tyapin and Muhammed Bilal for their guidance and support throughout the master thesis. I would also like to thank Karl Berge Rød and the staff in UiA's mechatronical lab for helping with equipment.

I would also like to thank co-students, family members and my significant other for supporting me through the master thesis.

# Abstract

With increase of Electrical Vehicles, the production of Lithium-ion batteries will be in large demand. The Lithium-Ion Battery Recycling (LIBRES) research project aims to recycle the batteries, to re-use material from batteries at end-of-life cycle. This thesis will solve one of LIBRES tasks of development of a fully automated system, by detecting flexible structures with computer vision. The aim of the system is to test different segmentation algorithms, evaluate and create pose estimation. The system only managed to test one algorithm in Matlab, with three different algorithms failing to train.

# Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Research question . . . . .	2
1.2 Project outline . . . . .	3
<b>2 Background theory</b>	<b>4</b>
2.1 Robotic Operation System . . . . .	4
2.2 Matlab . . . . .	5
2.3 Sensors . . . . .	6
2.3.1 3D data . . . . .	6
2.3.2 Structured light sensors . . . . .	7
2.3.3 Camera Configuration . . . . .	8
2.3.4 Eye-in-hand system . . . . .	12
2.4 Image recognition . . . . .	13
2.4.1 Neural Network . . . . .	13
2.4.2 Object Detection . . . . .	14
2.4.3 NVIDIA/semantic-segmentation . . . . .	16
2.5 Pose estimation . . . . .	18
<b>3 Method</b>	<b>20</b>
3.1 Hardware . . . . .	20
3.1.1 Zivid camera . . . . .	20
3.1.2 ABB robot . . . . .	20
3.2 ROS Setup . . . . .	22
3.3 Object Detection . . . . .	24
3.3.1 Image collection and labeling . . . . .	24
3.3.2 Training . . . . .	27
3.4 Pixel to world coordinates . . . . .	34
<b>4 Results &amp; Discussion</b>	<b>36</b>
4.1 Results . . . . .	36
4.1.1 Algorithms & neural networks . . . . .	36
4.2 Discussion . . . . .	39
<b>5 Conclusions</b>	<b>41</b>

<b>Bibliography</b>	<b>42</b>
<b>A Datasheet A</b>	<b>44</b>
A.1 detectron2Training.py . . . . .	44
A.2 labelme2coco.py . . . . .	46
<b>B Datasheet B</b>	<b>49</b>
<b>C Datasheet C</b>	<b>54</b>
C.1 Train semantic segmentation Network . . . . .	54
C.2 Change image size . . . . .	58
<b>D Datasheet D</b>	<b>60</b>

# List of Figures

2.1	ROS structure [12]	5
2.2	Geometry of structured light setup [15]	7
2.3	Zivid One+ industrial 3D camera [16]	8
2.4	Difference in stops [17]	8
2.5	The effect gain has on the signal [20]	9
2.6	The effect gain has on the signal [22]	10
2.7	Eye in hand [28]	12
2.8	Simplification of neural network [30]	13
2.9	Visual of object detection [31]	14
2.10	Difference between Semantic and Instance Segmentation [32]	15
2.11	Mask R-CNN Framework for Instance Segmentation [32]	16
2.12	NVIDIA Network Architecture [33]	16
2.13	NVIDIA predictions at 0.5x & 2.0x scale [33]	17
2.14	Architecture of DEXTR [34]	17
2.15	Pinhole camera model [35]	18
3.1	Zivid One + camera [16]	20
3.2	IRB 4400 Robot [37]	21
3.3	IRB 4400/60 dimensions and range [37]	21
3.4	Segmentation VW LIB [5]	24
3.5	ImageLabeler	25
3.6	ImageLabeler export	26
3.7	Detectron2 Training script structure [5]	27
3.8	Deep Network Designer App	31
3.9	Upload network to Deep Network Designer App	32
3.10	Import training & validation data to App	32
3.11	Training options	33
3.12	Pose estimation pixel to world coordinates script [5]	34
4.1	Detectron2 error code	36
4.2	DEXTR-PyTorch	37
4.3	Training of Matlab Network	37
4.4	Semantic segmentation mask	38

# List of Tables

2.1	f-number to Stops . . . . .	9
3.1	Class labels . . . . .	24
3.2	Class labels . . . . .	25
3.3	Semantic segmentation network layer input . . . . .	31
3.4	Network training options . . . . .	31
4.1	Global- and Mean Accuracy, and Mean- and Weighted IoU . . . . .	37
4.2	Normalized Confusion Matrix (%) . . . . .	38
4.3	Class accuracy and IoU . . . . .	38

# Chapter 1

## Introduction

In 2019 during March, thousands of Tesla Model 3 were bought in Norway, setting a record high sale of electric vehicles (EV) [1]. As of 2020, the sale has declined, but the popularity of EV in Norway still stands strong. 80 percent of all new cars sold in Norway are either electrical, chargeable hybrids or hybrids as of 2020, with only 20 percent being cars with combustion engine.

The steady increase of EV is probably due to newer models coming out cheaper and improvement in range and performance. For many an EV is environment friendly and emission-free, but EV are far from emission-free. The resources for production and the production itself causes environmental challenges, and focus on recycle as much as possible of EV at its end of life cycle should not be overlooked.

### **LIBRES**

LIBRES research project focus on the recycling of the lithium ion battery. Partnered up are Batteriretur, Hydro, Glencore Nikkelverk and Keliber OY [2]. Together they are working with R&D partners Elkem Technology, IME RWTH Aachen, MIMI Tech, Agder University and NTNU to develop and commercialize a new lithium battery recycling process that covers the need for the anticipated volumes of batteries in the future. Key aspect of the project is to improve the material recovery rate than present day.

Norway has the highest share of EV as proportion of new cars in the world, and by 2025 the aim is that all new cars sold in Norway will be free of emission. In EU, the sale of EV and plug-in hybrids have increased tremendously. From 2020 to 2021, the sales doubled with 6750 sales, where 29 percent was plug-in hybrid [3]. The increase of EV in Europe will create a need to recycle end-of-life batteries.

### **Reason for automation**

Currently, the process of dismantling the batteries is done manually in Norway. The work is done by two authorised electricians with high voltage experience, where dismantling one battery pack takes about 45 min [4]. This process is time consuming and expensive. With the ever growing EV market, training staff to meet the demand will be expensive, and not viable for mass industrial recycling. A possible solution could be to make autonomous robots do the dismantling. An autonomous dismantling plant acting like a reverse assembly line would require less specialized workers. It would also reduce the danger of working with battery-packs, such as voltage, lithium or gas poisoning etc.



## Challenges

The task of disassembling the battery packs consists of many steps. Removing screws, zip ties, cables etc. The biggest obstacle is that batteries have different design from model to model, and new models and technology will make it difficult for the robot to have pre-defined disassembly process of all battery packs. As well as the possibility of damages or changes in packs that would make the pack unrecognizable in terms of the machine vision. This would be a difficult problem to solve, so instead of focusing on the entire packs, a solution is to shift the focus to the components of the pack. To solve this, the robot will need to learn to detect different components, such as wires, screws and bolts, and remove them in a safe manner. If an autonomous robot can do the dismantling, it will reduce the need for skilled electrician, save money, and reduce the risk of injury.

## Important criteria for plant

For LIBRES autonomous disassembly plant to be a viable solution, the plant needs to be able to operate with a large scope of different battery packs. This project have set some key criteria for the disassembly plant.

- **Model-free approach:** Because there are too many different packs, the system cannot depend on having access to 3D models of the battery packs. It would require constant updating the 3D models as new car models were announced, as well as adjusting to recognise either damaged or altered battery packs.
- **Part segmentation system :** With sensors scanning the packs, the system will need algorithms capable of identifying objects of interest.
- **Robotic manipulator :** To dismantle the entire battery pack will require a large set of complex manipulation tasks. A few robots could be versatile enough to complete these tasks. They would need to have a set of interchangeable tools which would allow for the different tasks required.

## 1.1 Research question

This thesis will look into what algorithm is best suited for segmentation of cables. The project will then focus these tasks:

- Collect data
- Find suitable algorithms & network for segmentation
- Train algorithm and network
- Evaluate algorithms
- Create pose estimation

If the project is a success, it will look into optional task such as:

- Design cable gripper tool
- Implement algorithm and network with MoveIt
- Perform cable removal with ABB robot

## 1.2 Project outline

This report is structured as the following:

Chapter 2 in the report is going over the theoretical parts and background information. The theoretical on the algorithms and network chosen, 3D sensors, as well as explaining camera setting for a Zivid camera and a solution for sensor placement.

Chapter 3 contains the methods used for this project, and the hardware used. The physical work and tests are explained here, as to show the steps done in the project. The project will only focus on detection of cables, as other projects have already completed detection and removal of screws and bolts.

Chapter 4 go over and discuss the results of the project, as well as problems during the project.

The last chapter contains the conclusion of the thesis.

# Chapter 2

## Background theory

This chapter contains the theory that lays the foundation for the work done in this thesis.

### 2.1 Robotic Operation System

This thesis uses Robotic Operating System (ROS), as it has been used by previous students working on the LIBRES project [5]. ROS is an open source operating system that allows for different components to operate together seamlessly [6]. It provides libraries and tools to build all types of robotic applications. The project server is running on ROS kinetic, so using ROS to communicate with the manipulator arm is a requirement. Machines are usually set up of actuators, sensors and code/algorithms to tell it what to do. For all of it to work together, ROS is build up to allow all these things to communicate with each other. ROS is build up with nodes that publish or subscribe data to a topic. This way the nodes do not need to know of each other, only the topics. ROS uses these different building blocks:

1. Nodes. A node in ROS is a process that performs computation [7]. For instance, one node controls an actuator, while another controls a sensor. For the actuator to know how much it needs to move, the sensor node sends data to a topic where the actuator node subscribes.
2. Topics. ROS uses topics to allow nodes to exchange messages. They are named buses and uses anonymous publish/subscribe semantics [8]. Topics work like a place of data for nodes to collect and deliver data. Topics can have multiple subscribers and publisher, where non of the nodes know of each other.
3. Message. When nodes are subscribing and publishing they are using a simple data structure that ROS communicate with [9]. This is called a message (.msg). The message files consist of two parts: field and constants. The fields are the data inside the message, and can be specified from a set of basic types, like integer, floating-point or boolean. Constant defines useful values that can be used to interpret those fields. The messages are what is being broadcasted on a topic.
4. Services. Similar to topics, nodes can subscribe and publish data (publish/subscribe system), while the service uses Request/reply system [10]. A service is provided under a name and the system sends a request to that name and waits for a reply.
5. Master. At the core of ROS, the ROS Master takes care of the naming and registration services to the nodes in the system [11]. It helps nodes locate one another, and tracks publishers and subscribers to topics and services. Figure 2.1 shows how the master works.

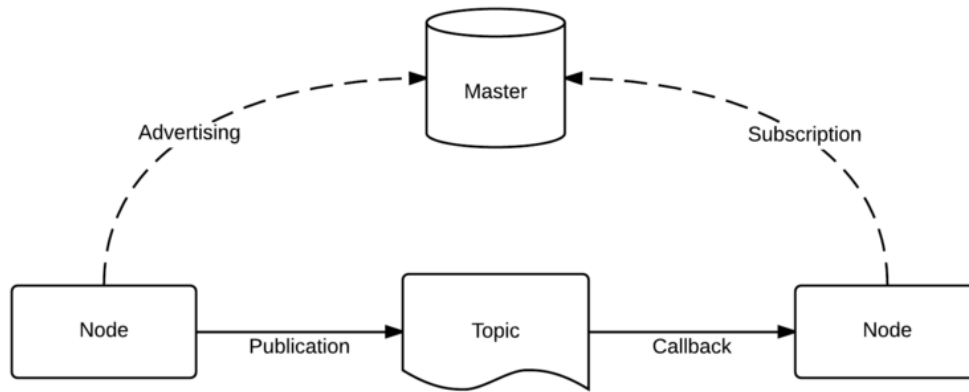


Figure 2.1: ROS structure [12]

## 2.2 Matlab

This project uses Matlab. It is a programming platform used to analyze data, design systems, develop algorithms, and create models [13]. Matlab can be downloaded with an active license. Matlab also comes with different toolboxes to fit your project. Everything from simulating hydraulics to machine learning. The toolboxes are professionally developed, tested and documented, allowing for easy use and application. For this project the key toolbox are the Image Processing Toolbox, Computer Vision Toolbox and Deep Learning Toolbox.

### Image Processing Toolbox

This toolbox provides algorithms for image processing, visualization and algorithm development. It also let the user perform image segmentation, image enhancement and noise reduction. It supports processing of 2D and 3D images, as well as arbitrarily large images.

### Computer Vision Toolbox

The toolbox provides algorithms, functions and apps that allows you to designing and test computer vision. The toolbox allows for object detection, tracking, feature detection, extracting and matching. Using deep learning algorithms such as U-Net and Mask R-CNN allows for semantic and instance segmentation. The toolbox allows for running algorithms on multicore processors and GPUs to speed up algorithms, as well as it supports C/C++ code generation for integrating with existing code.

### Deep Learning Toolbox

Deep learning toolbox allows Matlab to design and implement deep neural network with algorithms, and provides pretrained models. The toolbox allows the user to build network architectures, analyse and train network graphically. The networks created can be exchanged with TensorFlow and PyTorch through the ONNX format.

## 2.3 Sensors

Getting rough data on where the cables are located in the battery pack is done by sensors. It is then important that reliable sensors are placed so that they are in position to capture the packs from different angles, as many of the cables are either under each other, or under panels or cells. To locate all cables, both good choice of sensors and placement is paramount. This section looks into the different choices for sensors to capture 3D data, as well as solutions for placement.

### 2.3.1 3D data

There are several ways to obtain 3D data. The most common point clouds are [14]:

- Light Detection and Ranging (Lidar) - Uses light in form of pulsed laser to measure ranges.
- Red Green Blue - Depth (RGB-D) - A depth-sensing device in association with red, green and blue (RGB). The sensors adds depth information per pixel to images.
- Synthetic Aperture Radar (SAR) - Sends out microwave signals to measure range.

SAR is mostly useful for building and landscapes reconstruction and will not be suited for for much smaller parts. Object tracking is one of RGB-D's three main application, and given that it is cheaper than Lidar and can generate the 3D position of each pixel makes it the best option for this thesis. RGB-D sensor uses three different ways to detect depth: Structured light, stereo and time of flight.

### 2.3.2 Structured light sensors

This project uses a sensor which uses the structured light.

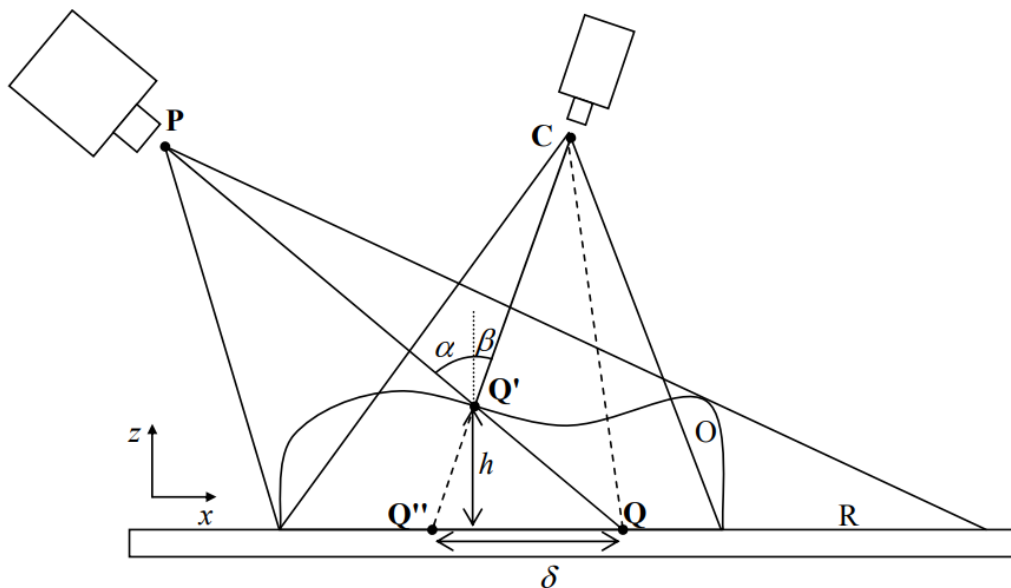


Figure 2.2: Geometry of structured light setup [15]

As seen in figure 2.2, **P** is the exit pupil of the projector, while **C** is the entrance pupil of the camera. **R** is a plane reference, that defines the zero-level height. From the projector (**P**) a beam hits point **Q** on the reference plain. Introducing object **O**, the line from **P** to **Q** will intercept the surface of object at point **Q'**. This point is captured on the camera (**P**) as a displacement,  $\delta = |\mathbf{Q}''\mathbf{Q}|$  [15]. The relation between displacement  $\delta$  and height  $h$  is:

$$h = \frac{\delta}{\tan \alpha + \tan \beta} \quad (2.1)$$

## 2.3.3 Camera Configuration

### Capture settings

To optimize the data collected, the sensor used have capture settings to allow for different environment (light/dark etc.). In this project, a Zivid camera (shown in figure 2.3) is provided, and will be discussed further in section 3.1.1.



Figure 2.3: Zivid One+ industrial 3D camera [16]

For a Zivid camera the settings are:

### Stops

Stops or Exposure stops are used to describe how much light hits the sensor, or the brightness of a image related to a reference level. This being referred to as "0 stops". Going up to 1 stop will double the brightness, and going down to -1 stop will reduce the brightness by half. Figure 2.4 show the factor and intensity with stops from minus two to plus two.

The Zivid camera has 23 stops available, which allows it to get decent data, even from shiny objects/surfaces [17].

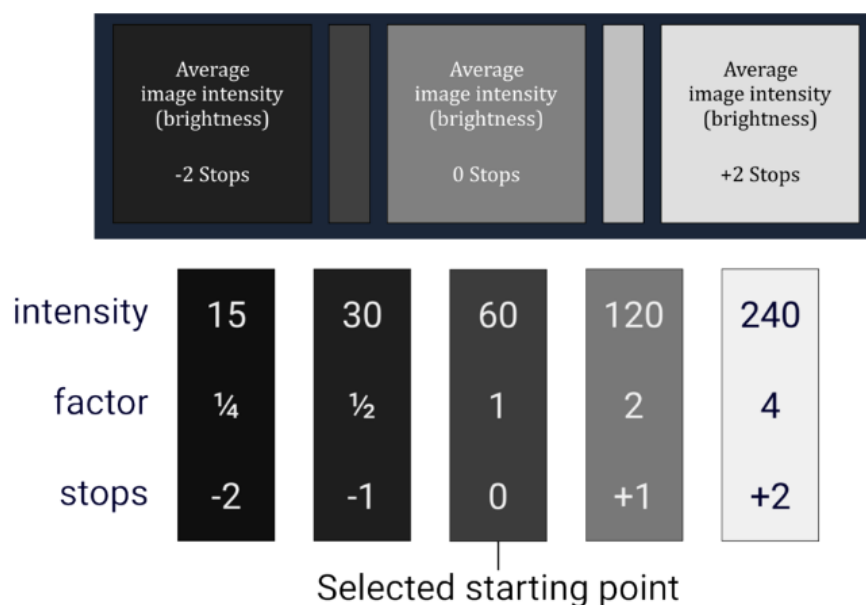


Figure 2.4: Difference in stops [17]

## Exposure time

Exposure time is when the camera lets in light to capture an image. To regulate the time, the camera is equipped with shutters. To get the best exposure time, it is important to look at the ambient light source. In the EU this is 50 Hz [18]. The best exposure time is calculated as follows:

$$t_{exp} = \frac{n}{2f_s}, n = |Z| \quad (2.2)$$

where  $n$  is a positive integer, and  $f$  is the light source frequency. As the EU is using 50 Hz, it is recommended to use multiple of 10 000  $\mu$ s. It is possible to use higher, such as 20 000  $\mu$ s, but it requires stops.

## Aperture

Aperture, or iris is the opening of the lens. The size of the opening can be described with an equation

$$N = \frac{f}{D} \quad (2.3)$$

where  $f$  is the focal length and  $D$  is the diameter of the entrance pupil [19].

Modern lenses mostly use a standard f-number with a geometric sequence of number that correspond to the power of the square root of 2.

$$\frac{f}{\sqrt{2^n}} \quad (2.4)$$

where  $n$  goes from 0 and up. On a zivid camera there is a integrated electro-mechanical iris that can be quickly adjusted. If using the stop methode, the following f-number values should be used as shown in table 2.1.

f-numbers	$\frac{f}{1.4}$	$\frac{f}{2}$	$\frac{f}{2.8}$	$\frac{f}{4}$	$\frac{f}{5.6}$	$\frac{f}{8}$	$\frac{f}{11}$	$\frac{f}{16}$	$\frac{f}{22}$	$\frac{f}{32}$
Stops	+4	+3	+2	+1	0	-1	-2	-3	-4	-5

Table 2.1: f-number to Stops

The different sizes of aperture with corresponding f-numbers can be seen in figure 2.5



Figure 2.5: The effect gain has on the signal [20]



## Brightness

The projector brightness controls the amount of light emitted from the projector. The projector brightness is the most effective way to maximize signal-to-noise ratio (SNR). [21]. Having a high setting on brightness allows for large amplitude of the received data by the camera, thus minimizing the effect of the noise. The only concern is that the projector does not over-saturate pixels.

## Gain

The gain increase the dynamic range when using a high dynamic range (HDR). It allows for minimal time penalty, as well as getting point cloud in very dark regions. The drawback is everything gets amplified, including the noise [22]. This reduces the signal-to-noise ratio, so it is recommended to keep the gain low to get optimal 3D point cloud reading. In figure 2.6 the effect of gain on the signal is shown. The dynamic range is increased and the camera is able to distinguish objects in low light areas.

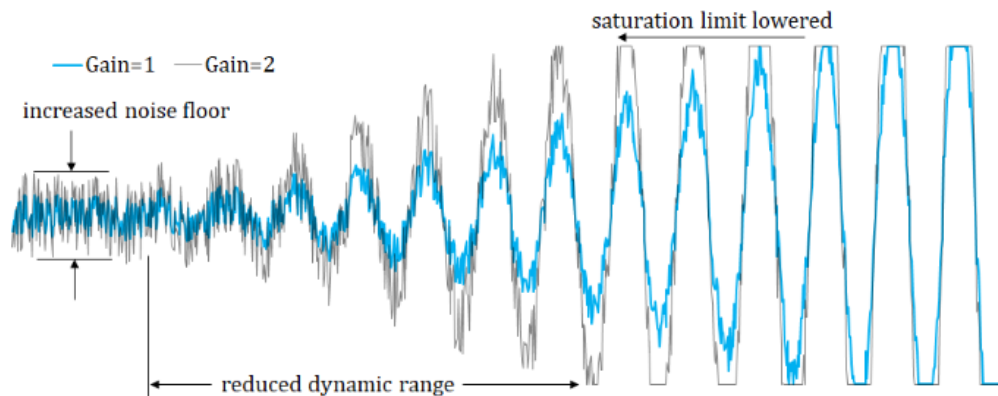


Figure 2.6: The effect gain has on the signal [22]

## Filters

- **Contrast** filter corrects and/or removes points effected by contrast distortion. In point cloud, areas with high specular reflection and in regions with large texture gradients can cause blur in the image [23]. The filter has two modes to deal with this problem. Correction and removal. If large correction is above the threshold parameter the points are removed instead. It works best when aperture value chosen gives the camera a good focus.
- **Gaussian** The gaussian filter smoothe the pixel within a small local region, which allows for suppressing noise and align pixel to a grid [24]. Because the filter preforms smoothing, it can improve the absolute noise and performance, and be useful for some matching algorithms. The reduced noise helps making parts more similar as a whole, which come in handy when using object detection.
- **Outlier** Depended on neighboring pixels, the pixel becomes a outlier if it is too far away from its closest neighbor and removed [25]. It is used to remove potential noise and stray points.
- **Reflection** The filter removes points impacted by reflections. It is usually seen as a "ghost planes", where regions of unwanted points are stretching towards or away from the camera. Since the Zivid camera knows the signals sent out, it can try to make sense of the data makes sense or not [26]. It then removes pixels effected by either interreflections, excessive movements in the scene, or alternating alien light sources.
- **Noise Filter** This filter removes points under a specified limit set by the SNR filter [27]. The filter limit should not be set too high to remove too much of the data, as a little noise is better than loss of data.

### 2.3.4 Eye-in-hand system

The eye-in-hand system is what it sounds like, an eye in your hand. It is a range sensor, usually a camera, mounted at the tip of a manipulator arm. This allows for the camera to be moved around the workspace. This gives the camera view from different positions, as well as to get a better picture of where the workspace is according to the manipulator. It also allows for higher resolution of what the manipulator is looking for, as it can move closer to what is manipulating. With the eye-in-hand system multiple pictures with higher resolution can be taken, while also getting a picture of the entire workspace. While several stationary cameras would be needed to get pictures from multiple angles, an eye-in-hand would require one single camera.

Many of the battery packs have multiple cables that are overlapping and in different sections of the battery pack.

One single picture is not enough to get all the cables, and to avoid missing some cables, several pictures from different positions are needed. Then the eye-in-hand system is effective, as it can cable ties etc. With an algorithm, the eye-in-hand can be used to look over the entire battery-pack and log every region. Figure 2.7 shows how two transformation matrices,  $H_{EE}^{ROB}$  and  $H_{CAM}^{EE}$  are used to get the sensor data in reference to the base of the robot.  $H_{EE}^{ROB}$  transform from the robot base to the end-effector (the robot tool point) [28], while the  $H_{CAM}^{EE}$  is the transformation from the end-effector to the camera.

[28]

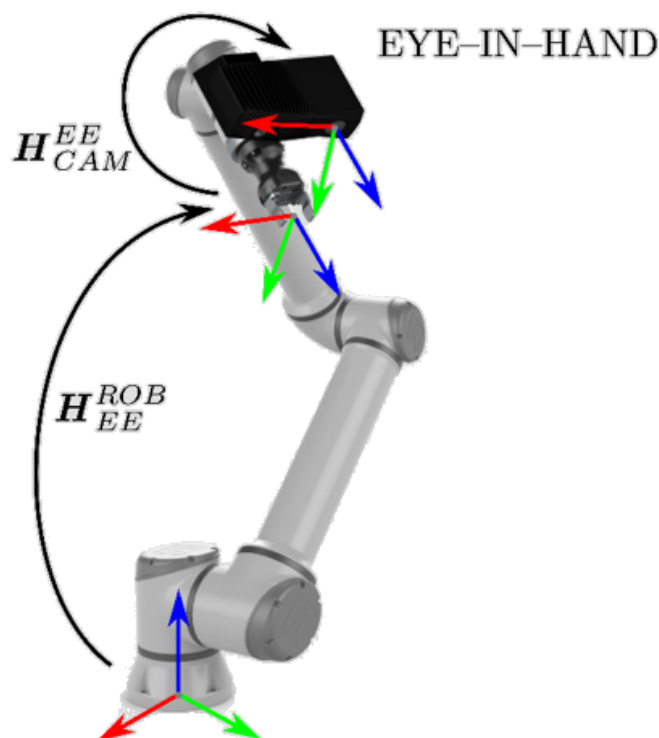


Figure 2.7: Eye in hand [28]

## 2.4 Image recognition

Machine learning is essentially getting a machine to think for itself. Like teaching a child to walk and talk, machine learning focuses on using data and algorithms to either look for patterns or predict outcomes from gathered data. It can be used to give data to an algorithm to look for certain objects. Teaching the machine with data can be done in three different ways [29].

- Supervised machine learning is the most common form of machine learning, where the algorithm is fed large amount of labeled training data till the model is fitted appropriately. It then make prediction on data it never have seen before based on correlation it got from the labeled data.
- Unsupervised machine learning are algorithms that analyse and cluster unlabeled data. They discover hidden patterns or data grouping without help from humans.
- Semi-supervised learning is the mix between the two. To train it it uses smaller labeled data set to guide classification, and feature extraction from a larger, unlabeled data set.

### 2.4.1 Neural Network

A neural network is a computational learning system that uses a network of functions to understand and translate data input [30]. Inspired from the way neurons in the human brain work, the network takes labeled data and process them to be able to recognize the pattern. To give an example, recognizing a person require certain characteristics to match. Black hair, tall, moustache and glasses create a model for the person to be recognized. The same model is created in the neural network, where the network evaluates the data points of the image towards the model. Based on how close the match is, the person is recognized, or the person is a stranger who matches only a few of the criteria in the model.

Figure 2.8 shows a simplified version of how input data in a neural network is checked with the model. Out of the model the output is either a match or not.

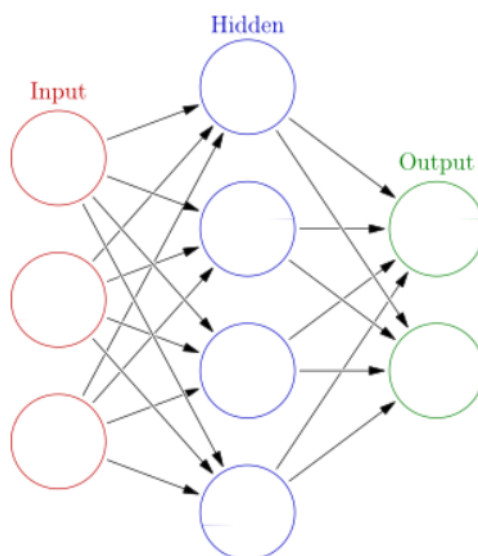


Figure 2.8: Simplification of neural network [30]

## 2.4.2 Object Detection

For the machines to detach the cables, it needs to locate and recognize cables. To do this object detection is used. When the camera capture where the cables are, it can give the information to the rest of the system, which then plan on the next step to disconnect the cables from the battery packs.

### What is object detection and how does it work?

Humans can determine a cow from other animals by learning how a cow looks like. The same principle is applied to object detection. It is a computer vision technique that works to identify and locate objects within an image or video [31]. The main objective is to point out what is on an image or video. If there are pictures of several dogs in a photo, the object detection places a box over all the dogs and label the boxes for "dog". As seen in figure 2.9 where two cats, a person and a dog are boxed with correct labels.

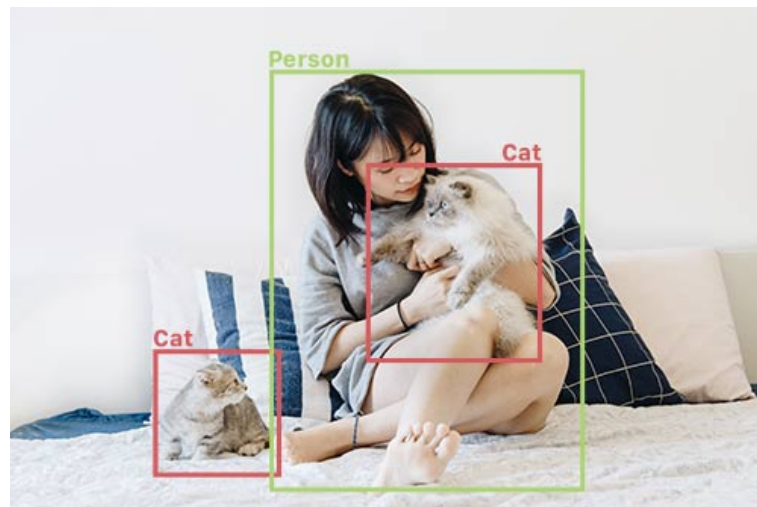


Figure 2.9: Visual of object detection [31]

By using machine learning or deep learning, algorithms recognizes patterns or shapes and can learn to locate and identify objects of interest.

This project looks into four different object detection methods using segmentation.

- Detectron2
- NVIDIA/semantic-segmentation
- DEXTR-PyTorch
- Matlab semantic segmentation

## Network

Network is used to set class of objects that are present in an image, that is marked by a square around said objects. The classes is what the machine learning is looking for, whether it is humans, cars or dogs. This is where the machine learning is coming in, where the the machine learning is fed shapes with labels. The network chosen in this thesis is Mask R-CNN, as it is continuity of a bachelor thesis from LIBRES.

Mask R-CNN is a deep convolution neural network and state-of-art in terms of image segmentation and instance segmentation [32]. The network can be used for instant or semantic segmentation. This thesis focuses on instant segmentation.

## Segmentation

Image segmentation is gathering part of an image together that belong to the same object, or object class as it is called. This means finding each pixel of an object in an image and outlining it.

This project looks into segmentation, where there are two types; semantic and instance. The difference between instant and semantic segmentation is that while semantic uses all pixels within a class as one entity, the instance segmentation splits them into different entities as shown in figure 2.10.

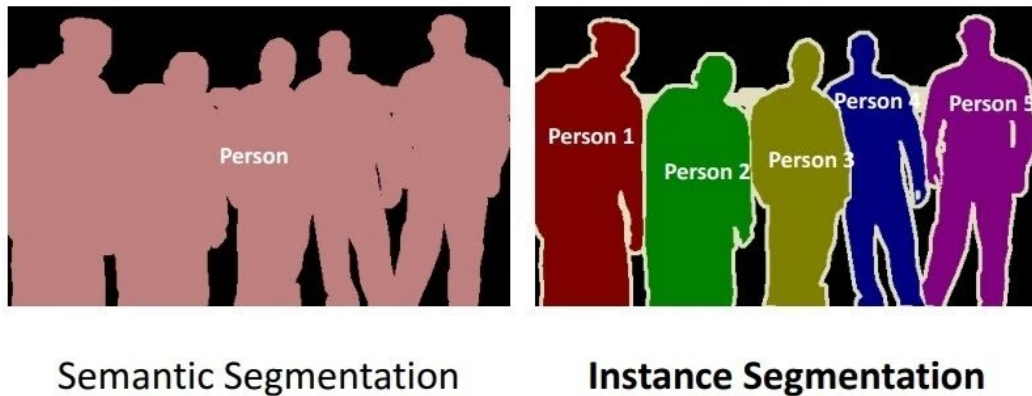


Figure 2.10: Difference between Semantic and Instance Segmentation [32]

## Mask R-CNN

The Mask R-CNN was build from Faster R-CNN which uses two stages: Regional Proposal Network (RPN) and Region of Interest Pooling (RoIPool). In Mask C-RNN, the first stage is RPN, which is a Neural Network that propose multiple objects that are available within a particular image. The next stage is predicting the class of the object withing the regions and box offset. While class and box offset are set, the Mask R-CNN also outputs a binary mask for each Region of Interest (ROI). The framework for instance segmentation is shown in figure 2.11.

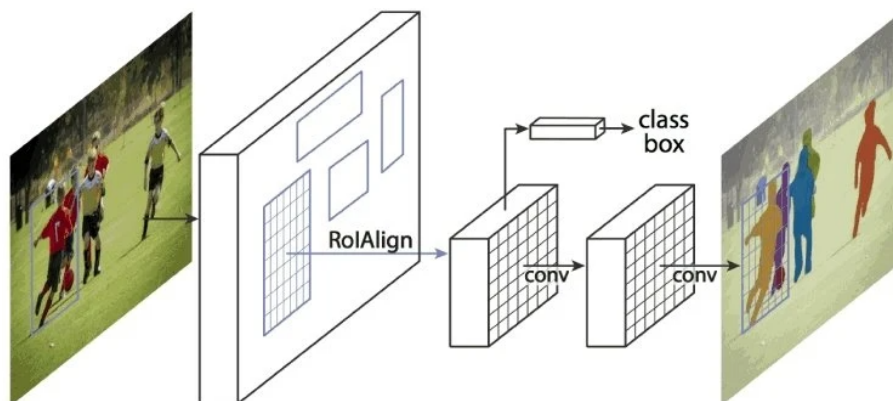


Figure 2.11: Mask R-CNN Framework for Instance Segmentation [32]

## Detectron2

Detectron2 is a segmentation algorithm from Facebook AI Research (FAIR). Detectron2 can be used with Mask R-CNN, and is implemented in PyTorch.

### 2.4.3 NVIDIA/semantic-segmentation

To handle difficulty with both high and low inference resolution, NVIDIA uses multi-scale inference. The network learns to predict a relative weighting between adjacent scales, and only require to augment the training pipeline [33]. As shown in figure 2.12, the algorithms works with predicting attention between adjacent scale pairs.

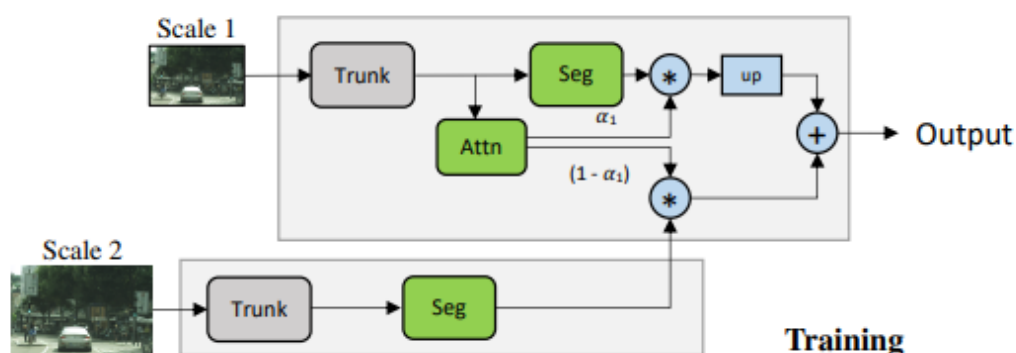


Figure 2.12: NVIDIA Network Architecture [33]

Because of this method, the algorithm can segment smaller objects at the larger scale, and larger objects at with lower resolution. As seen in figure 2.13, the fence post is better segmented at the large scale, while the road/divider region is better segmented with the lower scale.

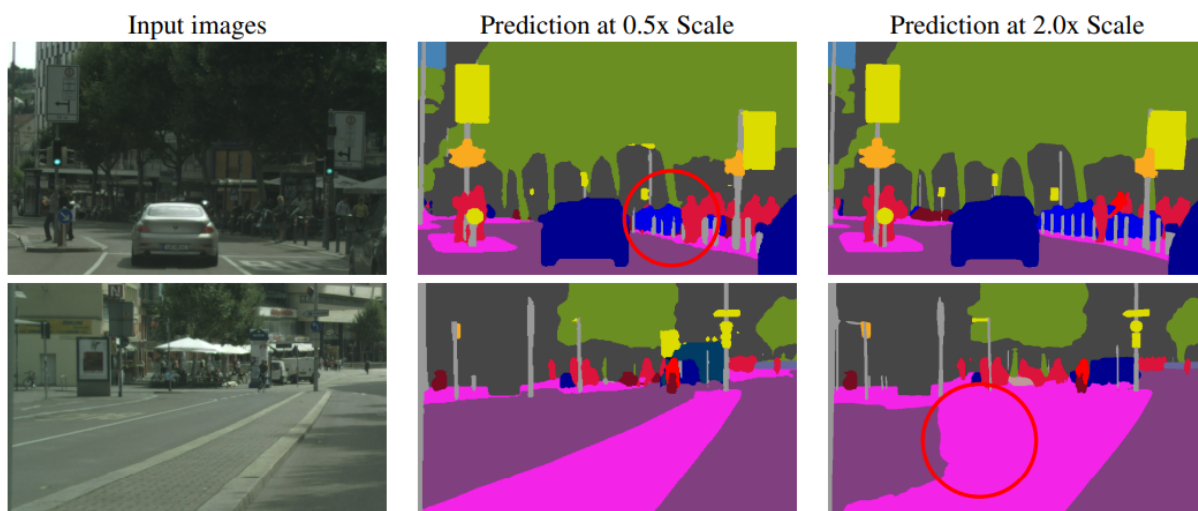


Figure 2.13: NVIDIA predictions at 0.5x & 2.0x scale [33]

## DEXTR-PyTorch

DEXTR-PyTorch is an object segmentation algorithms that uses CNN with extreme points in an object to generate precise object segmentation [34]. The extreme points are created on the far most left, right, top and bottom, and contains a 2D Gaussian centered in them. This creates a heatmap which is contained inside a bounding box for the object of interest. The CNN then learns to transform this information into segmentation. Figure 2.14 shows the architecture of DEXTR, where the image first get extreme points, then processed by CNN to produce a segmented mask. This process works for instance, semantic, video and interactive segmentation.

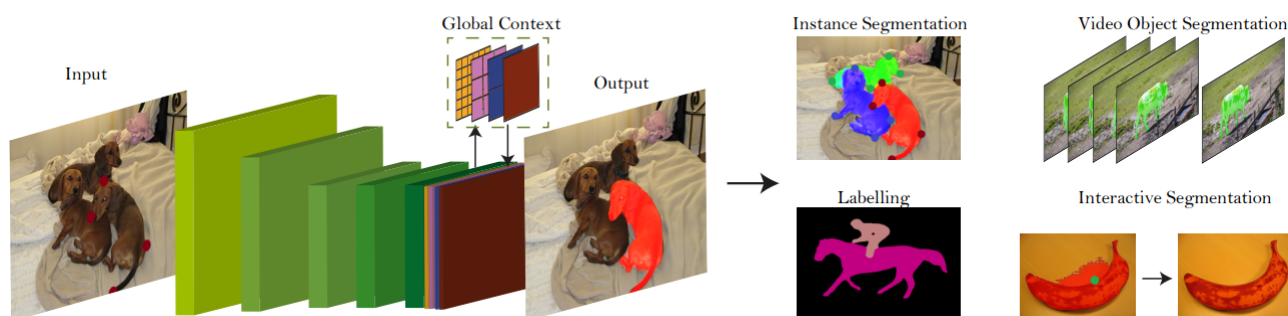


Figure 2.14: Architecture of DEXTR [34]



## 2.5 Pose estimation

For computer vision it is important to figure out the position and orientation of objects in a picture. For a machine to work with the object on the picture, the object needs to be placed in a coordinate system shared by the machine. Since the machine is to disconnect the cables, it needs to know where the cables are in a 3D environment, which is estimated by the pose estimate. A solution is to use the so-called pinhole camera model as shown in figure 2.15. 3D points are formed into an image plane using a perspective transformation [35].

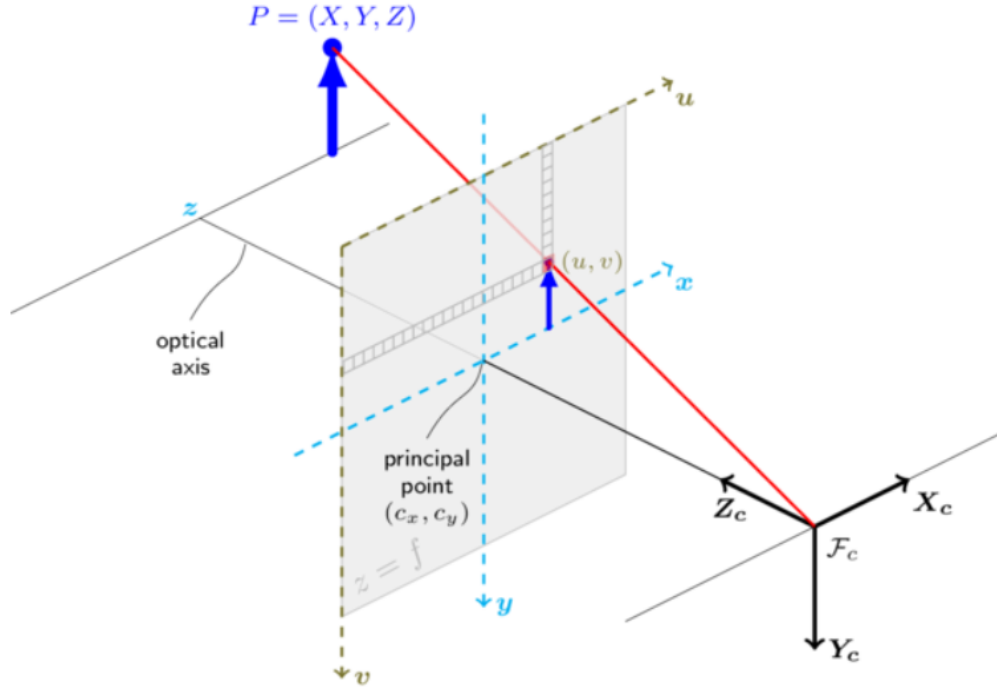


Figure 2.15: Pinhole camera model [35]

From figure 2.15, equation (2.5) can be calculated.

- $s$  is the scaling factor
- $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  are the 3D points in the world coordinate space
- $\mathbf{u}, \mathbf{v}$  are the x and y pixel coordinates
- $\mathbf{A}$  is camera matrix
- $c_x, c_y$  principal point that usually at the image center
- $f_x, f_y$  The focal lengths expressed in pixel units

$$s\mathbf{m}' = \mathbf{A}[\mathbf{R}|\mathbf{t}]\mathbf{M}' \quad (2.5)$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.6)$$

Once estimated, the matrix of intrinsic parameters can be re-used as long as the focal length is fixed. The  $[\mathbf{R}|\mathbf{t}]$  matrix is called extrinsic parameters, and is used to describe the camera

motion around a static scene [36], or the other way around when the camera is still and the object is in motion. When  $z \neq 0$ , the transformation above equals the following:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \quad (2.7)$$

Solving equation (2.7) for  $X, Y, Z$  gives:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \left( s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \mathbf{A}^{-1} - t \right) \mathbf{R}^{-1} \quad (2.8)$$

The formula is used to determine pose estimation, and transform pixel coordinates to world coordinates.

# Chapter 3

## Method

### 3.1 Hardware

For this project, several components and hardware is needed. The Libres project provided several important parts which is explained in this section.

#### 3.1.1 Zivid camera



Figure 3.1: Zivid One + camera [16]

The camera used for this project is a Zivid One+M. A Zivid One+ camera is seen in figure 3.1. The camera uses structured light, and is capable of capturing 2.3 Megapixels (1920 x 1200). That is 2.3 million point in point cloud per capture. It can give data in 3D (XYZ), or in color (RGB). The camera is mounted as an eye in hand system at the end of a ABB robot.

#### 3.1.2 ABB robot

The robot manipulator used in this project is ABB IRB 4400 robot manipulator as shown in figure 3.2. The manipulator is a 6-axis industrial robot used for flexible robot-based automation [37]. With a range from medium to heavy handling, capable of loads up to 60 kg, the robot is a good match for this work, as its design allows for fast and smooth motions withing the working range [37].

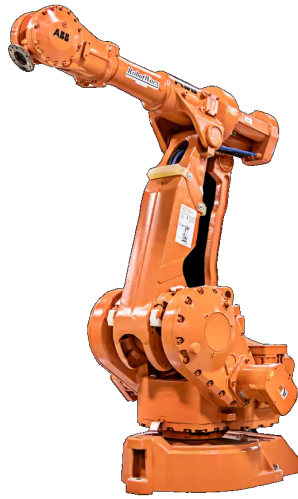


Figure 3.2: IRB 4400 Robot [37]

The working range of the ABB is shown in figure 3.3.

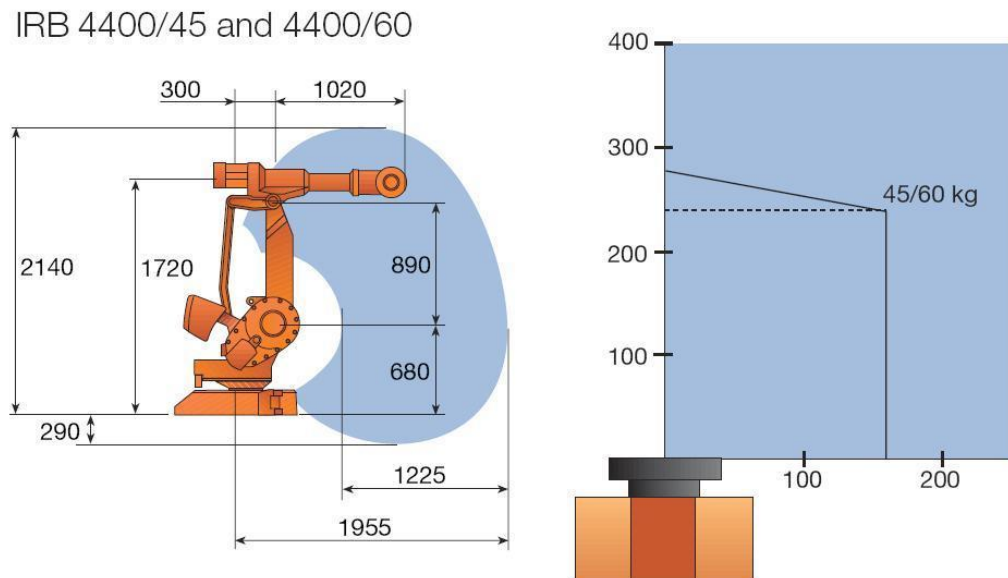


Figure 3.3: IRB 4400/60 dimensions and range [37]

The IBR 4400 is also equipped with a controller (IRC5), that can be used to control the robots joints. This is useful when taking images of cables in different positions.

## 3.2 ROS Setup

The ROS melodic was used in this project because it runs on Ubuntu 18.04, while ROS kinetic runs on Ubuntu 16.04. As of 2021 Ubuntu 16.04 is no longer supported, therefore Ubuntu 18.04 was used, along with ROS melodic.

After the Ubuntu is set up, ROS is set up by open a terminal and type in:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release ...  
-sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

This is set up the repositories, and allows the computer to accept software from packages.ros.org.

Then to set up the ROS key for the computer:

```
sudo apt install curl  
  
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | ...  
sudo apt-key add
```

Then make sure everything is updated:

```
sudo apt update
```

Then, installing ROS with:

```
sudo apt install ros-melodic-desktop-full
```

Setting up the environment for ROS is done with:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc  
  
source ~/.bashrc
```

And installing additional dependencies for packages like python etc:

```
sudo apt install python-rosdep python-rosinstall ...  
python-rosinstall-generator python-wstool build-essential  
  
sudo apt install python-rosdep  
  
sudo rosdep init  
  
rosdep update
```

Next step is to install the Zivid ROS.

For this a nvidia driver is needed. For the machine a nvidia 460 is used with:

```
sudo add-apt-repository ppa:graphics-drivers/ppa  
  
sudo apt update  
  
sudo apt install nvidia-driver-460  
  
sudo reboot
```

Also need to install OpenCL and GPU using:

```
sudo apt install -y clinfo
sudo /usr/bin/clinfo
/usr/bin/clinfo -l
```

For the installation of Zivid, it can be found on github at <https://github.com/zivid/zivid-ros>.

## 3.3 Object Detection

### 3.3.1 Image collection and labeling

#### ROS

To create the training data, labeled images of the cables are needed. Already labeled data is used from a previous project [5]. The images were collected on an Apple iPhone 7, and recorded videos of a VW battery pack from different angles. Adobe Premier Pro was used to split up the videos into different frames.

Using LLabelMe, the images could be annotated so that the Mask R-CNN would know what to train for. The labeling is to highlight areas of interest, in this case the cables, so that the model knows what to detect. The car batteries are set up of different cables, and were therefore classified into different classes using LabelMe's polygon tool. The cables are assigned a mask and a class to each cable of interest. The classes used are showed in table 3.1 and visualized in figure 3.4.



Figure 3.4: Segmentation VW LIB [5]

Class	Description (Mask Color)
type1	zip ties (yellow)
type2	BMS connector (blue)
type3	Cable connector (red)
type4	Cable restraint (red)
type5	Orange plastic cables cell side (dark orange)
type6	Pole cover (green)
type7	Cable connector cell side (red)
type8	Orange wire harness (azur blue)
type9	Ground cable (green)
type10	Ground cable connector (purple)
type11	Main wire harness (white)

Table 3.1: Class labels

Each labeled data was made in a JSON file for each image. All the JSON files together with the raw image files are stored in a folder called dataset. Mask R-CNN only accepts COCO json files, and an additional network (find name) uses PASCAL VOC files, so the files needed to be converted. To convert to COCO json files, a script called labelme2coco.py (see appendix A.2) was used to convert the dataset into a COCO json file. To convert the files to PASCAL, the COCO Json files were used and the website <https://roboflow.com/convert/coco-json-to-pascal-voc-xml> was used to convert the files from COCO to PASCAL.

## Matlab

The Matlab semantic segmentation uses different data than COCO and PASCAL, so the *image labeler* was used to label the cables. The images was taken on a OnePlus Nord2 phone, then sent to the PC. The images from the phone are 4096x3072 pixels, which caused a problem when training the data. Because of the large pixel size, the GPU ran out of memory, so the images needed to be reduced. A script (cite script here) was used to take the original images and downsize them to 448x448. This size was chosen because the resolution was still good enough to recognize cables without being too big. The images were then put into ImageLabeler on Matlab as shown in figure 3.5.

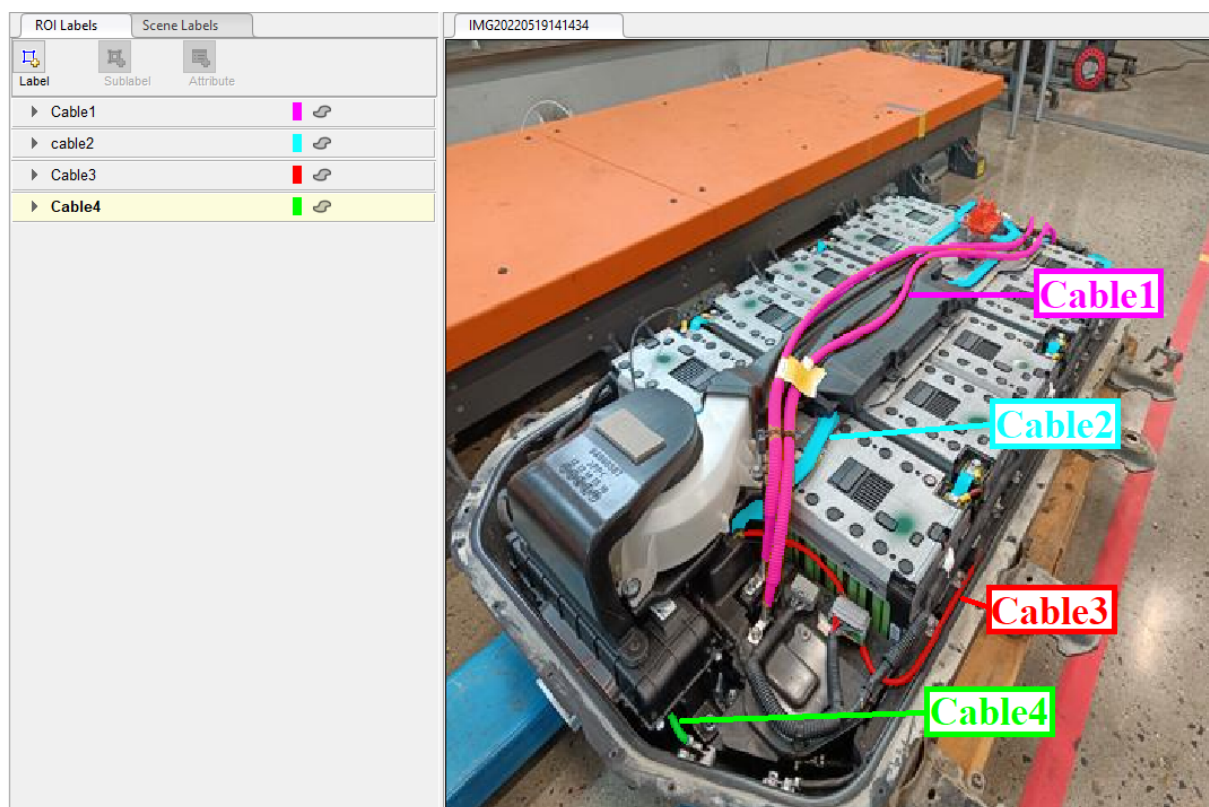


Figure 3.5: ImageLabeler

Class	Description (Mask Color)
Cable1	orange cable (magenta)
Cable2	blue connectors (Turquoise)
Cable3	red cable (red)
Cable4	green cable (green)

Table 3.2: Class labels



The labeldata is exported, and could either be sent to Workspace or to a file as shown in figure 3.6.

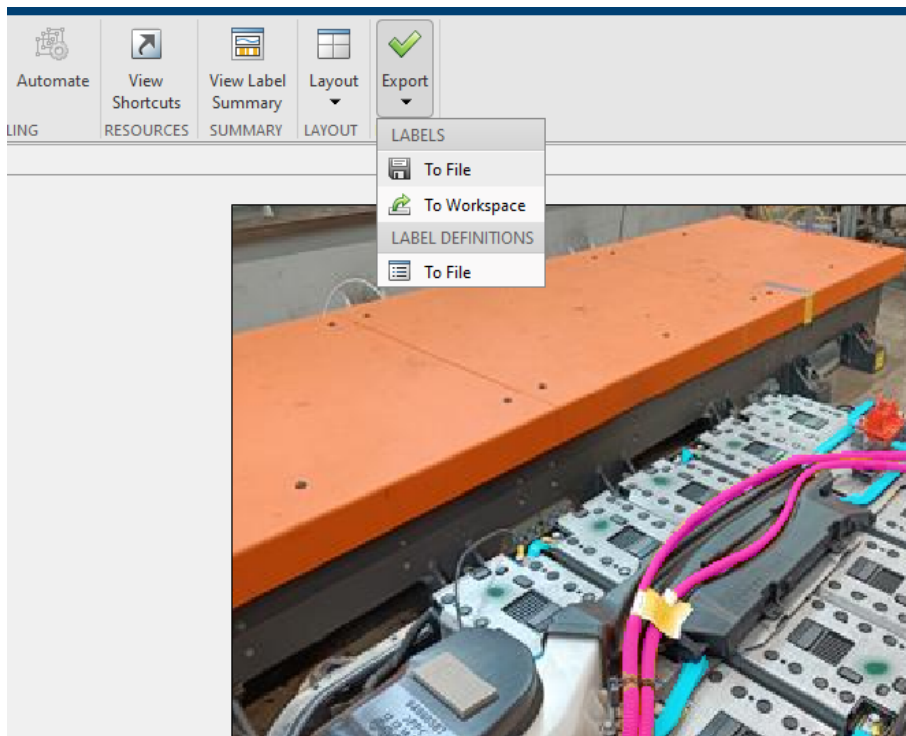


Figure 3.6: ImageLabeler export

To save it for later use, the labeldata was exported to a file. This creates a `gTruth.mat` file and a `PixelLabelData` file with label images. The `gTruth.mat` file contains the `LabelDefinitions` and `LabelData`, which tells about the classnames and labelIDs. Both are needed to train the network. In the matlab script, the image and label directory is set. The training images are stored with the `imageDataStore` command. The pixel data is stored with `pixelLabelDatastore` command using the `gTruth.mat` file.

### 3.3.2 Training

Training the model, a set of algorithms are to be used. Detectron2, NVIDIA/semantic-segmentation, DEXTR-PyTorch and Matlab semantic segmentation is to be used to compare which one is best suited for detecting the cables.

#### Detectron2

The Detectron2 is using bounding boxes and mask segmentation.

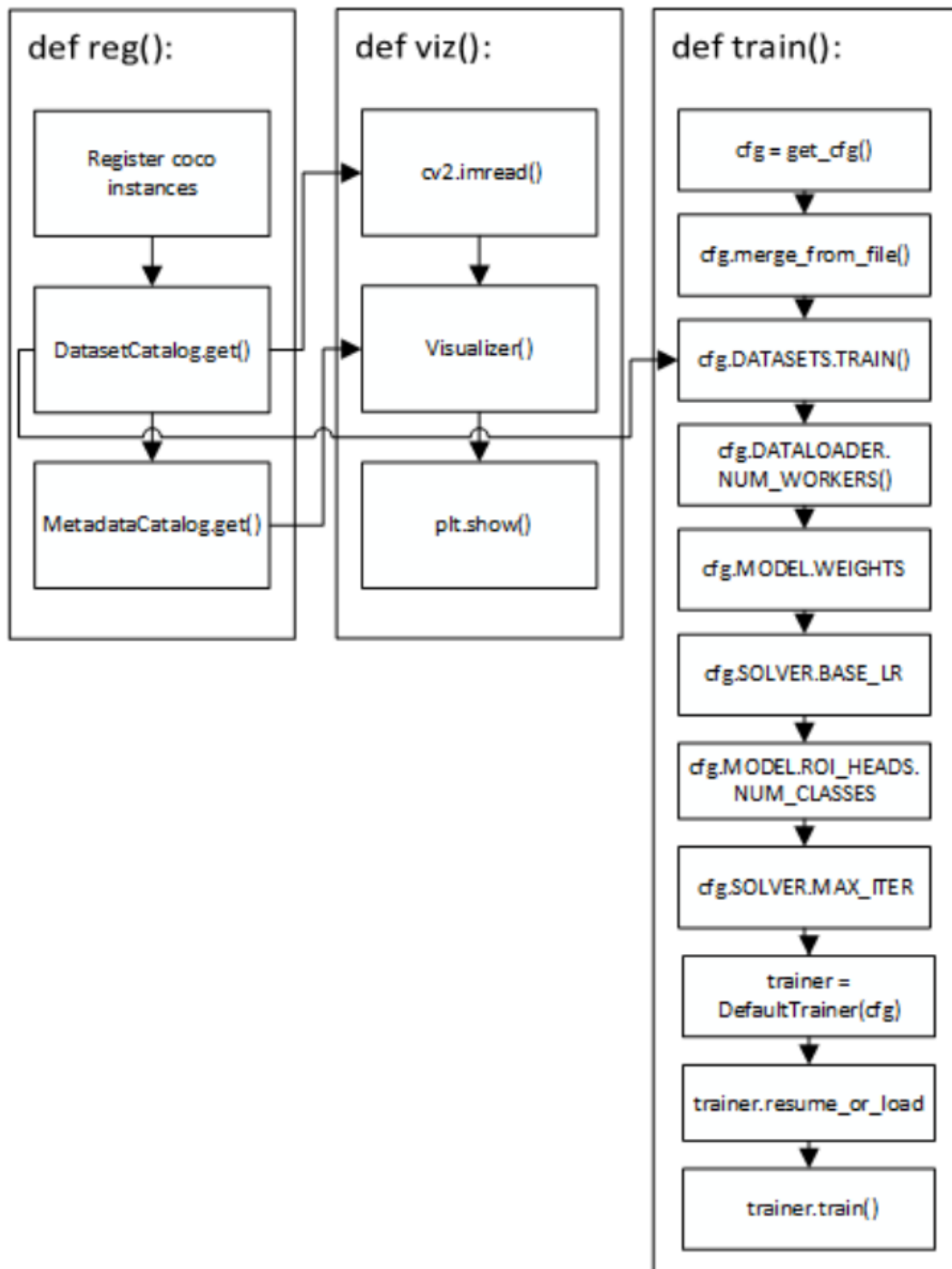


Figure 3.7: Detectron2 Training script structure [5]

The figure shows the structure of the "detectron2Traning.py" script (see appendix A.1). It consists of different components used to train the Mask R-CNN model from Detectron2 library. The three main functions are:

**def reg():**

This function loads the training data in the Detectron2 library using a function called *register\_coco\_instances*. For the function to work, it needs three arguments. The first is the name of the dictionary where the name *data\_train* assigned for the training data. The second argument is the path to the COCO annotation json file. This contains the data about the bounding boxes and mask. The last argument is the path to the folder containing the training images. The two remaining functions, *DatasetCatalog* and *MetaDataCatalog* are used to help extract the training data and metadata from the *data\_train* dictionary.

**def viz():**

To confirm the data is loaded correctly this function is used by visualizing the training data. It consists of three subfunctions: *cv2.imread()*, *visualizer()* and *plt.show()*. *cv2.imread()* is an OpenCV function used to load the training data images. *visualizer()* then loads the metadata catalog of the training dataset. Together with *cv2.imread()* they visualize the data. *plt.show()* then displays the loaded training data.

**def train():**

In this function, the goal is to set the config of the training. This is done by the *get\_cfg()* class. From this class there are six arguments used to config the training of the model. The first, *merge\_from\_file()*, loads the model (Mask R-CNN) used for the training session. The second loads the data used for training the model from the *def reg()* function. The third argument, *NUM\_WORKERS* takes the training data and sets the amount of subprocess used to load the data. *MODEL.WEIGHTS* loads the weights of the model, but only if the if the training is on the same dataset with the same parameters. Then the fifth argument sets the learning rate, which is the most important argument in the config. This determines how quickly the models adapts to the problem, and effect the training loss. The sixth argument, *SOLVER.MAX\_ITER* sets the maximum number of training iterations. When the config is set, the function *DefaultTrainer* is loaded. The *trainer.resume\_or\_load* function can then be configured to either continue from the last iterations or start over. In the end the function *trainer.train()* is called and the training can begin.

## NVIDIA/semantic-segmentation

NVIDIA/semantic-segmentation was another segmentation algorithm to be used. It was cloned from github using:

```
git clone --recursive https://github.com/NVIDIA/semantic-segmentation.git
```

Go get the docker file, the

```
sudo snap install docker
```

was used to install, and built with

Inside the semantic-segmentation folder, build the docker using:

```
docker build -t nvidia-segmentation -f Dockerfile .
```

This did not work, so "sudo" command was tried:

```
sudo docker build -t nvidia-segmmentation -f Dockerfile .
```

This also resulted in errors, so NVIDIA apex was cloned and installed a C++ extensions with:

```
git clone https://github.com/NVIDIA/apex
```

```
cd apex
pip install -v --no-cache-dir --global-option="--cpp_ext" ...
--global-option="--cuda_ext" ./
```

Due to error with the download of C++ extension, it needed nvidia-cuda-toolkit, which was installed with the following command:

```
sudo apt install nvidia-cuda-toolkit
```

The C++ extensions were installed with:

```
pip3 install -v --no-cache-dir --global-option="--cpp_ext" ...
--global-option="--cuda_ext" ./
```

## DEXTR-PyTorch

The algorithm is cloned from <https://github.com/scaelles/DEXTR-PyTorch>. Then installing dependencies to the software using:

```
conda install pytorch torchvision -c pytorch
conda install matplotlib opencv pillow scikit-learn scikit-image
```

Pytorch torchvision is a library for computer vision for pytorch, while matplotlib is used for image processing.

DEXTR-PyTorch also need tensorboard, that helps with monitoring and display data.

```
pip3 install tensorboard tensorboardx
```

Addition, because this project uses python3 and not python2, some additional tools are needed to be installed for the algorithm to work, such as matplotlib and opencv for python3.

```
pip3 install matplotlib
```

To install opencv-python, these steps needed to be done as well:

```
pip3 install scikit-build
```

```
pip3 install cmake
```

```
pip3 install --upgrade pip
```

```
pip3 install opencv-python
```

So that the script can download the dataset, inside *pascal.py* download is set to *True*.

Because of difficulty with GPU memory, the "trainBatch" and "testbatch" were set to low, as to avoid using too much memory when training.

Then to run the training, run:

```
sudo python3 train_pascal.py
```

The training script was build up as follow:

- Set GPU-id
- Setting Parameters Set the parameters like epoch, test and training batch.
- Results and model directories Creates a new directory for every training run.
- Network definition Set the definitions for the network, such as pretraining, inputchannels, weights, etc.
- Training Network
  - Logging into Tensorboard
  - Use the following optimizer
  - Preparation of the data loaders
  - Train variables
  - Main Training and Testing Loop

- Save the model
- One testing epoch
- Generate result of the validation image
  - Forward of the mini-batch
  - Save the result, attention to the index jj

## Matlab

Matlab takes the training images from the *imageDataStore* , with the label data, class names and label ID from *pixelLabelDataStore* and combines them into trainingData. The network is then creates with the following inputs into layer:

numFilters	64
filterSize	3
numClasses	4

Table 3.3: Semantic segmentation network layer input

The training options for the network is set up as following:

InitialLearningRate	1e-3
MaxEpochs	400
MiniBatchSize	14

Table 3.4: Network training options

The network is then trained by running the *trainNetwork* function, and uploaded to Workspace. Additional, combined testData is created and stored in Workspace to be used to evaluate the network during training. From here, the app *Deep Network Designer* is used. The app is located as shown in figure 3.8.

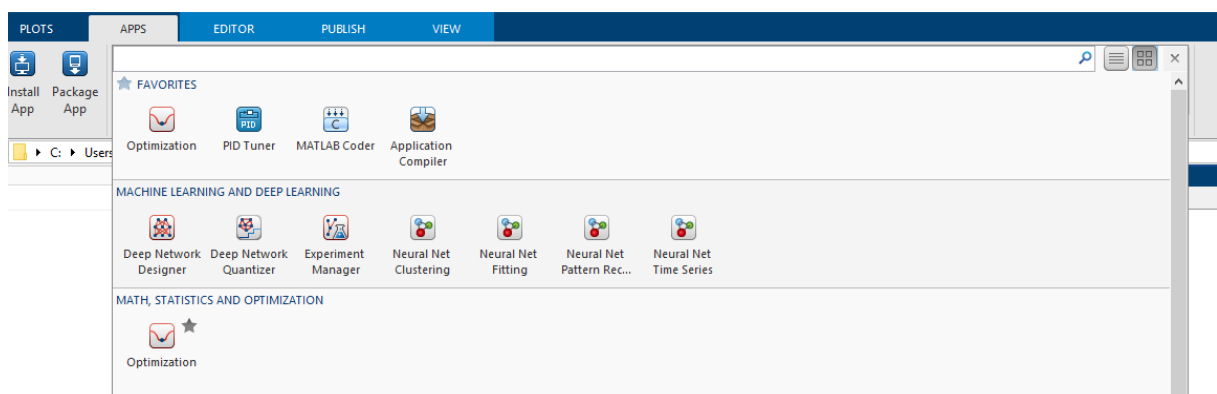


Figure 3.8: Deep Network Designer App

From here the network is uploaded from Workspace as shown in figure 3.9

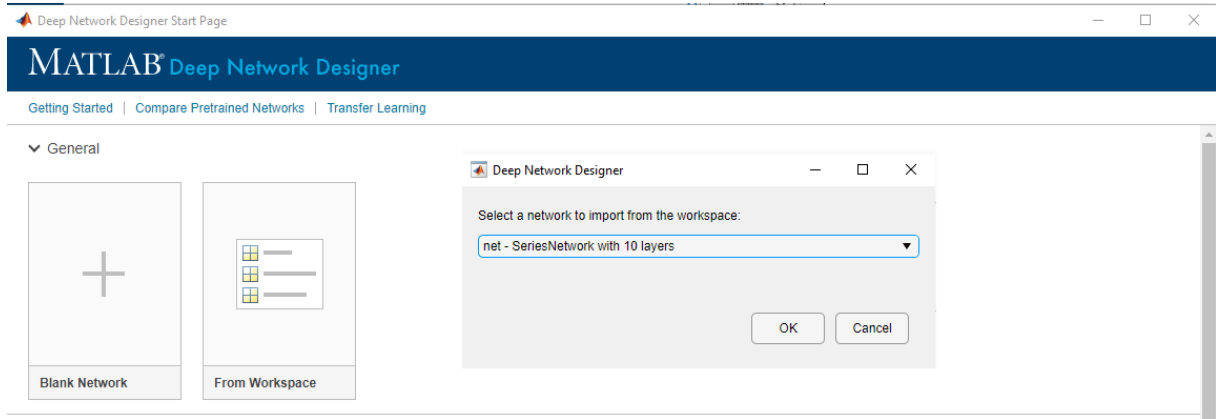


Figure 3.9: Upload network to Deep Network Designer App

Then the training data and test data is uploaded as shown in the figure 3.10

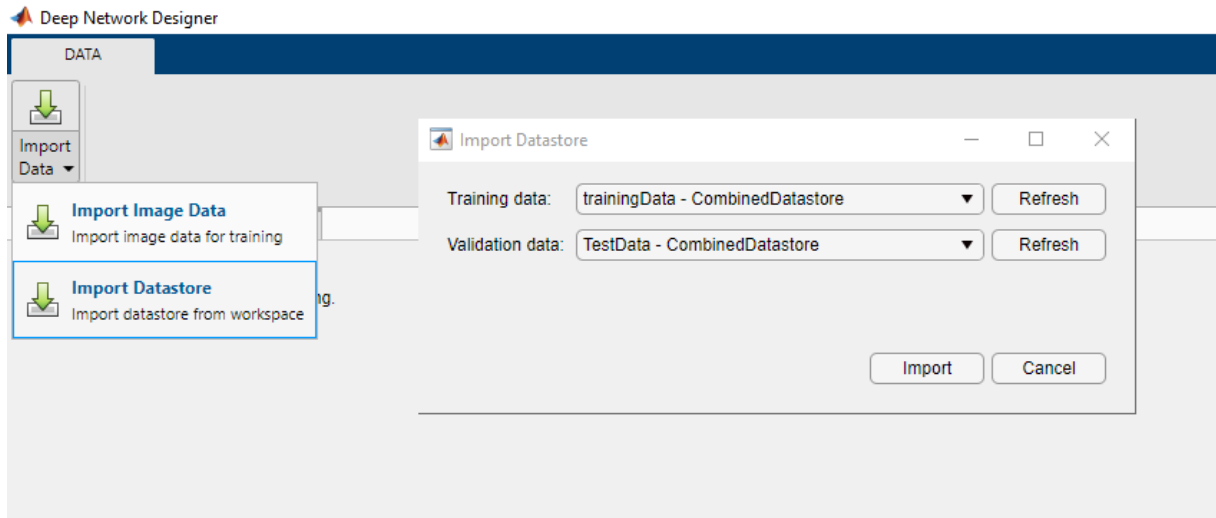


Figure 3.10: Import training & validation data to App

The same training options are used as shown in table 3.4 in training the network in the app as shown in figure 3.11.

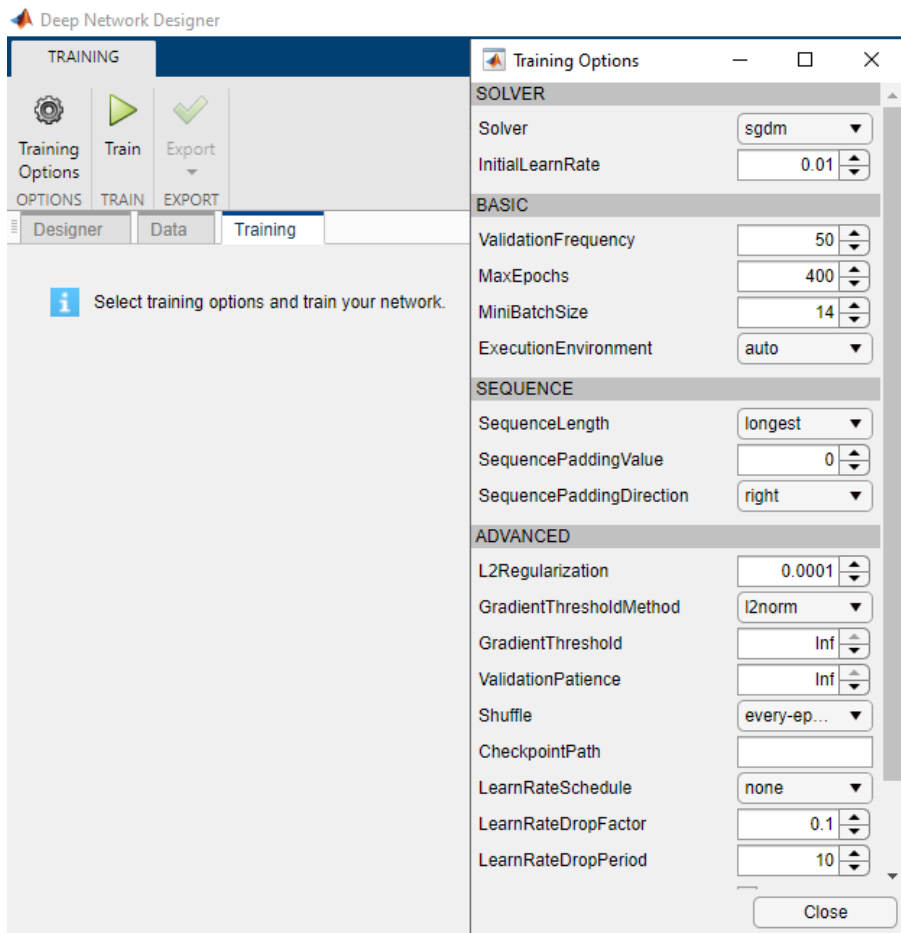


Figure 3.11: Training options

Then press Train to train the network.



### 3.4 Pixel to world coordinates

The structure of the pose estimation wrapper consist of two separate interconnected scrips shown in figure 3.12 below.

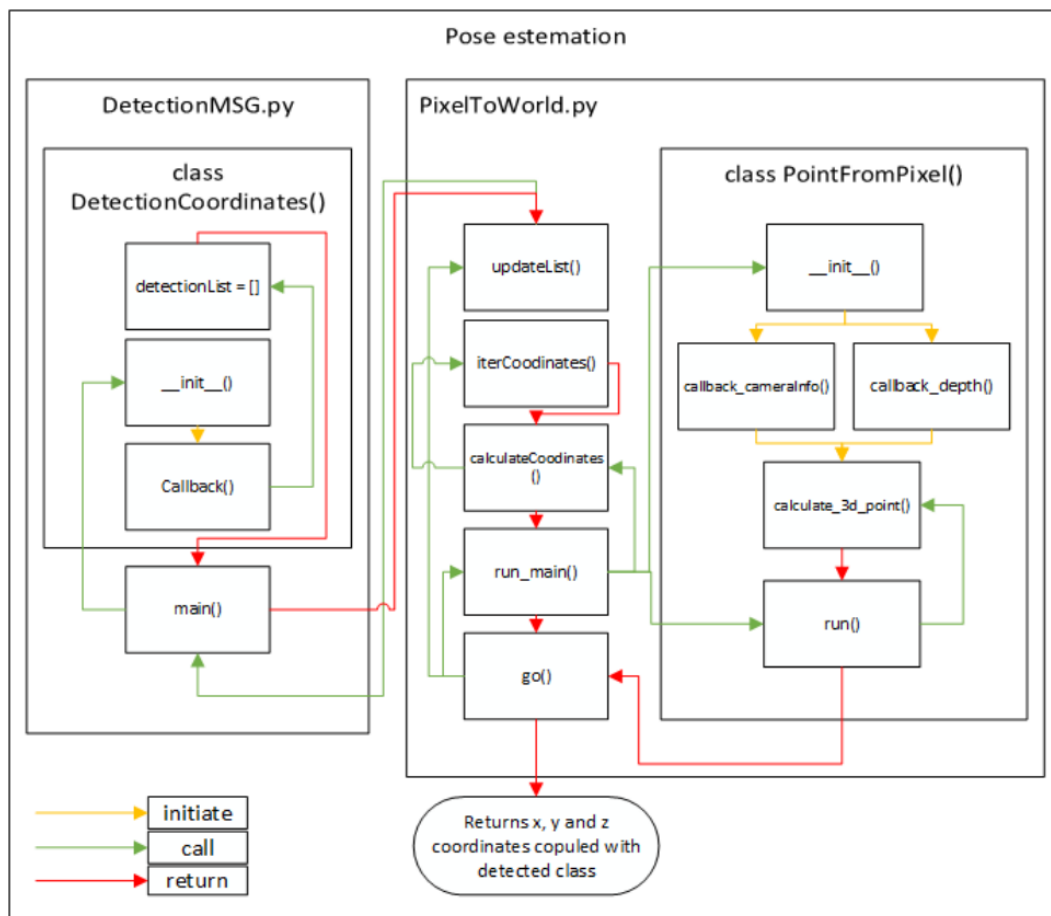


Figure 3.12: Pose estimation pixel to world coordinates script [5]

The script estimates the world coordinates based on the object detection result message. The two main task are *DetectionMSG.py* and *PixelToWorld.py*.

#### DetectionMSG.py

The first script of the pose estimation is to convert the detection results to pixel coordinates. It is responsible for acquiring and extracting the bounding boxes, class names and mask results of the detection result message. *DetectionMSG.py* consist of:

- *Main()*: This function runs and returns the results message. When the main function is called upon, it calls the `DetectionCoordinates` class. Then the `__init__` function runs and initialize the `Callback()` function. The `Callback()` sends it to the `detectionList` which returns to the main function.
- *class DetectionCoordinates()*
  - `__init__()`: It initialize the class by initializing the ROS subscriber, which is subscribed to the result message of *detectron2<sub>r</sub>os.py* script. The ROS subscriber

initit and calls the *Callback* function. The function passes the result message to the *Callback* function.

- *Callback()*: This function extracts the data from the result message, which is to be used for the pose estimation. The *detectionList* then calls for the extracted data from *Callback*, and then returns to the main function.

## PixelToWorld

This script creates 3D world coordinates based on the list of pixel coordinates returned by the *DetectionMSG.py* script. *DetectionMSG.py* calculates 3D coordinates from the detections and returns them with the *moveitCommander.py* script. *PixelToWorld* then transforms the coordinates to a 3D world coordinates, and is set up like this:

- Class *pixelToPoint()*
  - *\_\_init\_\_()*: This function initialize the *PointFromPixel* class and launches two ROS subscribers, camera info and depth image. It then initialize the *callback\_cameraInfo* and *Callback\_Depth* functions. These then initialize the *calculate\_3d\_point* which determines the world coordinates.
  - *callback\_cameraInfo()*: Is used to define the Pinhole Camera Model parameters.
  - *callback\_depth()*: Fetches depth image from the ROS subscriber, and declare the depth parameter of the class.
  - *calculate\_3d\_Point()*: Using the parameters from *callback\_cameraInfo* and depth from *callback\_depth*, it calculates the 3D coordinates. This is done with four steps.
    1. Look up the supplied pixel coordinates with its corresponding depth image coordinates.
    2. Create 3D ray using pinhole camera model called *projectPixelTo3dRay*. This makes a 3D ray through a given pixel coordinate.
    3. Normalize the 3D ray, so that the z component is 1.0.
    4. Multiply the 3D ray with the depth pixel coordinates from the first step. This creates the correct z component value.
  - *run()*: Gives the pixel coordinates used by *calculate\_3d\_Point* to calculate the world coordinates. It is called by the *run\_main*, and returns the calculated 3D point to *go*, both outside the class.
- *updateList()*: Calls the main function of *DetectionMSG* script to enquire the detection coordinate list, which values are stored in *result\_list*.
- *iterCoordinates()*: It returns one class and one set of coordinates each time it is called.
- *calculateCoordinates()*: It calculate the center position of the bounding boxes and returns them. It enquire the bounding box and class of detection from *iterCoordinates*.
- *run\_main()*: Initialize the *pixelToPoint* class. It calls the *calculateCoordinates* to get the pixel coordinates, and calls *run* from *PointFromPixel* to return the calculated world coordinate and class.
- *Go()*: Create a list of world coordinates of each detection. It calls the *run\_main* as many times as the length of the *result\_list*. This ensures that the list of coordinates matches the detections.

# Chapter 4

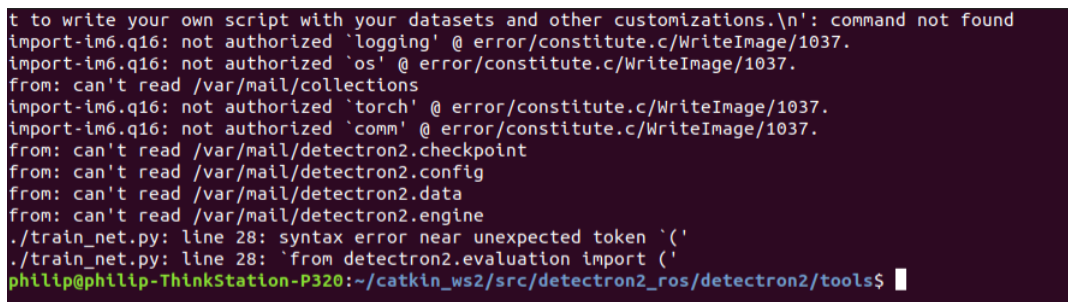
## Results & Discussion

### 4.1 Results

#### 4.1.1 Algorithms & neural networks

##### Detectron2

When trying to run Detectron2, several problems occurred. Most likely because of trouble with CUDA the training would not begin.

A terminal window with a dark background and light-colored text. The text shows a series of error messages from a shell script. The errors include: 'command not found', 'not authorized logging', 'not authorized os', 'not authorized torch', 'not authorized comm', and several 'can't read' errors for files like /var/mail/collections, /var/mail/detectron2.checkpoint, /var/mail/detectron2.config, /var/mail/detectron2.data, and /var/mail/detectron2.engine. The terminal ends with a prompt for 'philip@philip-ThinkStation-P320' in a directory path.

```
t to write your own script with your datasets and other customizations.\n': command not found
import-im6.q16: not authorized 'logging' @ error/constitute.c/WriteImage/1037.
import-im6.q16: not authorized 'os' @ error/constitute.c/WriteImage/1037.
from: can't read /var/mail/collections
import-im6.q16: not authorized 'torch' @ error/constitute.c/WriteImage/1037.
import-im6.q16: not authorized 'comm' @ error/constitute.c/WriteImage/1037.
from: can't read /var/mail/detectron2.checkpoint
from: can't read /var/mail/detectron2.config
from: can't read /var/mail/detectron2.data
from: can't read /var/mail/detectron2.engine
./train_net.py: line 28: syntax error near unexpected token '('
./train_net.py: line 28: 'from detectron2.evaluation import ('
philip@philip-ThinkStation-P320:~/catkin_ws2/src/detectron2_ros/detectron2/tools$
```

Figure 4.1: Detectron2 error code

As shown in figure 4.1, the error in the command window was never solved.

NVIDIA-semantic segmentation

NVIDIA-semantic segmentation would not build the algorithm, and was therefore not trained nor tested.

##### DEXTR-PyTorch

The DEXTR-PyTorch would start to train, but would run into an error due to CUDA running out of memory as shown in figure 4.2. Even when both batch size and epochs were turn down to 1, it would still run out of memory. Solutions to solve this was never found unfortunately, and because of time limitations, other options were prioritized.

```

phillip@phillip-ThinkStation-P320:~/algo/DEXTR-PyTorch$ sudo python3 train_pascal.py
Using GPU: 0
Constructing ResNet model...
Dilations: (2, 4)
Number of classes: 1
Number of Input Channels: 4
Initializing classifier: PSP
Skipping Conv layer with size: torch.Size([512, 2048, 1, 1]) and target size: torch.Size([1, 2048, 3, 3])
Initializing from pretrained Deeplab-v2 model
Files already downloaded and verified
Number of images: 1464
Number of objects: 3507
Files already downloaded and verified
Number of images: 1449
Number of objects: 3427
Training Network
Traceback (most recent call last):
  File "train_pascal.py", line 147, in <module>
    output = net.forward(inputs)
  File "/home/phillip/algo/DEXTR-PyTorch/networks/deeplab_resnet.py", line 193, in forward
    x = self.layer3(x)
  File "/home/phillip/.local/lib/python3.6/site-packages/torch/nn/modules/module.py", line 493, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/phillip/.local/lib/python3.6/site-packages/torch/nn/modules/container.py", line 92, in forward
    input = module(input)
  File "/home/phillip/.local/lib/python3.6/site-packages/torch/nn/modules/module.py", line 493, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/phillip/algo/DEXTR-PyTorch/networks/deeplab_resnet.py", line 55, in forward
    out = self.conv2(out)
  File "/home/phillip/.local/lib/python3.6/site-packages/torch/nn/modules/module.py", line 493, in __call__
    result = self.forward(*input, **kwargs)
  File "/home/phillip/.local/lib/python3.6/site-packages/torch/nn/modules/conv.py", line 338, in forward
    self.padding, self.dilation, self.groups)
RuntimeError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 0; 1.95 GiB total capacity; 959.05 MiB already allocated; 41.19 MiB free; 2.95 MiB cached)
phillip@phillip-ThinkStation-P320:~/algo/DEXTR-PyTorch$ >

```

Figure 4.2: DEXTR-PyTorch

### Matlab semantic segmentation

The semantic segmentation in Matlab showed promising results. The segmentation of the four cables did well in the training progress. The network used 400 for Max Epochs, and 16 for MiniBatchSize, as anything higher would result in the GPU to run out of memory. The results in figure 4.3 shows the training progress after 400 iterations.

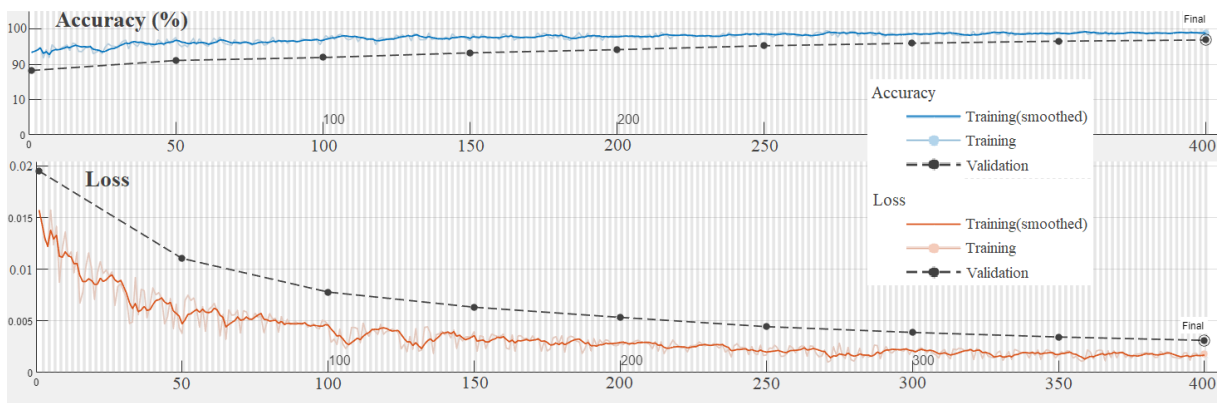


Figure 4.3: Training of Matlab Network

Global Accuracy	Mean Accuracy	Mean IoU	Weighted IoU	Mean BF Score
0.97845	0.75835	0.5755	0.96923	0.9652

Table 4.1: Global- and Mean Accuracy, and Mean- and Weighted IoU

As shown in table 4.2 on how well the network segmented the different classes on the test images.

True class	Predicted class			
	Cable1	Cable2	Cable3	Cable4
Cable1	97.2%	0.2%	2.5%	0.0%
Cable2	0.2%	99.6%	0.0%	0.2%
Cable3	19.5%	0.5%	79.9%	0.1%
Cable4	2.5%	8.7%	0.5%	88.4%

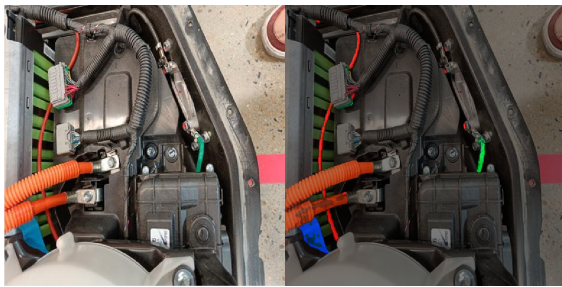
Table 4.2: Normalized Confusion Matrix (%)

The data for the different classes can be seen in table 4.3. Low IoU indicates that the area segmented for the labels are not overlapping properly, or are segmenting portions that are not part of the cable.

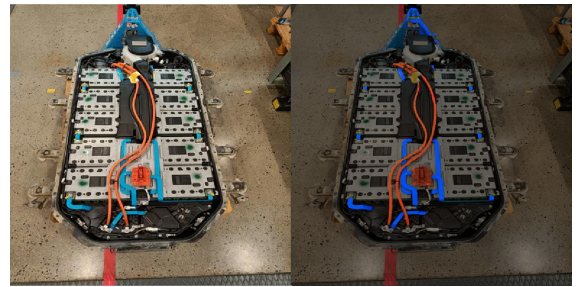
	Accuracy	IoU	MeanBFScore
Cable1	0.97204	0.94588	0.94629
Cable2	0.9963	0.98905	0.98597
Cable3	0.79935	0.66883	0.72094
Cable4	0.88354	0.838	0.96769

Table 4.3: Class accuracy and IoU

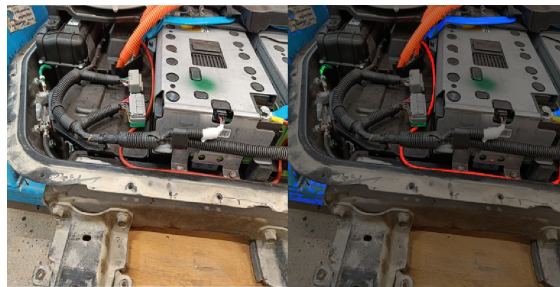
Figure 4.4 show the network segmenting the classes.



(a) Semantic segmentation mask 1



(b) Semantic segmentation mask 2



(c) Semantic segmentation mask 3

Figure 4.4: Semantic segmentation mask

## 4.2 Discussion

### Detectron2

Trying to get the Detectron2 to train instance segmentation resulted in errors. The reason for this might have been because of the driver for CUDA. According to the net information, CUDA 11.4 were supposed to be supported by Detectron2, but the Detectron2 training was not running as it should. The driver for CUDA may have been a contributing factor. There might also be a problem with python not executing at all, according to the errors displayed in the command window. A solution to use

```
#!/usr/bin/env python3
```

in the code was tried, but it did not work. Some minor solutions were looked into, but due to time limitations and stagnation with Detectron2, the focus was shifted to DEXTR-PyTorch in hopes to get progress with another algorithm.

This issue was discovered too late into the project when focus was shifted to the training of DEXTR-PyTorch.

### NVIDIA/semantic-segmentation

For unknown reasons, the NVIDIA/semantic-segmentation algorithm would not build. Several attempts to find a solution was tried, and several tools were downloaded as shown below:

```
pip3 install torch==1.1.0 torchvision==0.3.0 -f ...  
https://download.pytorch.org/whl/cu90/torch_stable.html
```

```
conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
```

```
sudo apt-get install python3.7-dev
```

```
pip install mujoco-py==0.5.7
```

The reason for it not being able to build the algorithm could be because of the wrong version of python, or lack of needed ROS packages.

### DEXTR-PyTorch

When training instance segmentation on cables with DEXTR-PyTorch, it would not complete training. The reason might also be trouble with CUDA version, same as with Detectron2. When the training data was loaded it might have loaded all at once, instead of piece by piece, and overloading the GPU. As mentioned in Chapter 4, epochs and both training and test batch size were turned down to reduce the data size, but to no avail.

The problem might have been that the computer was not optimal for the required GPU memory for training of segmentation algorithms. The PC used to run Ubuntu and ROS was not the same PC used with Matlab. This is the reason for Matlab not having a problem with GPU running out of memory, while DEXTR did. A solution could have been to use the PC used on Matlab, but due to it being a personal stationary PC at home, it was not used with Ubuntu out of fear of corrupting the already existing OS.

## Matlab semantic segmentation

The training of the Matlab segmentation Network appeared to be a success, but the project was unable to get images with the Zivid camera to run with the Matlab semantic segmentation network. It was able to capture 2D images from the Zivid camera, but when applying the images into the network with *semanticseg* function, it crashed because the image type was not correct.

The results for Matlab could have been explored deeper, but creating more labels was not prioritized in the project due to time management. More training data with different battery packs would have made better results as different packs and cables would show if the network would work over a broader selection of cables.

One problem with the segmenting was the dark areas with black cables. With 448x448 resolution on the images, the dark areas and cables was indistinguishable, and impossible to label correctly. A solution might have been to use images with higher resolution than 448x448, and with a sensor with a larger gain to get better contrast in dark areas.

Point cloud was also tested. Point cloud images was taken with the Zivid camera. When uploading it to Lidar Labeler in Matlab it would freeze the computer. From tutorials, it also seemed impossible to do semantic segmentation with Lidar Labeler, and was not furthered tested.

## Comparison

The project was not able to complete the different algorithms and test them against each other or test them on a battery pack in real time. However, getting Matlab to run semantic segmentation with rather high accuracy, and with more training and testing with higher image resolution, the trained network in Matlab could prove good enough to use for the battery disassembly.

# Chapter 5

## Conclusions

The conclusion of this thesis is a partial success. The thesis was to find a suitable segmentation algorithm for the battery disassembly plant. As shown in chapter 4, the Matlab semantic segmentation shows good results. However, because the project was unable to run the test on Detectron2, NVIDIA/semantic-segmentation, DEXTR-PyTorch, the project can not conclude if Matlab is the best option of the four.

### Further work

The scope of this thesis has been changed over time. This has made room for improvements and further work. If work on this thesis is to be continued, the following work should be considered:

- Algorithm & pose estimation  
The algorithm should be tested further to obtain good accuracy on all cable types the battery packs would contain. When the algorithm gives good results, the algorithm and pose estimation should be combined to get accurate location of the cables.
- Cable gripper  
A cable gripper must be capable of detaching and lifting the cables to be able to remove them. As the batteries still have volts in them, cutting or damaging the cables must therefor be avoided at all costs. A gripper must be strong enough to detach, but still gentle enough to avoid damage should be designed to fit on the ABB robot.



# Bibliography

- [1] *Antall elbiler ned - men elbilandelen øker*. URL: <https://ofv.no/aktuelt/2020/antall-elbiler-ned-men-elbilandelen-%C3%B8ker>.
- [2] *LIBRES: BATTERY RECYCLING IN A CIRCULAR ECONOMY*. URL: <http://mission-innovation.net/our-work/mission-innovation-breakthroughs/libres-battery-recycling-in-a-circular-economy/>.
- [3] *Global EV Sales for 2021*. URL: <https://www.ev-volumes.com/>.
- [4] Atle Christiansen. *The University of Agder dismantles electric car batteries for reuse*. URL: <https://www.uia.no/en/news/the-university-of-agder-dismantles-electric-car-batteries-for-reuse>.
- [5] SEBASTIAN THORSTAD HANSEN. *Development of Computer Vision System to detect, identify and localize flexible structures used in EV batteries*. University of Agder, 2021.
- [6] *ROS/Introductions*. URL: <http://wiki.ros.org/ROS/Introduction>.
- [7] *Nodes*. URL: <http://wiki.ros.org/Nodes>.
- [8] *Topics*. URL: <http://wiki.ros.org/Topics>.
- [9] *msg*. URL: <http://wiki.ros.org/msg>.
- [10] *Services*. URL: <http://wiki.ros.org/Services>.
- [11] *Master*. URL: <http://wiki.ros.org/Master#:~:text=The%20ROS%20Master%20provides%20naming,nodes%20to%20locate%20one%20another..>
- [12] *Robot Operating System*. URL: [https://fr.wikipedia.org/wiki/Robot\\_Operating\\_System](https://fr.wikipedia.org/wiki/Robot_Operating_System).
- [13] *What Is MATLAB?* URL: <https://se.mathworks.com/discovery/what-is-matlab.html>.
- [14] Yuxing Xie, Jiaojiao Tian, and Xiao Xiang Zhu. “Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation.” In: *IEEE Geoscience and Remote Sensing Magazine* 8.4 (2020), pp. 38–59. DOI: 10.1109/MGRS.2019.2937630.
- [15] F. Couwleers Ø. Skotheim. *Structured light projection for accurate 3D shape determination*. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.595.1192&rep=rep1&type=pdf>.
- [16] *Zivid One+ industrial 3D camera*. URL: <https://shop.zivid.com/products/zivid-one-industrial-3d-camera>.
- [17] Zivid AS. *Introduction to Stop*. URL: <https://support.zivid.com/v2.4/reference-articles/introduction-to-stops.html>.
- [18] Zivid AS. *Exposure Time*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/acquisition-settings/exposure-time.html>.

- [19] Zivid AS. *Aperture*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/acquisition-settings/aperture.html>.
- [20] Wikipedia. *Aperture*. URL: <https://en.wikipedia.org/wiki/Aperture>.
- [21] Zivid AS. *Projector Brightness*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/acquisition-settings/projector-brightness.html>.
- [22] Zivid AS. *Gain*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/acquisition-settings/gain.html>.
- [23] Zivid AS. *Contrast Distortion Filter*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/processing-settings/contrast-distortion-filter.html>.
- [24] Zivid AS. *Gaussian Smoothing*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/processing-settings/gaussian-smoothing.html>.
- [25] Zivid AS. *Outlier Filter*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/processing-settings/outlier-filter.html>.
- [26] Zivid AS. *Reflection Filter*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/processing-settings/reflection-filter.html>.
- [27] Zivid AS. *Noise Filter*. URL: <https://support.zivid.com/v2.4/reference-articles/settings/processing-settings/noise-filter.html>.
- [28] *Understanding the importance of 3D hand-eye calibration*. URL: <https://blog.zivid.com/importance-of-3d-hand-eye-calibration>.
- [29] IBM Cloud Education. *Machine Learning*. URL: <https://www.ibm.com/cloud/learn/machine-learning>.
- [30] DeepAI. *What is a Neural Network?* URL: <https://deepai.org/machine-learning-glossary-and-terms/neural-network>.
- [31] Fitz AI. *Object Detection Guide*. URL: <https://www.fritz.ai/object-detection/#:~:text=Object%20detection%20is%20a%20computer,all%20while%20accurately%20labeling%20them..>
- [32] Elisha Odemakinde. *Mask R-CNN: A Beginner's Guide*. URL: <https://viso.ai/deep-learning/mask-r-cnn/#:~:text=Mask%20R%2DCNN%20is%20a,segmentation%20mask%20for%20each%20instance..>
- [33] Andrew Tao, Karan Sapra, and Bryan Catanzaro. “Hierarchical Multi-Scale Attention for Semantic Segmentation.” In: *CoRR* abs/2005.10821 (2020). arXiv: 2005.10821. URL: <https://arxiv.org/abs/2005.10821>.
- [34] K.-K. Maninis et al. “Deep Extreme Cut: From Extreme Points to Object Segmentation.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 616–625. DOI: 10.1109/CVPR.2018.00071.
- [35] *Camera Calibration and 3D Reconstruction*. URL: [https://docs.opencv.org/2.4/modules/calib3d/doc/camera\\_calibration\\_and\\_3d\\_reconstruction.html](https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html).
- [36] Sean Haggins. *Everything you need to know about point clouds | NavVis*. URL: <https://www.navvis.com/blog/everything-you-need-to-know-about-point-clouds-navvis>.
- [37] *ABB IRB 4000 Robot Series*. URL: <https://www.robots.com/robots/abb-irb-4400>.

# Appendix A

## Datasheet A

### A.1 detectron2Training.py

```
from logging import log
from detectron2.utils.events import TensorboardXWriter
import torch, torchvision
# Some basic setup:
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
# import some common libraries
import numpy as np
import os, json, cv2, random
import matplotlib.pyplot as plt
# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
from detectron2.data.datasets import register_coco_instances
from detectron2.engine import DefaultTrainer

def reg():
    global data_metadata
    register_coco_instances("data_train", {},
        "/home/ros/Detectron2/detectron2/cabels/dataset/coco_dat/coco_annotations.json",
        "/home/ros/Detectron2/detectron2/cabels/dataset/coco_dat")
    register_coco_instances("data_test", {},
        "/home/ros/Detectron2/detectron2/cabels/dataset/SynData/json/test.json",
        "/home/ros/Detectron2/detectron2/cabels/dataset/SynData/test")
    dataset_dicts = DatasetCatalog.get("data_train")
    data_metadata = MetadataCatalog.get("data_train")

def viz():
    global dataset_dicts
    dataset_dicts = DatasetCatalog.get("data_train")
    data_metadata = MetadataCatalog.get("data_train")
    data_metadata.ignore_label = 1
    for d in random.sample(dataset_dicts, 3):
        img = cv2.imread(d["file_name"])
        v = Visualizer(img[:, :, :-1], metadata=data_metadata, scale=0.5)
        v = v.draw_dataset_dict(d)
```

```

plt.figure(figsize = (14, 10))
plt.imshow(cv2.cvtColor(v.get_image()[:, :, :-1], cv2.COLOR_BGR2RGB))
plt.show()

def train():
    global cfg

    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file(
        "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
    cfg.DATASETS.TRAIN = ("data_train",)
    cfg.DATASETS.TEST = ()
    cfg.DATALOADER.NUM_WORKERS = 2
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("
        COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")# Let training ...
        initialize from model zoo
    cfg.SOLVER.IMS_PER_BATCH = 2
    cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
    cfg.SOLVER.MAX_ITER = 6000 # 300 iterations seems good enough for this ...
        toy dataset; you will need to train longer for a practical dataset
    cfg.SOLVER.STEPS = [] # do not decay learning rate
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 11 # only has one class (ballon). (see ...
        https://detectron2.readthedocs.io/tutorials/datasets.html#update-the-config-for-new
    # NOTE: this config means the number of classes, but a few popular ...
        unofficial tutorials incorrect uses num_classes+1 here.

    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
    trainer = DefaultTrainer(cfg)
    trainer.resume_or_load(resume=True)
    trainer.train()

def test():
    global predictor
    cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
    cfg.DATASETS.TEST = ("data_test", )
    predictor = DefaultPredictor(cfg)

def ShowRes():
    global data_metadata
    from detectron2.utils.visualizer import ColorMode
    dataset_dicts = DatasetCatalog.get("data_train")
    for d in random.sample(dataset_dicts, 5):
        im = cv2.imread(d["file_name"])
        outputs = predictor(im)
        v = Visualizer(im[:, :, :-1],
            metadata=data_metadata,
            scale=0.8,
            instance_mode=ColorMode.IMAGE_BW # remove the colors of unsegmented pixels
        )
        v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    plt.figure(figsize = (14, 10))
    plt.imshow(cv2.cvtColor(v.get_image()[:, :, :-1], cv2.COLOR_BGR2RGB))
    plt.show()

reg()
viz()
train()
test()
ShowRes()

```

## A.2 labelme2coco.py

```
import os
import argparse
import json

from labelme import utils
import numpy as np
import glob
import PIL.Image

class labelme2coco(object):
    def __init__(self, labelme_json=[], save_json_path="./coco.json"):
        """
        :param labelme_json: the list of all labelme json file paths
        :param save_json_path: the path to save new json
        """
        self.labelme_json = labelme_json
        self.save_json_path = save_json_path
        self.images = []
        self.categories = []
        self.annotations = []
        self.label = []
        self.annID = 1
        self.height = 0
        self.width = 0

        self.save_json()

    def data_transfer(self):
        for num, json_file in enumerate(self.labelme_json):
            with open(json_file, "r") as fp:
                data = json.load(fp)
                self.images.append(self.image(data, num))
                for shapes in data["shapes"]:
                    label = shapes["label"].split("_")
                    if label not in self.label:
                        self.label.append(label)
                    points = shapes["points"]
                    self.annotations.append(self.annotation(points, label, num))
                self.annID += 1

        # Sort all text labels so they are in the same order across data splits.
        self.label.sort()
        for label in self.label:
            self.categories.append(self.category(label))
        for annotation in self.annotations:
            annotation["category_id"] = self.getcatid(annotation["category_id"])

    def image(self, data, num):
        image = {}
        img = utils.img_b64_to_arr(data["imageData"])
        height, width = img.shape[:2]
        img = None
        image["height"] = height
        image["width"] = width
        image["id"] = num
        image["file_name"] = data["imagePath"].split("/")[-1]
```

```

self.height = height
self.width = width

return image

def category(self, label):
    category = {}
    category["supercategory"] = label[0]
    category["id"] = len(self.categories)
    category["name"] = label[0]
    return category

def annotation(self, points, label, num):
    annotation = {}
    contour = np.array(points)
    x = contour[:, 0]
    y = contour[:, 1]
    area = 0.5 * np.abs(np.dot(x, np.roll(y, 1)) - np.dot(y, np.roll(x, 1)))
    annotation["segmentation"] = [list(np.asarray(points).flatten())]
    annotation["iscrowd"] = 0
    annotation["area"] = area
    annotation["image_id"] = num

    annotation["bbox"] = list(map(float, self.getbbox(points)))

    annotation["category_id"] = label[0] # self.getcatid(label)
    annotation["id"] = self.annID
    return annotation

def getcatid(self, label):
    for category in self.categories:
        if label == category["name"]:
            return category["id"]
    print("label: {} not in categories: {}".format(label, self.categories))
    exit()
    return -1

def getbbox(self, points):
    polygons = points
    mask = self.polygons_to_mask([self.height, self.width], polygons)
    return self.mask2box(mask)

def mask2box(self, mask):
    index = np.argwhere(mask == 1)
    rows = index[:, 0]
    clos = index[:, 1]

    left_top_r = np.min(rows) # y
    left_top_c = np.min(clos) # x

    right_bottom_r = np.max(rows)
    right_bottom_c = np.max(clos)

    return [
        left_top_c,
        left_top_r,
        right_bottom_c - left_top_c,
        right_bottom_r - left_top_r,

```

```

]

def polygons_to_mask(self, img_shape, polygons):
    mask = np.zeros(img_shape, dtype=np.uint8)
    mask = PIL.Image.fromarray(mask)
    xy = list(map(tuple, polygons))
    PIL.ImageDraw.Draw(mask).polygon(xy=xy, outline=1, fill=1)
    mask = np.array(mask, dtype=bool)
    return mask

def data2coco(self):
    data_coco = {}
    data_coco["images"] = self.images
    data_coco["categories"] = self.categories
    data_coco["annotations"] = self.annotations
    return data_coco

def save_json(self):
    print("save coco json")
    self.data_transfer()
    self.data_coco = self.data2coco()

    print(self.save_json_path)
    os.makedirs(
        os.path.dirname(os.path.abspath(self.save_json_path)), exist_ok=True
    )
    json.dump(self.data_coco, open(self.save_json_path, "w"), indent=4)

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser(
        description="labelme annotation to coco data json file."
    )
    parser.add_argument(
        "labelme_images",
        help="Directory to labelme images and annotation json files.",
        type=str,
    )
    parser.add_argument(
        "--output", help="Output json file path.", default="trainval.json"
    )
    args = parser.parse_args()
    labelme_json = glob.glob(os.path.join(args.labelme_images, "*.json"))
    labelme2coco(labelme_json, args.output)

```

# Appendix B

## Datasheet B

```
import socket
import timeit
from datetime import datetime
import scipy.misc as sm
from collections import OrderedDict
import glob

# PyTorch includes
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.nn.functional import upsample

# Tensorboard include
from tensorboardX import SummaryWriter

# Custom includes
from dataloaders.combine_dbs import CombineDBs as combine_dbs
import dataloaders.pascal as pascal
import dataloaders.sbd as sbd
from dataloaders import custom_transforms as tr
import networks.deeplab_resnet as resnet
from layers.loss import class_balanced_cross_entropy_loss
from dataloaders.helpers import *

# Set gpu_id to -1 to run in CPU mode, otherwise set the id of the ...
corresponding gpu
gpu_id = 0
device = torch.device("cuda:"+str(gpu_id) if torch.cuda.is_available() ...
else "cpu")
if torch.cuda.is_available():
    print('Using GPU: {} '.format(gpu_id))

# Setting parameters
use_sbd = False
nEpochs = 10 # Number of epochs for training
resume_epoch = 0 # Default is 0, change if want to resume

p = OrderedDict() # Parameters to include in report
classifier = 'psp' # Head classifier to use
p['trainBatch'] = 1 # Training batch size
testBatch = 1 # Testing batch size
useTest = 1 # See evolution of the test set when training?
nTestInterval = 10 # Run on test set every nTestInterval epochs
```



```

snapshot = 20 # Store a model every snapshot epochs
relax_crop = 50 # Enlarge the bounding box by relax_crop pixels
nInputChannels = 4 # Number of input channels (RGB + heatmap of extreme ...
    points)
zero_pad_crop = True # Insert zero padding when cropping the image
p['nAveGrad'] = 1 # Average the gradient of several iterations
p['lr'] = 1e-8 # Learning rate
p['wd'] = 0.0005 # Weight decay
p['momentum'] = 0.9 # Momentum

# Results and model directories (a new directory is generated for every run)
save_dir_root = os.path.join(os.path.dirname(os.path.abspath(__file__)))
exp_name = os.path.dirname(os.path.abspath(__file__)).split('/')[ -1]
if resume_epoch == 0:
    runs = sorted(glob.glob(os.path.join(save_dir_root, 'run_*')))
    run_id = int(runs[-1].split('_')[ -1]) + 1 if runs else 0
else:
    run_id = 0
save_dir = os.path.join(save_dir_root, 'run_' + str(run_id))
if not os.path.exists(os.path.join(save_dir, 'models')):
    os.makedirs(os.path.join(save_dir, 'models'))

# Network definition
modelName = 'dextr_pascal'
net = resnet.resnet101(1, pretrained=False, nInputChannels=nInputChannels, ...
    classifier=classifier)
if resume_epoch == 0:
    print("Initializing from pretrained Deeplab-v2 model")
else:
    print("Initializing weights from: {}".format(
        os.path.join(save_dir, 'models', modelName + '_epoch-' + ...
            str(resume_epoch - 1) + '.pth')))
    net.load_state_dict(
        torch.load(os.path.join(save_dir, 'models', modelName + '_epoch-' ...
            + str(resume_epoch - 1) + '.pth'),
            map_location=lambda storage, loc: storage))
train_params = [{'params': resnet.get_1x_lr_params(net), 'lr': p['lr']},
    {'params': resnet.get_10x_lr_params(net), 'lr': p['lr'] * 10}]

net.to(device)

# Training the network
if resume_epoch != nEpochs:
    # Logging into Tensorboard
    log_dir = os.path.join(save_dir, 'models', ...
        datetime.now().strftime('%b%d_%H-%M-%S') + '_' + socket.gethostname())
    writer = SummaryWriter(log_dir=log_dir)

    # Use the following optimizer
    optimizer = optim.SGD(train_params, lr=p['lr'], ...
        momentum=p['momentum'], weight_decay=p['wd'])
    p['optimizer'] = str(optimizer)

# Preparation of the data loaders
composed_transforms_tr = transforms.Compose([
    tr.RandomHorizontalFlip(),
    tr.ScaleNRotate(rots=(-20, 20), scales=(.75, 1.25)),
    tr.CropFromMask(crop_elems=('image', 'gt'), relax=relax_crop, ...
        zero_pad=zero_pad_crop),
    tr.FixedResize(resolutions={'crop_image': (512, 512), 'crop_gt': ...
        (512, 512)}),

```

```

tr.ExtremePoints(sigma=10, pert=5, elem='crop_gt'),
tr.ToImage(norm_elem='extreme_points'),
tr.ConcatInputs(elems=('crop_image', 'extreme_points')),
tr.ToTensor())])
composed_transforms_ts = transforms.Compose([
tr.CropFromMask(crop_elems=('image', 'gt'), relax=relax_crop, ...
zero_pad=zero_pad_crop),
tr.FixedResize(resolutions={'crop_image': (512, 512), 'crop_gt': ...
(512, 512)}),
tr.ExtremePoints(sigma=10, pert=0, elem='crop_gt'),
tr.ToImage(norm_elem='extreme_points'),
tr.ConcatInputs(elems=('crop_image', 'extreme_points')),
tr.ToTensor())])

voc_train = pascal.VOCSegmentation(split='train', ...
transform=composed_transforms_tr)
voc_val = pascal.VOCSegmentation(split='val', ...
transform=composed_transforms_ts)

if use_sbd:
sbd = sbd.SBDSegmentation(split=['train', 'val'], ...
transform=composed_transforms_tr, rename=True)
db_train = combine_dbs([voc_train, sbd], excluded=[voc_val])
else:
db_train = voc_train

p['dataset_train'] = str(db_train)
p['transformations_train'] = [str(tran) for tran in ...
composed_transforms_tr.transforms]
p['dataset_test'] = str(db_train)
p['transformations_test'] = [str(tran) for tran in ...
composed_transforms_ts.transforms]

trainloader = DataLoader(db_train, batch_size=p['trainBatch'], ...
shuffle=True, num_workers=2)
testloader = DataLoader(voc_val, batch_size=testBatch, shuffle=False, ...
num_workers=2)

generate_param_report(os.path.join(save_dir, exp_name + '.txt'), p)

# Train variables
num_img_tr = len(trainloader)
num_img_ts = len(testloader)
running_loss_tr = 0.0
running_loss_ts = 0.0
aveGrad = 0
print("Training Network")
# Main Training and Testing Loop
for epoch in range(resume_epoch, nEpochs):
start_time = timeit.default_timer()

net.train()
for ii, sample_batched in enumerate(trainloader):

inputs, gts = sample_batched['concat'], sample_batched['crop_gt']

# Forward-Backward of the mini-batch
inputs.requires_grad_()
inputs, gts = inputs.to(device), gts.to(device)

output = net.forward(inputs)

```

```

output = upsample(output, size=(512, 512), mode='bilinear', ...
                  align_corners=True)

# Compute the losses, side outputs and fuse
loss = class_balanced_cross_entropy_loss(output, gts, ...
                                          size_average=False, batch_average=True)
running_loss_tr += loss.item()

# Print stuff
if ii % num_img_tr == num_img_tr - 1:
    running_loss_tr = running_loss_tr / num_img_tr
    writer.add_scalar('data/total_loss_epoch', ...
                    running_loss_tr, epoch)
    print('[Epoch: %d, numImages: %5d]' % (epoch, ...
        ii*p['trainBatch']+inputs.data.shape[0]))
    print('Loss: %f' % running_loss_tr)
    running_loss_tr = 0
    stop_time = timeit.default_timer()
    print("Execution time: " + str(stop_time - start_time)+"\n")

# Backward the averaged gradient
loss /= p['nAveGrad']
loss.backward()
aveGrad += 1

# Update the weights once in p['nAveGrad'] forward passes
if aveGrad % p['nAveGrad'] == 0:
    writer.add_scalar('data/total_loss_iter', loss.item(), ii ...
                    + num_img_tr * epoch)
    optimizer.step()
    optimizer.zero_grad()
    aveGrad = 0

# Save the model
if (epoch % snapshot) == snapshot - 1 and epoch != 0:
    torch.save(net.state_dict(), os.path.join(save_dir, 'models', ...
        modelName + '_epoch-' + str(epoch) + '.pth'))

# One testing epoch
if useTest and epoch % nTestInterval == (nTestInterval - 1):
    net.eval()
    with torch.no_grad():
        for ii, sample_batched in enumerate(testloader):
            inputs, gts = sample_batched['concat'], ...
                sample_batched['crop_gt']

            # Forward pass of the mini-batch
            inputs, gts = inputs.to(device), gts.to(device)

            output = net.forward(inputs)
            output = upsample(output, size=(512, 512), ...
                              mode='bilinear', align_corners=True)

            # Compute the losses, side outputs and fuse
            loss = class_balanced_cross_entropy_loss(output, gts, ...
                size_average=False)
            running_loss_ts += loss.item()

            # Print stuff
            if ii % num_img_ts == num_img_ts - 1:
                running_loss_ts = running_loss_ts / num_img_ts

```

```

        print('[Epoch: %d, numImages: %5d]' % (epoch, ...
            ii*testBatch+inputs.data.shape[0]))
        writer.add_scalar('data/test_loss_epoch', ...
            running_loss_ts, epoch)
        print('Loss: %f' % running_loss_ts)
        running_loss_ts = 0

    writer.close()

# Generate result of the validation images
net.eval()
composed_transforms_ts = transforms.Compose([
    tr.CropFromMask(crop_elems=('image', 'gt'), relax=relax_crop, ...
        zero_pad=zero_pad_crop),
    tr.FixedResize(resolutions={'gt': None, 'crop_image': (512, 512), ...
        'crop_gt': (512, 512)}),
    tr.ExtremePoints(sigma=10, pert=0, elem='crop_gt'),
    tr.ToImage(norm_elem='extreme_points'),
    tr.ConcatInputs(elems=('crop_image', 'extreme_points')),
    tr.ToTensor())])
db_test = pascal.VOCSegmentation(split='val', ...
    transform=composed_transforms_ts, rename=True)
testloader = DataLoader(db_test, batch_size=1, shuffle=False, num_workers=1)

save_dir_res = os.path.join(save_dir, 'Results')
if not os.path.exists(save_dir_res):
    os.makedirs(save_dir_res)

print('Testing Network')
with torch.no_grad():
    # Main Testing Loop
    for ii, sample_batched in enumerate(testloader):

        inputs, gts, metas = sample_batched['concat'], ...
            sample_batched['gt'], sample_batched['meta']

        # Forward of the mini-batch
        inputs = inputs.to(device)

        outputs = net.forward(inputs)
        outputs = upsample(outputs, size=(512, 512), mode='bilinear', ...
            align_corners=True)
        outputs = outputs.to(torch.device('cpu'))

        for jj in range(int(inputs.size()[0])):
            pred = np.transpose(outputs.data.numpy()[jj, :, :, :], (1, 2, 0))
            pred = 1 / (1 + np.exp(-pred))
            pred = np.squeeze(pred)
            gt = tens2image(gts[jj, :, :, :])
            bbox = get_bbox(gt, pad=relax_crop, zero_pad=zero_pad_crop)
            result = crop2fullmask(pred, bbox, gt, zero_pad=zero_pad_crop, ...
                relax=relax_crop)

            # Save the result, attention to the index jj
            sm.imwrite(os.path.join(save_dir_res, metas['image'][jj] + '-' ...
                + metas['object'][jj] + '.png'), result)

```

# Appendix C

## Datasheet C

### C.1 Train semantic segmentation Network

```
clear; clc; close all

%% Train A Semantic Segmentation Network

% Load the training data.
imageDir = 'Train_Images';
labelDir = 'Train_Label\PixelLabelData';

% Create an image datastore for the images.
imds = imageDatastore(imageDir);

% Create a pixelLabelDatastore for the ground truth pixel labels
classNames = [
    "Cable1"
    "cable2"
    "cable3"
    "cable4"
];
labelIDs = [1 2 3 4 0];

pxds = pixelLabelDatastore(labelDir,classNames,labelIDs)
% pxds = pixelLabelDatastore(gTruth);

augmenter = imageDataAugmenter('RandRotation',[0 90],'RandXReflection',true);

% Visualize training images and ground truth pixel labels.
I = read(imds);
C = read(pxds);

I = imresize(I,5);
L = imresize(uint8(C{1}),5);
imshowpair(I,L,'montage')

% Create a semantic segmentation network. This network uses a simple ...
    semantic segmentation network based on a downsampling and upsampling ...
```

```

    design.
numFilters = 64; %64;
filterSize = 3;
numClasses = 4; %2;

layers = [
    imageInputLayer([448 448 3])
    convolution2dLayer(filterSize,numFilters,'Padding',1)
    reluLayer()
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(filterSize,numFilters,'Padding',1)
    reluLayer()
    transposedConv2dLayer(4,numFilters,'Stride',2,'Cropping',1);
    convolution2dLayer(1,numClasses);
    softmaxLayer()
    pixelClassificationLayer()
];

% Setup training options.
opts = trainingOptions('sgdm', ...
    'InitialLearnRate',1e-3, ...
    'MaxEpochs',100, ... %100
    'MiniBatchSize',14); %64

% Combine the image and pixel label datastore for training.
trainingData = combine(imds,pxds);

% Train the network.
net = trainNetwork(trainingData,layers,opts);
%

% Read and display a test image.
testImage = imread('Bilder\Resized to 448x448\IMG20220519141500.jpg');
testImage = imread('Bilder4\aoK6XrJW_700w_0.jpg');
testImage = imread('NyeBilder\Resized to 448x448\IMG20220528184829.jpg');
subplot(1, 2, 1)
imshow(testImage)

Segment the test image and display the results.
C = semanticseg(testImage,net);
% C = semanticseg(testImage,trainedNetwork_2);

% B = labeloverlay(testImage,C);
B = labeloverlay(testImage,C, 'Colormap', [0.5 0.1 0; 0 0 1; 1 0 0; 0 1 0; ...
    0 0 0]);
subplot(1, 2, 2)
imshow(B)

%% Evaluate and Inspect the Results of Semantic Segmentation

% Define the location of the test images.
testImagesDir = 'Test_images';

```

```

% Create an imageDatastore object holding the test images.
imds = imageDatastore(testImagesDir);

% Define the location of the ground truth labels.
testLabelsDir = 'Test_Label\PixelLabelData';

% Define the class names and their associated label IDs. The label IDs are ...
% the pixel values used in the image files to represent each class.
classNames = ["Cable1" "cable2" "cable3" "cable4"];
labelIDs = [1 2 3 4];

% Create a pixelLabelDatastore object holding the ground truth pixel ...
% labels for the test images.
pxdsTruth = pixelLabelDatastore(testLabelsDir,classNames,labelIDs);

% trainingData = combine(imds,pxds);
TestData = combine(imds, pxdsTruth);

% Run the network on the test images. Predicted labels are written to disk ...
% in a temporary directory and returned as a pixelLabelDatastore object.
pxdsResults = semanticseg(imds,net,"WriteLocation",tempdir);
% pxdsResults = semanticseg(imds,trainedNetwork_1,"WriteLocation",tempdir);

% Evaluate the Quality of the Prediction
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTruth);

% Inspect Class Metrics
metrics.ClassMetrics

% Display the confusion matrix.
metrics.ConfusionMatrix

% Visualize the normalized confusion matrix as a confusion chart in a ...
% figure window.
cm = confusionchart(metrics.ConfusionMatrix.Variables, ...
    classNames, Normalization='row-normalized');
cm.Title = 'Normalized Confusion Matrix (%)';

% Visualize the histogram of the per-image intersection over union (IoU).
imageIoU = metrics.ImageMetrics.MeanIoU;
figure
histogram(imageIoU)
title('Image Mean IoU')

% Find the test image with the lowest IoU.
[minIoU, worstImageIndex] = min(imageIoU);
minIoU = minIoU(1);
worstImageIndex = worstImageIndex(1);

```

```

% Read the test image with the worst IoU, its ground truth labels, and its ...
    predicted labels for comparison.
worstTestImage = readimage(imds,worstImageIndex);
worstTrueLabels = readimage(pxdsTruth,worstImageIndex);
worstPredictedLabels = readimage(pxdsResults,worstImageIndex);

% Convert the label images to images that can be displayed in a figure window.
worstTrueLabelImage = im2uint8(worstTrueLabels == classNames(1));
worstPredictedLabelImage = im2uint8(worstPredictedLabels == classNames(1));

% Display the worst test image, the ground truth, and the prediction.
worstMontage = ...
    cat(3,worstTestImage,worstTrueLabelImage,worstPredictedLabelImage);
worstMontage = imresize(worstMontage,4,"nearest");
figure
% montage(worstMontage,'Size',[1 3])
montage(worstMontage,'Size',[1 3])
title(['Test Image vs. Truth vs. Prediction. IoU = ' num2str(minIoU)])

% Similarly, find the test image with the highest IoU.
[maxIoU, bestImageIndex] = max(imageIoU);
maxIoU = maxIoU(1);
bestImageIndex = bestImageIndex(1);

% Repeat the previous steps to read, convert, and display the test image ...
    with the best IoU with its ground truth and predicted labels.
bestTestImage = readimage(imds,bestImageIndex);
bestTrueLabels = readimage(pxdsTruth,bestImageIndex);
bestPredictedLabels = readimage(pxdsResults,bestImageIndex);

bestTrueLabelImage = im2uint8(bestTrueLabels == classNames(1));
bestPredictedLabelImage = im2uint8(bestPredictedLabels == classNames(1));

bestMontage = cat(3,bestTestImage,bestTrueLabelImage,bestPredictedLabelImage);
bestMontage = imresize(bestMontage,4,"nearest");
figure
montage(bestMontage,'Size',[1 3])
title(['Test Image vs. Truth vs. Prediction. IoU = ' num2str(maxIoU)])

% Define the metrics to compute.
evaluationMetrics = ["accuracy" "iou"];

% Compute these metrics for the triangleImages test data set.
metrics = ...
    evaluateSemanticSegmentation(pxdsResults,pxdsTruth,"Metrics",evaluationMetrics);

% Display the chosen metrics for each class.
metrics.ClassMetrics

```



## C.2 Change image size

```
% Resizes images to a size of 224x224, which AlexNet needs, and saves the ...
    resized images in a "Resized to 224x224" subfolder of the images folder.
% Image Analyst, March 21, 2020.
clc; % Clear the command window.
fprintf('Beginning to run %s.m.\n', mfilename);
close all; % Close all figures (except those of imtool.)
clear; % Erase all existing variables. Or clearvars if you want.
workspace; % Make sure the workspace panel is showing.
format long g;
format compact;
fontSize = 15;
% Specify the folder where the files live.
inputImagesFolder = 'ToBilderTil'; % Change it to whatever you need, if ...
    you want/need to.
% Check to make sure that folder actually exists. Warn user if it doesn't.
if ~isfolder(inputImagesFolder)
    errorMessage = sprintf('Error: The following folder does not ...
        exist:\n\n%s\n\nPlease specify to an existing folder.', ...
        inputImagesFolder);
    uiwait(warndlg(errorMessage));
    % Try to find the highest level folder in that path that DOES exist.
    while ~isfolder(inputImagesFolder) && length(inputImagesFolder) > 3
        [inputImagesFolder, ~, ~] = fileparts(inputImagesFolder);
    end
    % Should have a good starting folder now.
    inputImagesFolder = uigetdir(inputImagesFolder);
    if inputImagesFolder == 0
        return;
    end
end
% Create output folder
outputImagesFolder = fullfile(inputImagesFolder, '/Resized to 448x448');
if ~isfolder(outputImagesFolder)
    mkdir(outputImagesFolder);
end
% Get a list of all files in the folder with the desired file name pattern.
filePattern = fullfile(inputImagesFolder, '*.jpg'); % Change to whatever ...
    pattern you need.
theFiles = dir(filePattern)
numFiles = length(theFiles);
hFig = figure;
hFig.WindowState = 'maximized';
for k = 1 : numFiles
    % Get the input file name.
    baseFileName = theFiles(k).name;
    fullFileName = fullfile(inputImagesFolder, baseFileName);
    % Get the output file name.
    outputFullFileName = fullfile(outputImagesFolder, baseFileName);
    fprintf(1, 'Now reading %d of %d "%s"\n', k, numFiles, fullFileName);

    % Read in input image with imread().
    inputImage = imread(fullFileName);
    % Resize it.
    outputImage = imresize(inputImage, [448, 448]);

    % Display input and output images.
    subplot(1, 2, 1);
```

```

imshow(inputImage); % Display image.
caption = sprintf('Input image (%d of %d):\n"%s", %d by %d', k, ...
    numFiles, baseFileName, size(inputImage, 1), size(inputImage, 2));
title(caption, 'FontSize', fontSize, 'Interpreter', 'none');
subplot(1, 2, 2);
imshow(outputImage); % Display image.
caption = sprintf('Output image: "%s", %d by %d', baseFileName, ...
    size(outputImage, 1), size(outputImage, 2));
title(caption, 'FontSize', fontSize, 'Interpreter', 'none');
drawnow; % Force display to update immediately.

% Write out the resized output file to the output folder.
imwrite(outputImage, outputFullFileName);
end
fprintf('Done running %s.m.\n', mfilename);
% Open the output folder in File Explorer.
winopen(outputImagesFolder);

```

Appendix D

Datasheet D

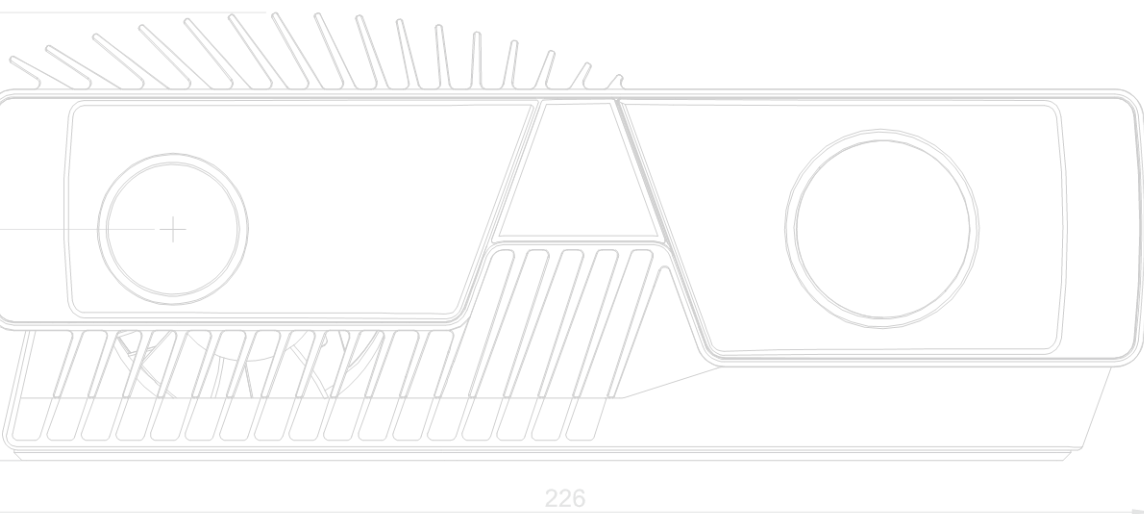
# Zivid One<sup>+</sup>

## Technical specification

Zivid One+ S (ZVD1P-S)

Zivid One+ M (ZVD1P-M)

Zivid One+ L (ZVD1P-L)



# Table of Contents

- Table of Contents..... 2
- General specifications..... 3
- Operating distance and field of view..... 4
- Accuracy specifications ..... 8
  - Common conditions.....8
  - Zivid One+ S Typical Specifications ..... 9
  - Zivid One+ M Typical Specifications ..... 12
  - Zivid One+ L Typical Specifications ..... 15
- Physical specifications..... 18
- Mechanical drawings ..... 19
  - Connectors**.....20
- Revision history ..... 21

# General specifications

Model (Part number)	Zivid One+ S (ZVD1P-S) Zivid One+ M (ZVD1P-M) Zivid One+ L (ZVD1P-L)
3D technology	Structured light
Imaging	1920 x 1200 (2.3Mpixel) Native 3D Color
Point cloud output	3D (XYZ) + Color (RGB) + SNR
Exposure time (minimum per pattern projection)	6.500 ms
Aperture (A)	f/1.4 to f/32
Gain (G)	1x to 16x
Projector Brightness (B)	0.25x to 1.8x 1x = 400 lumens
Calibration	Factory calibrated
Safety and EMC	CE CB EN60950 FCC Class A
Typical capture time <sup>1</sup>	100 ms to 1 s

<sup>1</sup> From capture initialized until point cloud is ready to copy. Includes processing. Acquisition time can be shorter.

## Operating distance and field of view

	<b>S</b>	<b>M</b>	<b>L</b>
Focus distance (mm)	500	1000	1800
Optimal working distance (mm)	350 to 700	700 to 1500	1200 to 2600
Recommended working distance (mm)	300 to 1000	500 to 2000	1200 to 3000
Field of view (mm)	164 x 132 at 300 350 x 220 at 500 621 x 439 at 1000	433 x 271 at 600 702 x 432 at 1000 1330 x 871 at 2000	843 x 530 at 1200 1252 x 783 at 1800 2069 x 1310 at 3000
Spatial resolution (mm)	0.18 at 500 $4.00 \times 10^{-4}$ per distance (z) in mm	0.37 at 1000 $3.71 \times 10^{-4}$ per distance (z) in mm	0.67 at 1800 $3.67 \times 10^{-4}$ per distance (z) in mm

Figure 1 - Zivid One+ S FOV

All values in degrees or mm.

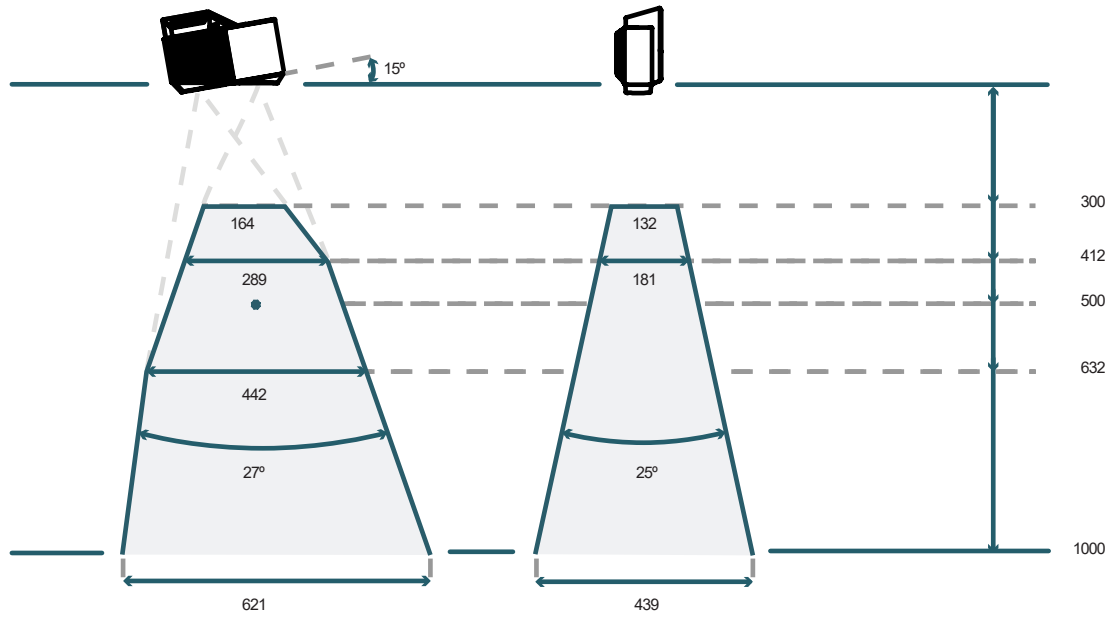


FIGURE 2 - ZIVID ONE+ S SPATIAL RESOLUTION VS. DISTANCE

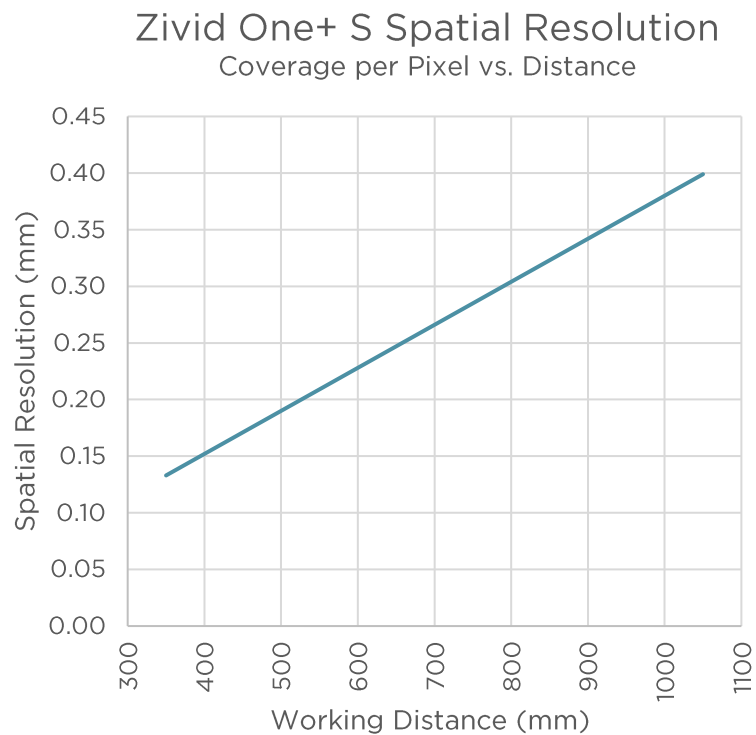




Figure 3 - Zivid One+ M FOV

All values in degrees or mm.

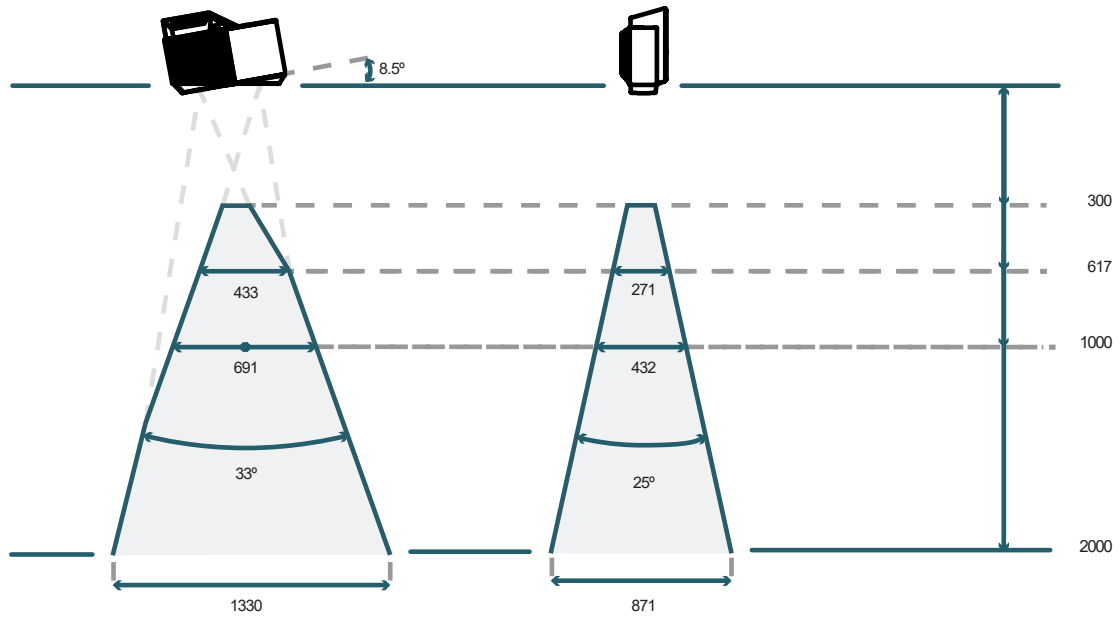


FIGURE 4 - ZIVID ONE+ M SPATIAL RESOLUTION VS. DISTANCE

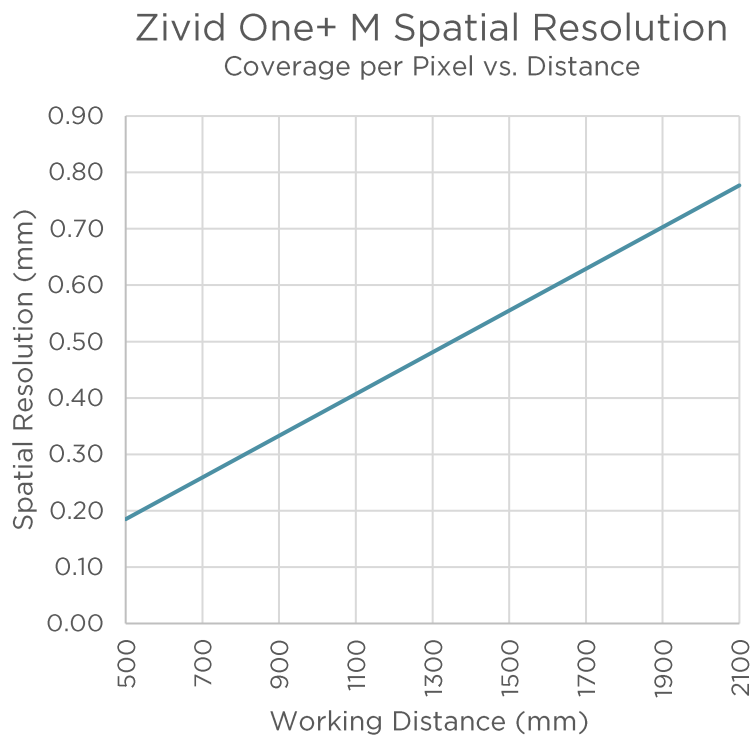


Figure 5 - Zivid One+ L FOV

All values in degrees or mm.

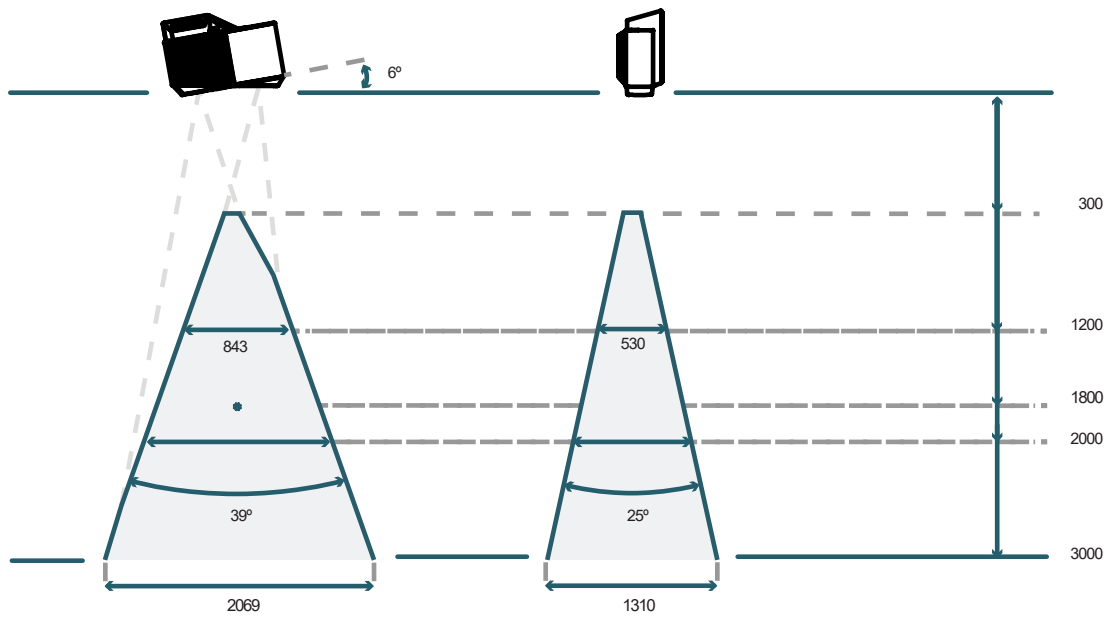
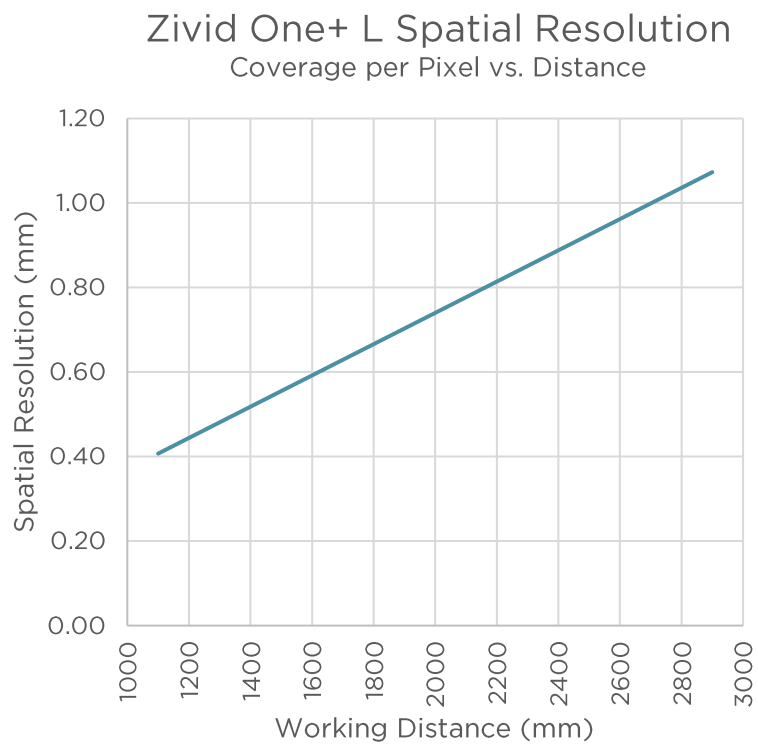


FIGURE 6 - ZIVID ONE+ L SPATIAL RESOLUTION VS. DISTANCE



# Accuracy specifications

## Common conditions

The following table outlines the conditions applied under test and to all specifications unless otherwise stated.

Parameter	Description	Typical
Working distance (D)		Zivid One+ S: 500 mm
	Focus distance	Zivid One+ M: 1000 mm
		Zivid One+ L: 1800 mm
Working distance (D)		Zivid One+ S: 350 - 700 mm
	Optimal working distance	Zivid One+ M: 700 - 1500 mm
		Zivid One+ L: 1300 - 2600 mm
Ambient temperature (Ta)	Typical temperature	15 - 30 °C
	Full temperature range	10 - 40 °C
Ambient light (La)		0 lux
Aperture (A)		f/8.0 - f/2.0
Gain (G)		1.0x
Projector Brightness (B)		1 - 1.8 x
Capture time	Acquisition time used during measurement	> 85 ms
	Capture time used during measurement	> 200 ms
Duty Cycle	Capture-to-Idle time ratio	5 - 30 %
Other		81% center crop (90% × 90%)
		HDR = off
		10 min warm-up Applied in-field correction

## Zivid One+ S Typical Specifications

Typical numbers are given at common conditions unless otherwise specified.

Property	Description	Typical
Warm-up time	Minimum recommended time needed for camera to stabilize from an idle state assuming capturing at a constant rate.  Some trueness changes may be experienced during warm-up phase.	10 minutes
Point precision	$1\sigma$ Euclidian distance variation for a point between consecutive measurements at focus distance, D. <sup>2</sup>	25 $\mu\text{m}$
Local Planarity Precision	$1\sigma$ Euclidian distance variation from a plane for a set of points within a smaller local region at focus distance, D. <sup>2</sup>	40 $\mu\text{m}$
Global Planarity Trueness	Average deviation from a plane in field of view at focus distance, D.	< 100 $\mu\text{m}$
Dimension Trueness	70-percentile dimension error in field of view at focus distance, D, and typical temperature range.	< 0.15 %
	70-percentile dimension error in field of view within optimal working distance and typical temperature range.	< 0.20 %
	70-percentile dimension error in field of view within optimal working distance and full temperature range.	< 0.30 %

<sup>2</sup> Measured with Gaussian filter disabled.

FIGURE 7 - TYPICAL ZIVID ONE+ S POINT PRECISION VS. DISTANCE

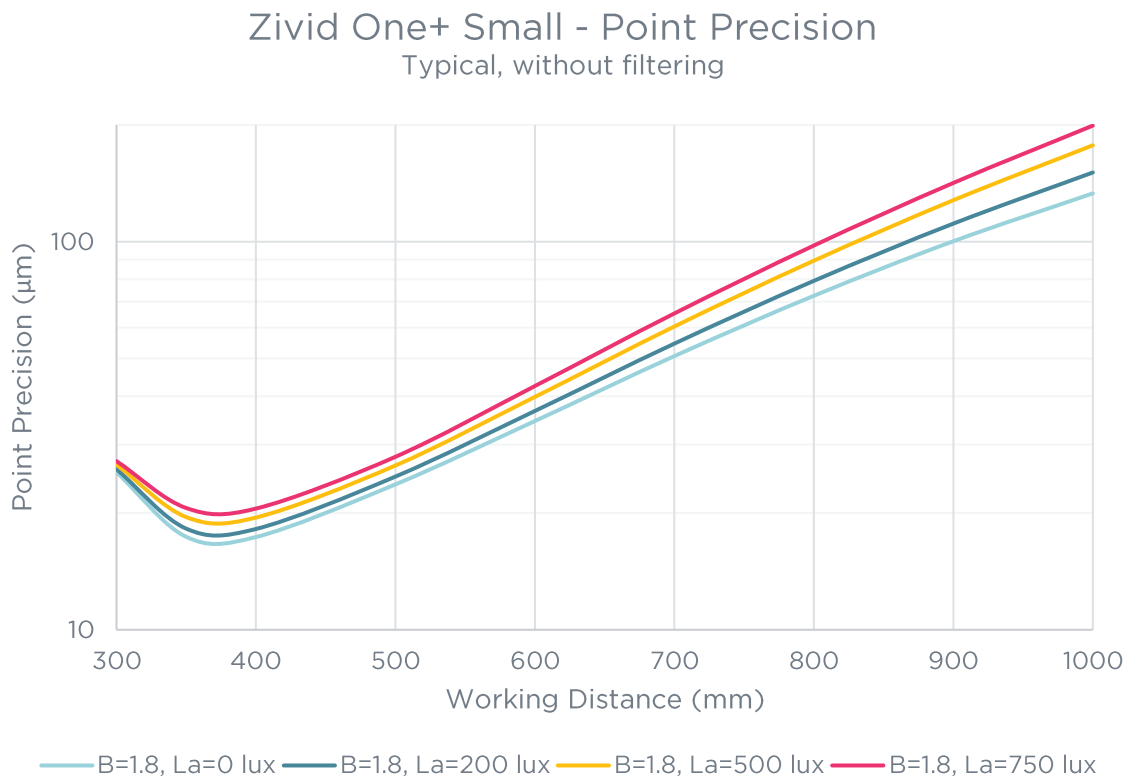


FIGURE 8 - TYPICAL ZIVID ONE+ S LOCAL PLANARITY PRECISION VS. DISTANCE

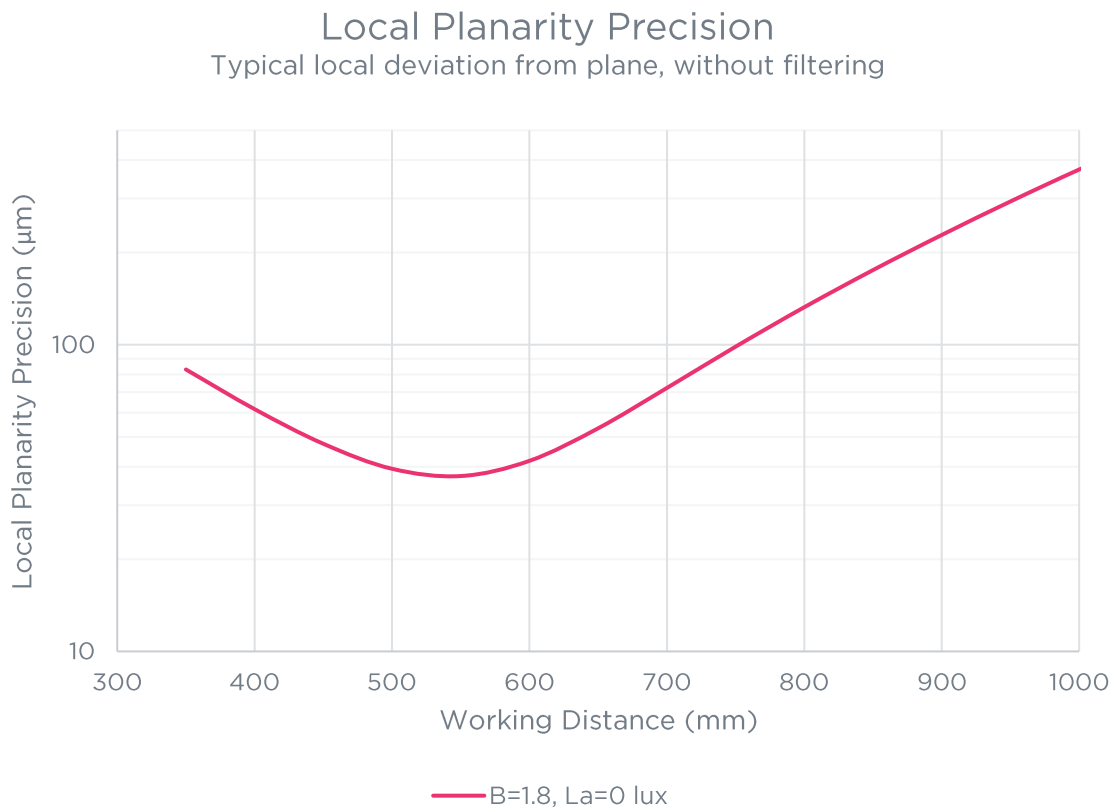


FIGURE 9 – TYPICAL ZIVID ONE+ S GLOBAL PLANARITY TRUENESS VS. DISTANCE

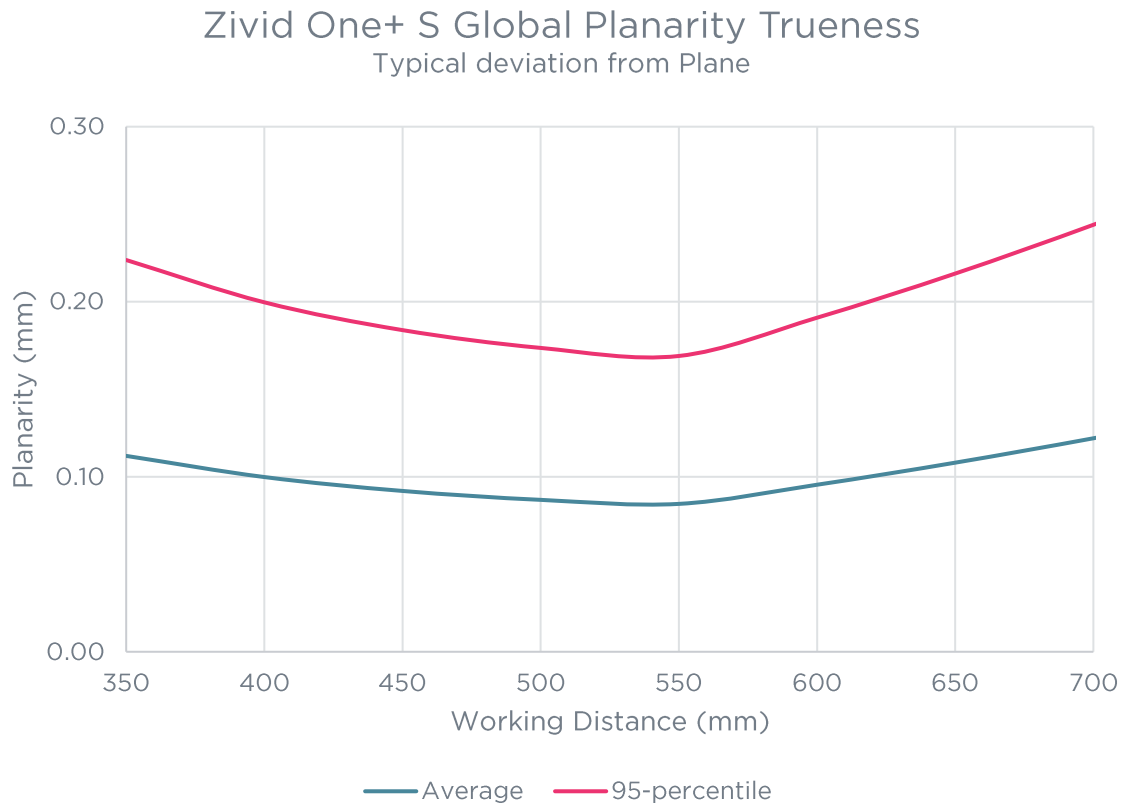
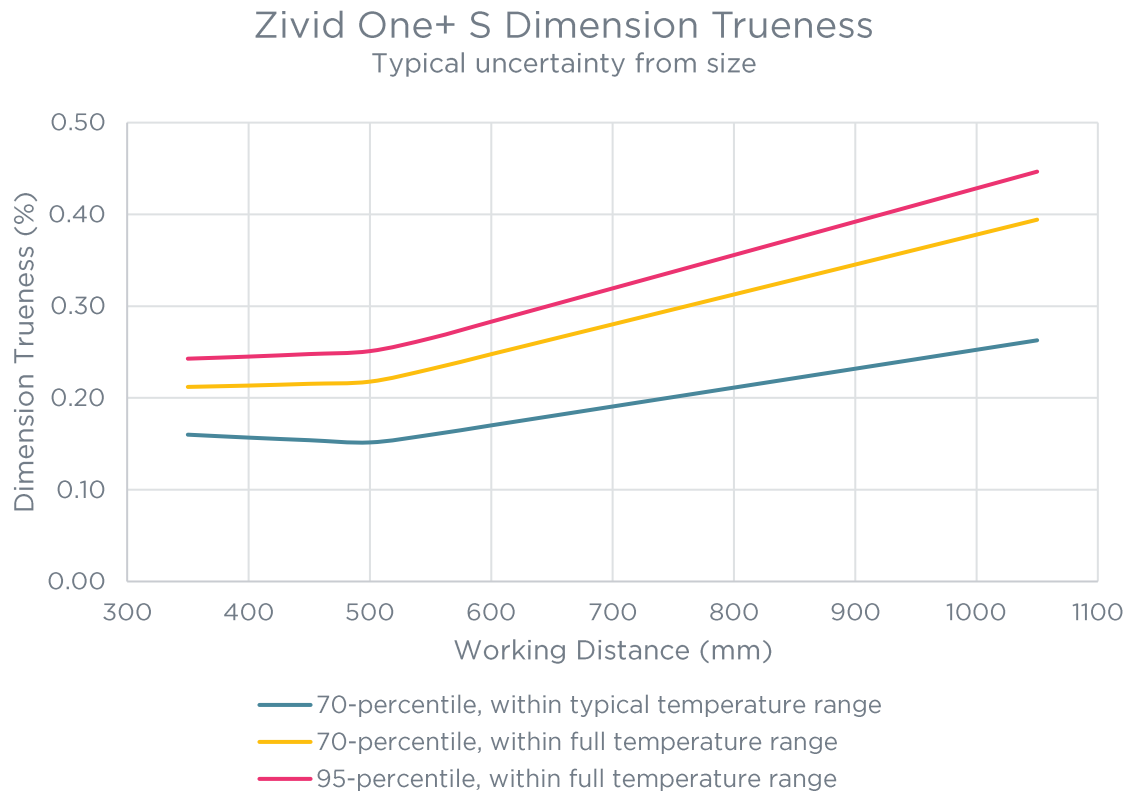


FIGURE 10 – TYPICAL ZIVID ONE+ S DIMENSION TRUENESS VS. DISTANCE



## Zivid One+ M Typical Specifications

Typical numbers are given at common conditions unless otherwise specified.

Property	Description	Typical
Warm-up time	Minimum recommended time needed for camera to stabilize from an idle state assuming capturing at a constant rate.	10 minutes
	Some trueness changes may be experienced during warm-up phase.	
Point precision	$1\sigma$ Euclidian distance variation for a point between consecutive measurements at focus distance, D. <sup>3</sup>	110 $\mu\text{m}$
Local Planarity Precision	$1\sigma$ Euclidian distance variation from a plane for a set of points within a smaller local region at focus distance, D. <sup>3</sup>	190 $\mu\text{m}$
Global Planarity Trueness	Average deviation from a plane in field of view at focus distance, D.	< 100 $\mu\text{m}$
Dimension Trueness	70-percentile dimension error in field of view at focus distance, D, and typical temperature range.	< 0.30 %
	70-percentile dimension error in field of view within optimal working distance and typical temperature range.	< 0.40 %
	70-percentile dimension error in field of view within optimal working distance and full temperature range.	< 0.50 %

<sup>3</sup> Measured with Gaussian filter disabled.

FIGURE 11 – TYPICAL ZIVID ONE+ M POINT PRECISION VS. DISTANCE

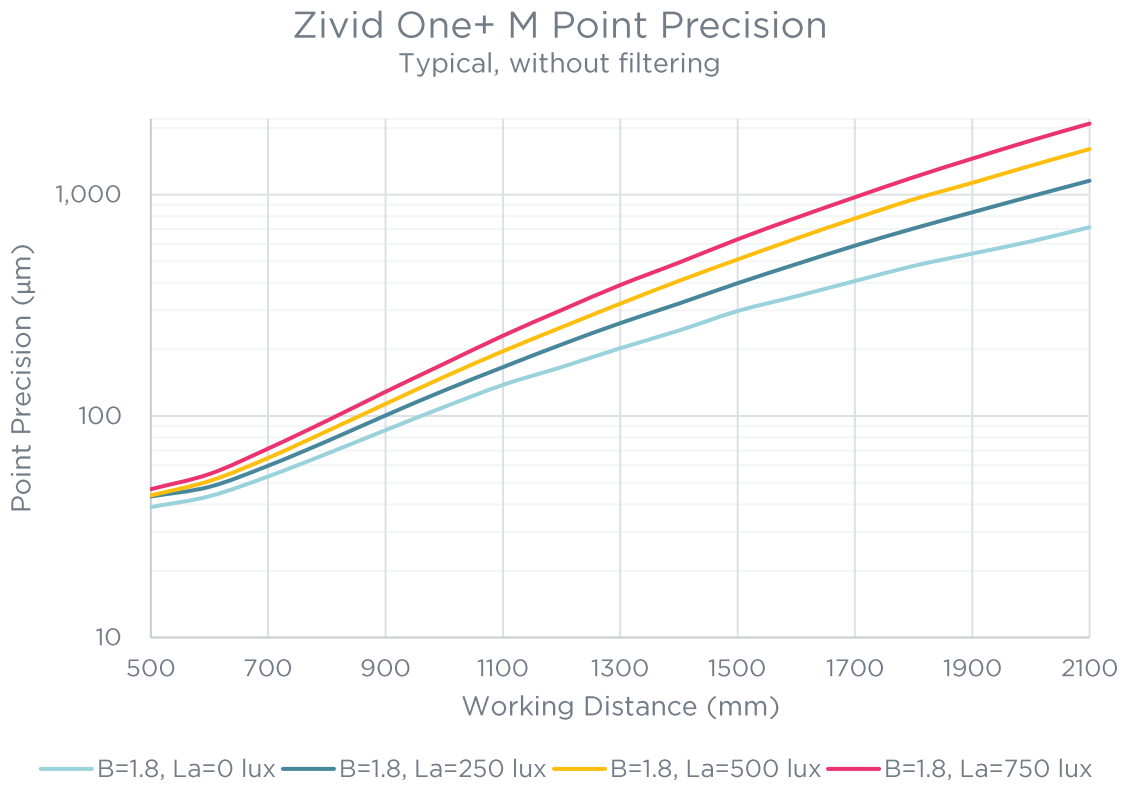


FIGURE 12 – TYPICAL ZIVID ONE+ M LOCAL PLANARITY PRECISION VS. DISTANCE

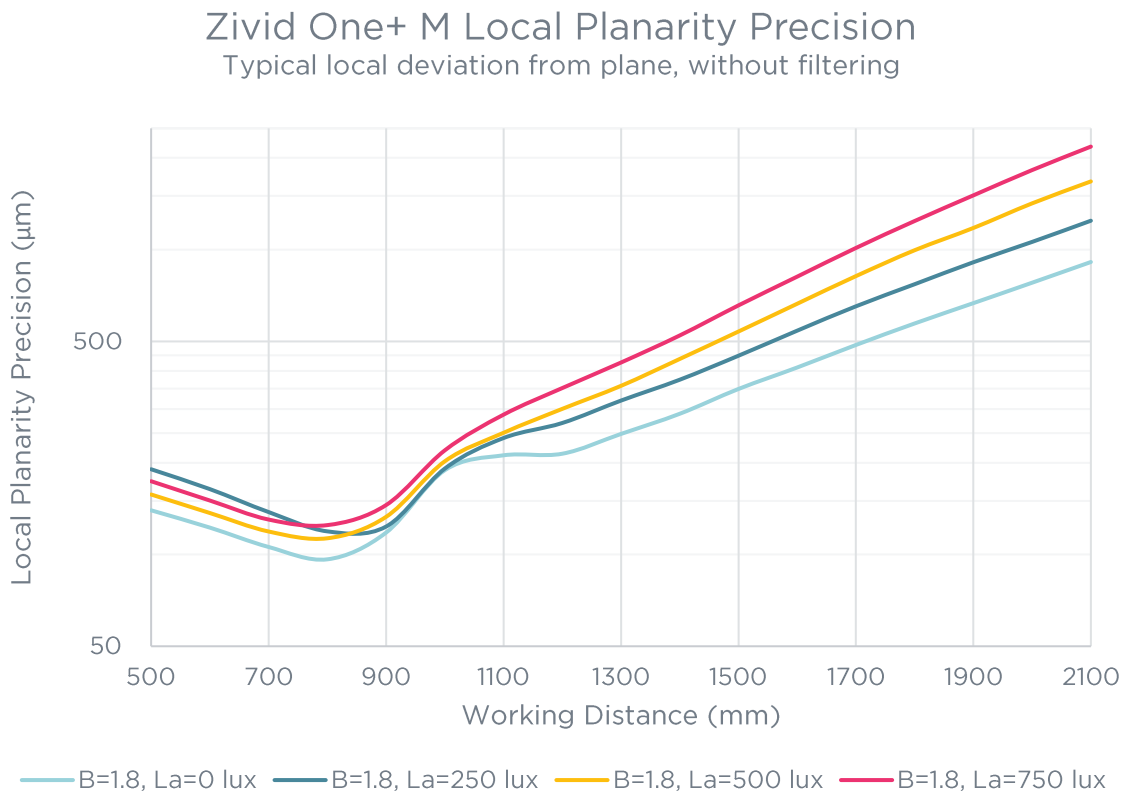




FIGURE 13 – TYPICAL ZIVID ONE+ M GLOBAL PLANARITY TRUENESS VS. DISTANCE

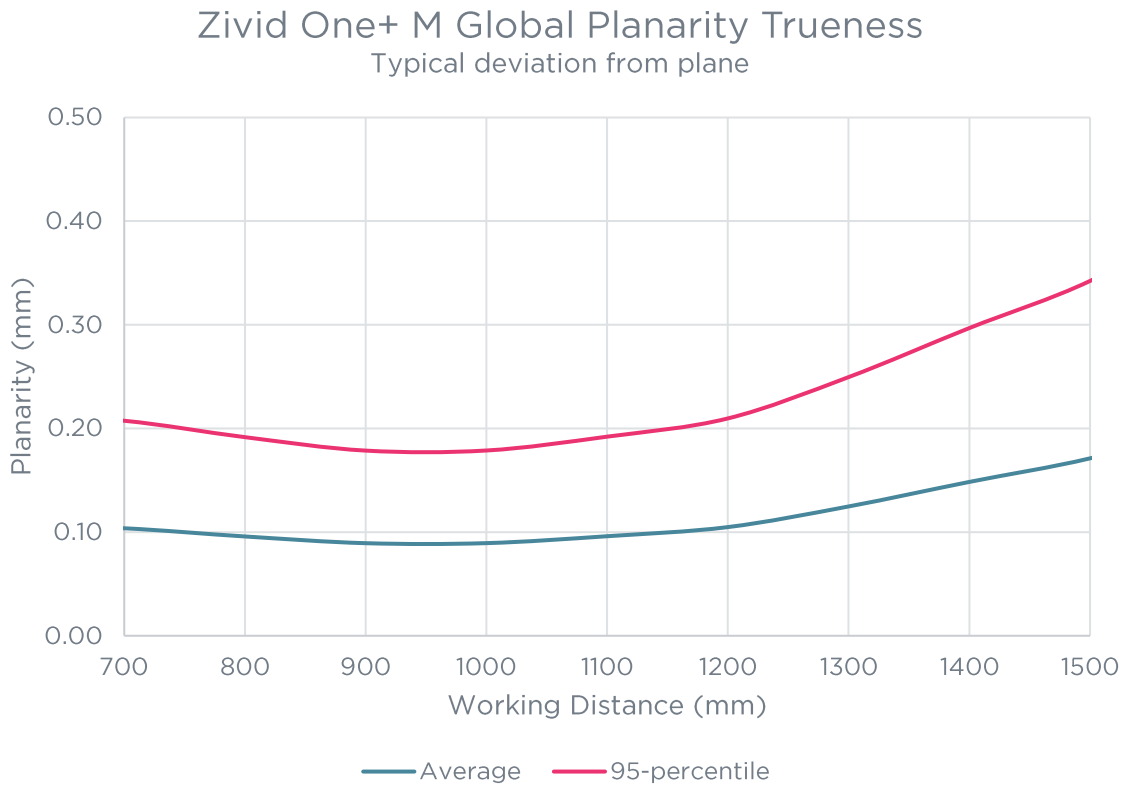
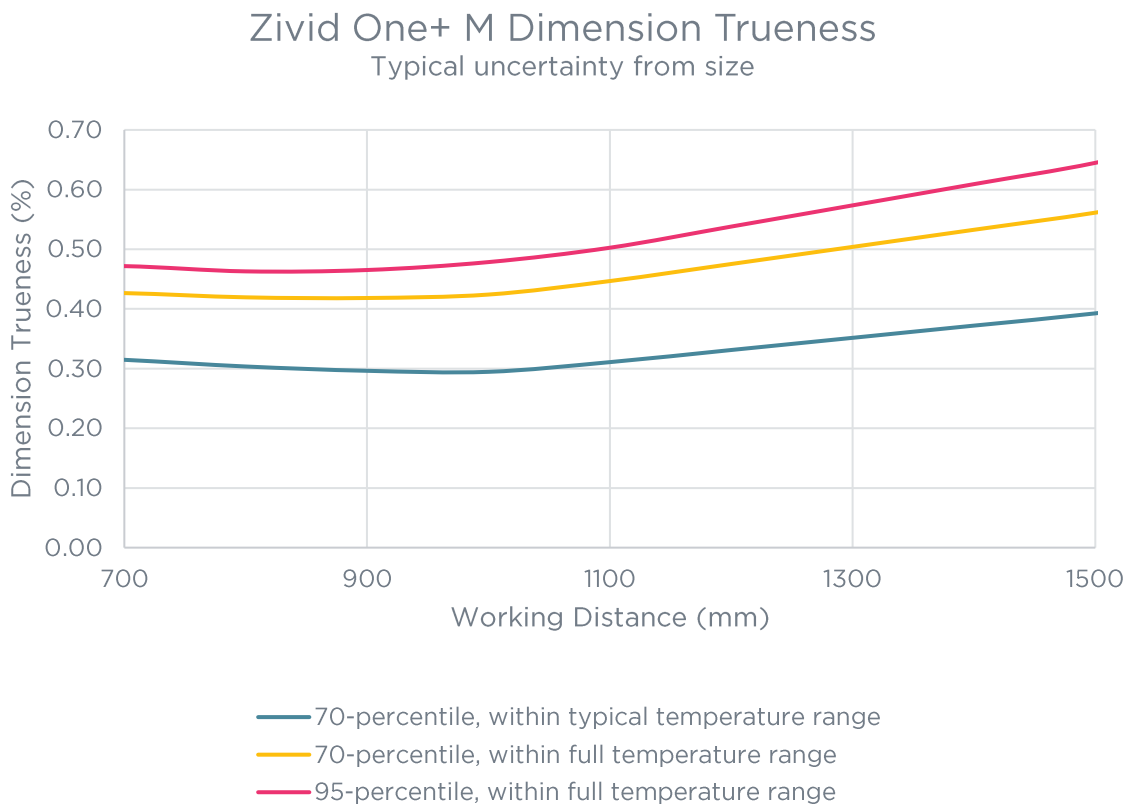


FIGURE 14 – TYPICAL ZIVID ONE+ M DIMENSION TRUENESS VS. DISTANCE



## Zivid One+ L Typical Specifications

Typical numbers are given at common conditions unless otherwise specified.

Property	Description	Typical
Warm-up time	Minimum recommended time needed for camera to stabilize from an idle state assuming capturing at a constant rate.  Some trueness changes may be experienced during warm-up phase.	10 minutes
Point precision	1 $\sigma$ Euclidian distance variation for a point between consecutive measurements at focus distance, D. <sup>4</sup>	350 $\mu$ m
Local Planarity Precision	1 $\sigma$ Euclidian distance variation from a plane for a set of points within a smaller local region at focus distance, D. <sup>4</sup>	700 $\mu$ m
Global Planarity Trueness	Average deviation from a plane in field of view at focus distance, D.	< 350 $\mu$ m
Dimension Trueness	70-percentile dimension error in field of view at focus distance, D, and typical temperature range.	< 0.50 %
	70-percentile dimension error in field of view within optimal working distance and typical temperature range.	< 0.60 %
	70-percentile dimension error in field of view within optimal working distance and full temperature range.	< 0.70 %

<sup>4</sup> Measured with Gaussian filter disabled.

FIGURE 15 – TYPICAL ZIVID ONE+ L POINT PRECISION VS. DISTANCE

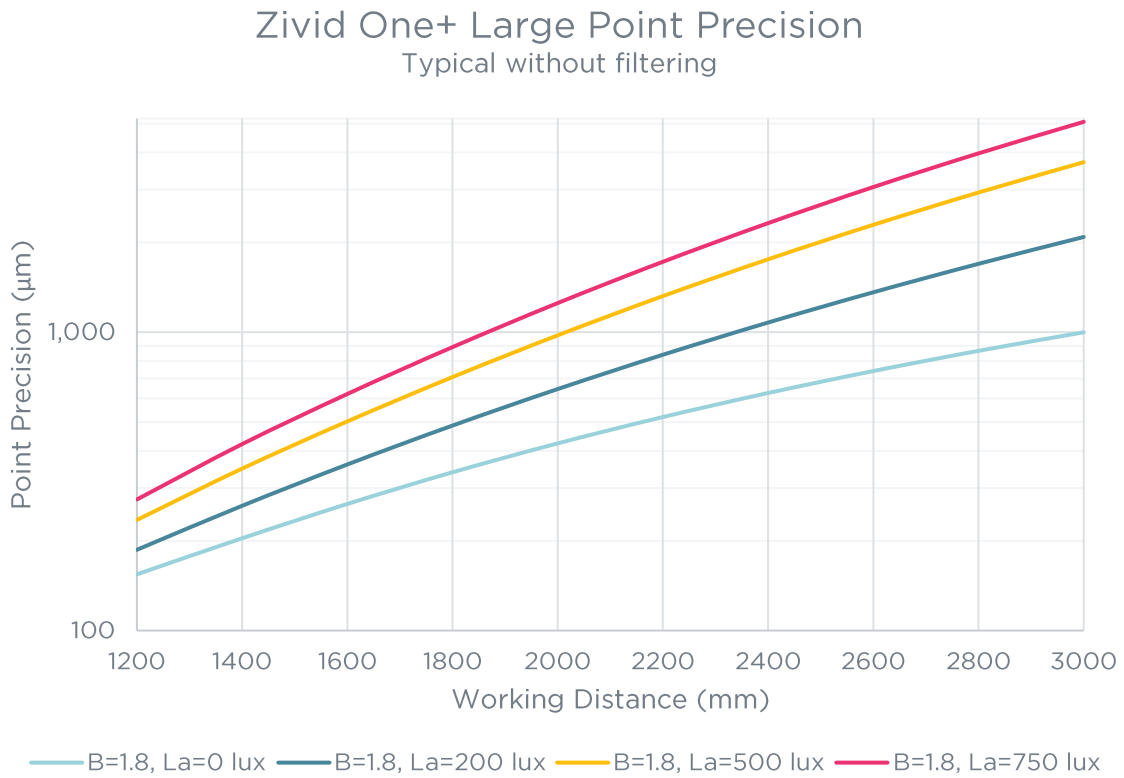


FIGURE 16 - TYPICAL ZIVID ONE+ L LOCAL PLANARITY PRECISION VS. DISTANCE

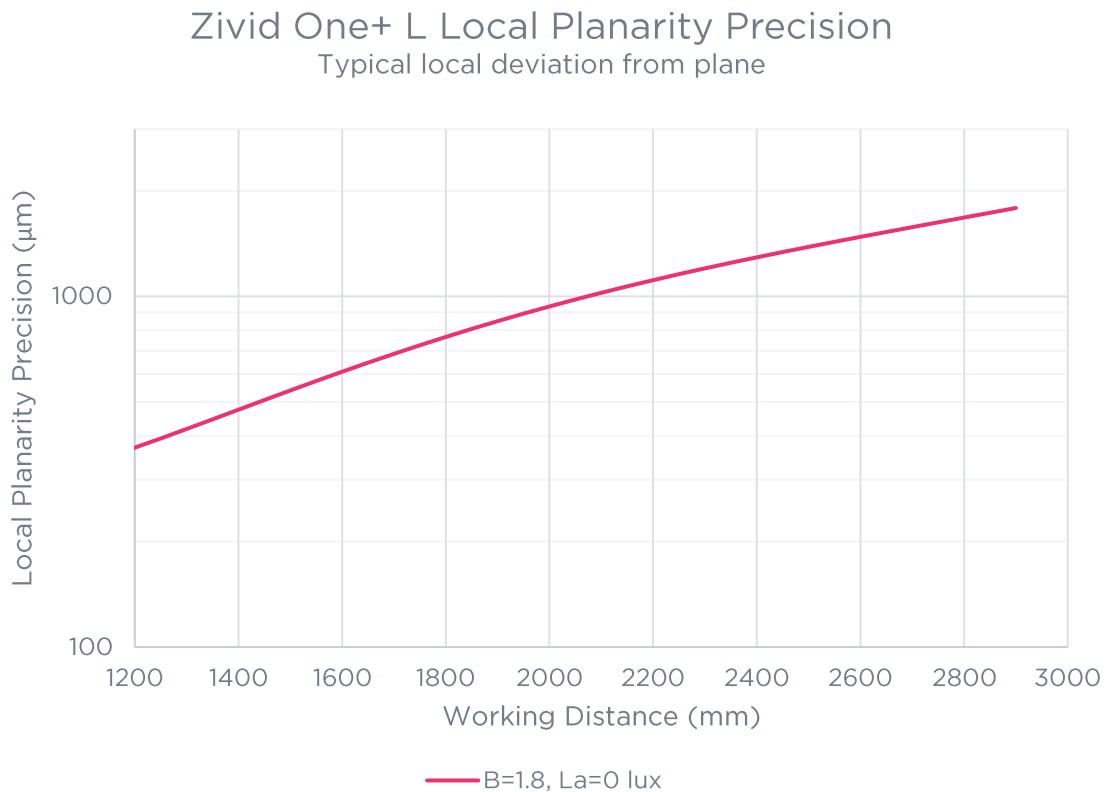


FIGURE 17 – TYPICAL ZIVID ONE+ L GLOBAL PLANARITY TRUENESS VS. DISTANCE

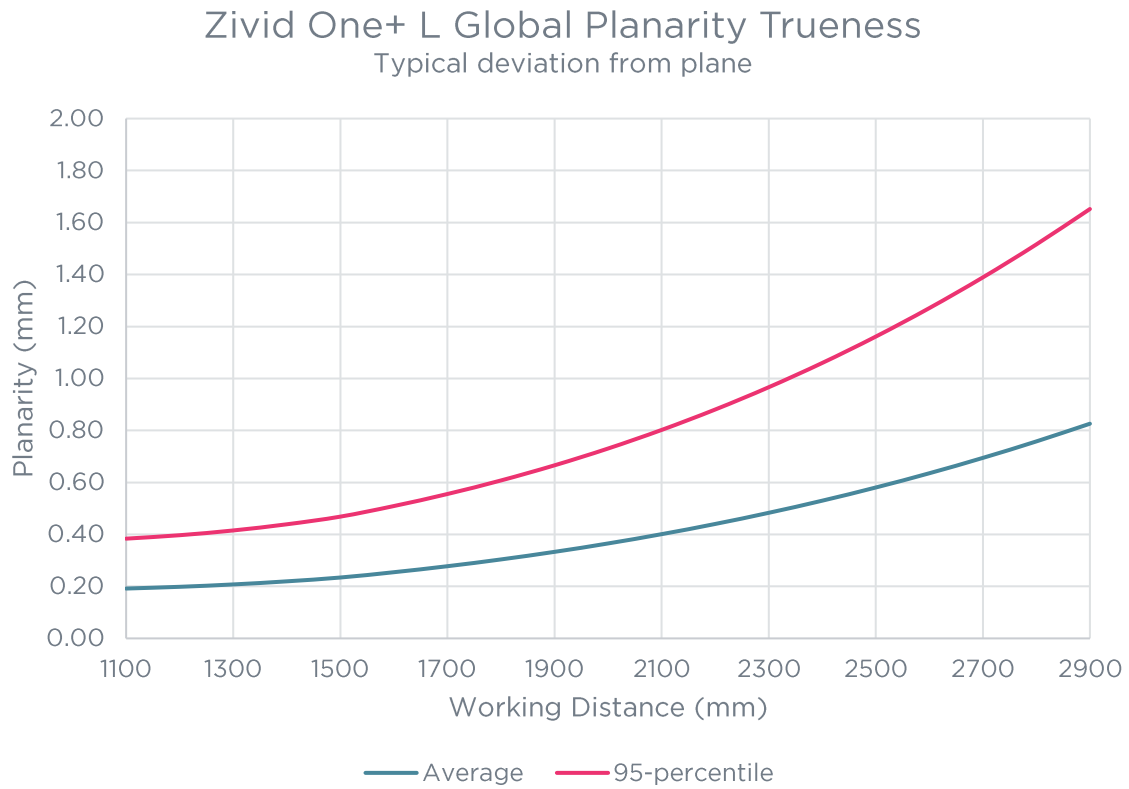
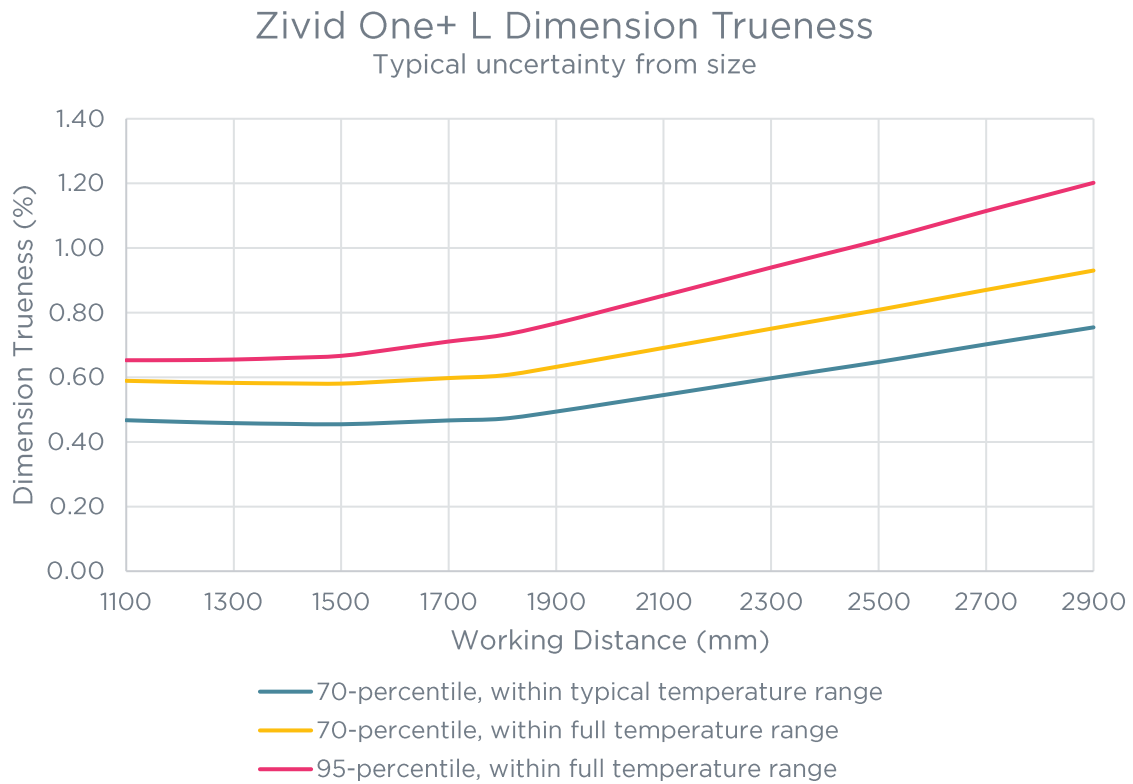


FIGURE 18 – TYPICAL ZIVID ONE+ L DIMENSION TRUENESS VS. DISTANCE



# Physical specifications

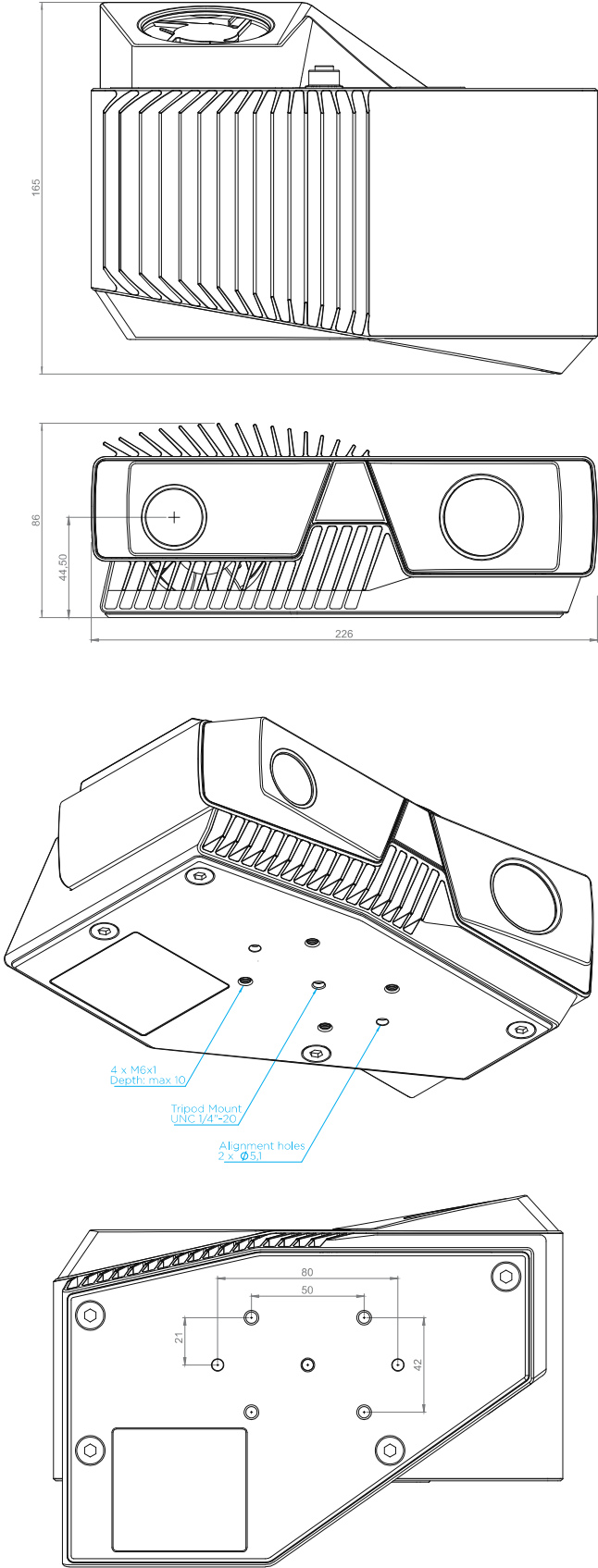
Size	226 mm x 165 mm x 86 mm
Weight	2 kg
Cable strain limit, power	90 N
Cable strain limit, data	30 N
Environmental	IP65 5 g sinusoidal <sup>5</sup> 15 g shock <sup>6</sup>
Operating temperature	10° to 40° C
Storage temperature	-20° to 60° C
Data connector	USB 3.0 SuperSpeed USB Type B Jack screw M2
Power connector	M12-5
Power adapter	24V = 5A EU, US, and UK power plug options
Power consumption, typical	15 W, Idle 45 W, TDP <sup>7</sup> 120 W, Peak

<sup>5</sup> IEC 60068-2-6, 10-150 Hz, 5 g, in X, Y and Z direction, 2 hour per axis. Sweep rate 1 octave per minute sweep rate.

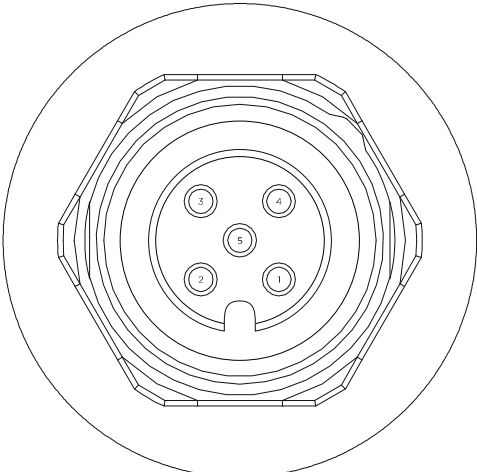
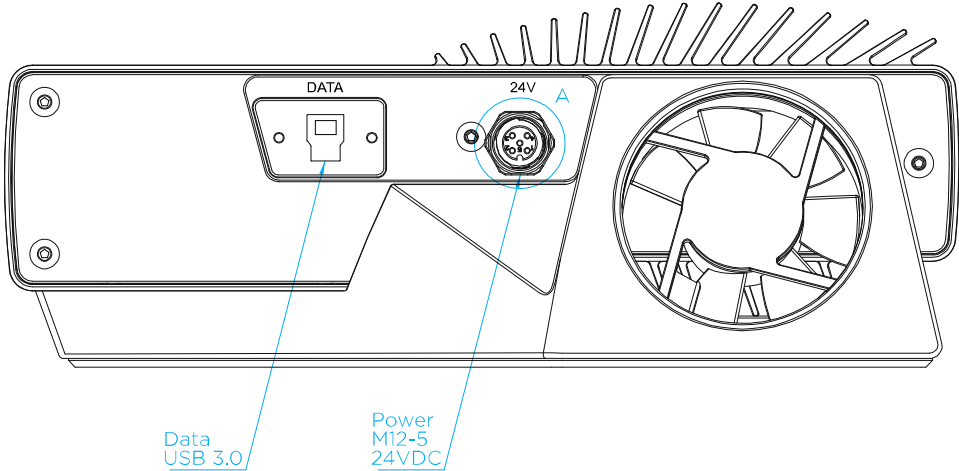
<sup>6</sup> IEC 60068-2-27, 15 g / 11 ms half sine shock pulses. 3 shocks per direction, 18 shocks in total.

<sup>7</sup> Thermal Design Power is the maximum power consumed by the camera when capturing 3D images in a continuous stream.

# Mechanical drawings



# Connectors



DETAIL A  
Power Inlet

### Pin

1	24V DC +/- 20% Max 4A
2	24V DC +/- 20% Max 4A
3	GND
4	GND
5	NC

# Revision history

Ver.	Date	Notes
1.0	05/21	Updated front page Updated table of contents Updated "General specifications" and added valid revision number Updated "Operating distance and field of view" Added figure 2-16 Updated "Accuracy specifications" and table "Common conditions" Updated table "Zivid One+ S Typical Specifications" Updated table "Zivid One+ M Typical Specifications" Updated table "Zivid One+ L Typical Specifications" Updated table "Physical specifications"
0.91	05/19	Added metrics for Zivid One+ Small. Added metrics for Zivid One+ Large. Updated Operating distance and field of view table. Updated Zivid One+ Medium spec. plots. Updated connector table.
0.9	04/19	Initial version.

Zivid AS  
Gjerdrums vei 10A  
N0484 Oslo  
Norway

© 2019 Zivid AS. All rights reserved. Subject to change without notice.