FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Better end-to-end object detection in low SNR environments with time-of-flight cameras

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Technische Informatik**

eingereicht von

**Bernhard Müllner, BSc**
Matrikelnummer 01225855

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Univ.Ass. Dott.mag. Ramin Mohammad Hasani

Wien, 24. August 2019

Bernhard Müllner | Radu Grosu

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Better end-to-end object detection in low SNR environments with time-of-flight cameras

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Computer Engineering**

by

**Bernhard Müllner, BSc**
Registration Number 01225855

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Univ.Ass. Dott.mag. Ramin Mohammad Hasani

Vienna, 24th August, 2019

Bernhard Müllner | Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

# Erklärung zur Verfassung der Arbeit

Bernhard Müllner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. August 2019

Bernhard Müllner

# Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mir geholfen haben, diese Diplomarbeit zu vollenden.

Im besonderen möchte ich meinen Eltern danken, welche mir das Studium ermöglicht haben.

Vielen Dank Christina, für deine sprachliche, aber vor allem emotionale Unterstützung!

Mein Dank gilt auch der Firma BECOM Systems GmbH, welche den Time-of-Flight Sensor gratis zur Verfügung gestellt hat.

# Acknowledgements

I would like to thank everybody, who supported me in creating this diploma thesis.

A very special thank you to my parents, who have made this master's degree possible for me.

Thank you Christina! You supported me with your linguistic skills, but above all you were here when I needed you.

Many thanks also to BECOM Systems GmbH, who provided the time-of-flight sensor for free.

# Kurzfassung

Diese Diplomarbeit beschäftigt sich mit dem Problem, dass es keine Datensätze zum Trainieren von neuronalen Netzen gibt, welche mit einer Time-of-Flight Kamera aufgenommen wurden. Solche Datensätze sind aber unerlässlich, um bessere End-zu-End Objekterkennungssysteme, die auf Time-of-Flight Daten basieren, zu entwickeln. Daher wird in dieser Arbeit ein neuer, klar strukturierter Datensatz mit einer Time-of-Flight Kamera aufgenommen, welcher elektrische Handwerkzeuge zeigt. 32 Werkzeuge werden auf drei unterschiedlichen Hintergründen in zwei oder drei verschiedenen Positionen von verschiedenen Winkeln aus aufgenommen. Neben der Klassifizierung der Werkzeuge wird für die circa 100 000 Bilder auch ein Begrenzungsrechteck zur Verfügung gestellt.

Zum Testen des aufgenommenen Datensatzes werden zwei Faltungsnetze (englisch: Convolutional Neural Networks) mit diesem trainiert. Des Weiteren wird die Kombination aus Time-of-Flight Daten mit Farbbildern analysiert und implementiert.

# Abstract

This diploma thesis deals with the problem, that there is no big dataset available yet, which is usable to train neural networks and is recorded with a time-of-flight camera. Such datasets are necessary to develop better end-to-end object detection networks which use time-of-flight data as input. Therefore, a new, big and clear structured dataset showing electrical handheld power tools is generated. It shows 32 tools on three backgrounds in two or three positions, recorded from different angles. Beside a mapping to a class, a bounding box for each of the nearly 100 000 instances is provided. Additionally, for each recorded time-of-flight depth image, a synchronously taken color image is given.

For testing the newly generated dataset, two state of the art convolutional neural networks are trained on it. Moreover, the combination of the time-of-flight sensor and the color sensor is discussed and implemented.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation

In the last decade, computer vision has arrived in the center of our society. With the development of neural networks in combination with the rising power of parallel computing, processing of visual input data has become feasible. This technological leap forward, made it possible for computers to perceive their environment fast enough to seamlessly interact with it. Regardless of whether it is used in industrial applications to automate more and more tasks which were thought to be exclusively done by humans a few years ago, or in consumer applications like picture enhancement, neural networks are used all around us. The idea of neural networks is not new, however, the dropping prices and increased capability in parallel computing hardware, like GPUs, made it affordable in daily live applications. Moreover, progress made it possible to develop even small and cheap embedded systems which also take advantage of neural networks.

On the other hand the development of sensors, capturing our environment made also big steps forward and diverged in different directions. Sensors which allow the perception of the third spacial dimension become available. Technologies like ToF or LIDAR inherently provide depth information. Such kind of information, is used to further improve the computers' perception and mapping of their environment. Especially in robotics, the knowledge of surrounding objects and their exact spacial position is essential.

With two such powerful technologies at hand, the combination of both, is particularly promising.

## 1.2 Problem statement

Looking for some literature on ToF sensors used especially in combination with neural networks, no satisfying amount of existing research can be found. It is even harder to

find prerecorded datasets which provide real world objects recorded with a ToF sensor. This, however, is a prerequirement if someone wants to do research on NN with ToF input data. Therefore, the problem which will be solved in this thesis, will be the lack of ToF based input data, prepared for the use in neural networks.

The aim of this thesis is to generate an image dataset recorded with a ToF sensor. It will feature different objects in different categories, and will be useful for designing neural networks which use ToF sensor data as input and try to classify and localize objects. Thus, it will provide categorization and localization ground truth. It will be big enough to train neural networks with it, however, it also must be manageable in size and small enough for labeling the localization by hand. ToF sensors work very well in low SNR environments, therefore, this dataset will be recorded in such an environment. To verify the dataset, some state of the art neural networks will be trained on the generated data to get basic information on the quality and complexity of the dataset. The dataset will also feature a complementary standard RGB color image for each recorded ToF sensor generated image. This will allow a combination of RGB image data with ToF generated depth data. The thesis will also evaluate the improvements such a combination can provide.

## 1.3 Structure

This thesis is structured as follows: After this chapter (chapter 1) of introduction, an overview of the state of the art (chapter 2) is provided. It gives an overview of some existing datasets which are commonly used for training neural networks. A rough categorization of these datasets is done and completed with some facts about these datasets. This chapter will also introduce the basics of neural networks and will present some state of the art networks, which later will be used for verifying the dataset. The last part of the chapter will give a short introduction to the ToF technology.

Chapter 3 gives an overview of the methodological approach of this thesis. It starts with the analysis of an ideal dataset. The chapter then describes the method of how the dataset will be recorded, followed by an analysis of the parameters influencing the size of the dataset. It introduces the used ToF sensor and provides some basic information of combining RGB data with ToF data.

Chapter 4 deals with the actual creation and analysis of the dataset. It starts with the description of the decision process of which objects should be used, followed by the description of the backgrounds, the positions and the different tools. Afterwards it deals with the recording and labeling process. The used data formats are analyzed, and the main numbers of the dataset are discussed. Moreover, this chapter deals with the evaluation process using the YOLO network and UNET.

Chapter 5 compares the generated dataset with the existing ones and the last chapter (chapter 6) of this thesis sums the main points up and provides an outline for possible extensions to this work.

CHAPTER 2

# State of the art

## 2.1 Data sets

In this part of the thesis, an overview of available data sets for object classification problems is given. All mentioned data sets are generated by sensors, which enable computers to get a visual perception of their environment. For each dataset mentioned in this section, a table for a quick overview is provided. Due to the large number of available data sets, this cannot be a complete list, however, the chosen data sets should highlight the main characteristics of different categories of data sets.

A good overview of available 2D-RGB datasets can be found in [GDL+17]. For real world point clouds and synthetic 3D models [Ank] provides a good overview.

### 2.1.1 2D-RGB

The sensors used to generate the images in these data sets are simple digital cameras. Millions of pictures ordered within thousands of object categories (e.g. ImageNet [DDS+09]) and labeled with different ground truth such as categorization, segmentation or localization (c.f. [LMB+14]) are used in these datasets. The provided resolution of theses datasets started with 32x32 pixels at CIFAR-10 ([KH+09]) and is rising constantly.

**CIFAR-10**

| | |
|---|---|
| Number of labeled instances | 60 000 |
| Number of pictures | 60 000 |
| Provided categories | 10 |
| Number of objects per instance | 1 |
| Provided objects per category | unique objects per instance |
| Provided ground truth | category |
| Image resolution | 32x32 pixels |
| Shown objects | Animals and transportation vehicles |
| Overlapping categories | no |
| Recording Technology | RGB camera |

Reference: [KH+09]

**Coco**

| | |
|---|---|
| Number of labeled instances | 2 500 000 |
| Number of pictures | 328 000 |
| Provided categories | 91 |
| Number of objects per instance | multiple |
| Provided objects per category | unique objects per instance |
| Provided ground truth | segmentation,captions,category,key points |
| Image resolution | different, from 72x72 to 640x640 pixels |
| Shown objects | complex everyday scenes |
| Overlapping categories | no |
| Recording Technology | RGB camera |

Reference: [LMB+14]

**Pascal**

| | |
|---|---|
| Number of labeled instances | 27 450 |
| Number of pictures | 11 530 |
| Provided categories | 20 |
| Number of objects per instance | multiple |
| Provided objects per category | unique objects per instance |
| Provided ground truth | category, segmentation, captions, person layout |
| Image resolution | different |
| Shown objects | persons, animals, vehicles, furniture |
| Overlapping categories | no |
| Recording Technology | RGB camera |

Reference: [EVGW+10]

### 2.1.2 2.5D/RGB-D

All datasets which are sorted in this category have additional depth information along with a normal RGB image. One common approach to generate the depth information is stereo imaging. As described in [WWY+19], one uses two RGB cameras which are offset by a few millimeters. Through triangulation one can calculate the distance. With this data, a colored point cloud can be calculated.

Another method to record depth data is by using ToF. A short introduction to this technology is given in section 2.3. Combining this information with an RGB sensor leads also to a color image with additional depth information. Having this information, one can calculate a colored point cloud. This is the technology which will be used in this thesis to generate the dataset.

Another technology used to generate point clouds directly is LIDAR. By sending out a laser pulse, and detecting the reflected light, one can measure the running time of the light pulse, which is directly proportional to the distance. Rotating the laser over all axis, generates a 3D point cloud of the environment (e.g. [GLU12]). Again, the combination with an RGB camera leads to a colored point cloud.

All these technologies have in common, that they can be used to generate a 3D model of a real environment.

**KITTI2015**

| | |
|---|---|
| Number of labeled instances | 80 000 |
| Number of pictures | 7481 |
| Provided categories | 2 |
| Number of objects per instance | multiple (max 15 cars, max 30 pedestrians) |
| Provided objects per category | unique |
| Provided ground truth | global position, trajectory,<br>optical flow map, 3D object labels |
| Image resolution | RGB: 2x1392x512 pixels<br>Gray scale 2x1392x512 pixels<br>3D laser scanner, 64 beams |
| Shown objects | people, cars |
| Overlapping categories | no |
| Recording Technology | laser scanner, RGB sensor |

Reference: [GLU12]

**Caltech 3D objects**

| | |
|---|---|
| Number of labeled instances | 43 200 |
| Number of pictures | 43 200 |
| Provided categories | 100 |
| Number of objects per instance | 1 |
| Provided objects per category | 1 |
| Provided ground truth | category |
| Image resolution | 2x3M-pixels RGB sensor |
| Shown objects | small daily life objects |
| Overlapping categories | no |
| Recording Technology | Stereo Imaging |

Reference: [MP07]

**RGB-D object dataset**

| | |
|---|---|
| Number of labeled instances | 41 877 |
| Number of pictures | 41 877 |
| Provided categories | 51 |
| Number of objects per instance | 1 |
| Provided objects per category | 3-14 |
| Provided ground truth | category, localization |
| Image resolution | 640x480 |
| Shown objects | daily life objects |
| Overlapping categories | no |
| Recording Technology | Microsoft Kinect |

Reference: [MMEB18]

### 2.1.3 3D models

The last section 2.1.2 had the aim to generate 3D models from the real world. However, in this category, datasets with synthetic 3D models, which are made up from computer generated 3D models, with no measurement errors or other artifacts generated by recording objects, are listed.

**PartNet**

| | |
|---|---|
| Number of labeled instances | 573 585 |
| Provided categories | 24 |
| Number of objects per instance | 1 |
| Provided objects per category | 127-9906 |
| Provided ground truth | category, partition |
| Shown objects | daily life objects |
| Overlapping categories | no |
| Recording Technology | 3D CAD models |

Reference: [MZC+19]

**Thingi10K**

| | |
|---|---|
| Number of labeled instances | 10 000 |
| Provided categories | 4892 |
| Number of objects per instance | 1 |
| Provided objects per category | very different |
| Provided ground truth | category, subcategory, tags |
| Shown objects | 3D printable models |
| Overlapping categories | no |
| Recording Technology | 3D CAD models |

Reference: [ZJ16]

## 2.2 CNNs

Most NNs designed for classification problems are based on a convolutional approach like the networks presented in this section. Therefore, the search for state of the art networks was limited to this same kind of NN.

After some basics about NNs, this section introduces some specific CNNs, to give an example. They were chosen, because they were presented in the last few years and achieve good results in classification on commonly available datasets as presented in section 2.1. These networks will serve as a reference on how complex the dataset generated in this thesis is.

### 2.2.1 Basics

Using NNs requires some basic background information about how they work internally. For this reason, the main working principle of a neuron is explained.

#### Neuron

A neuron, as described in equation 2.2 and figure 2.1, operates on $n$ input values called $a_i$ where $i = 1..n$. Input $a_0$ is fixed to 1 and can set a bias for the neuron over its weight

$w_0$. Each input value is first multiplied by a weight $w_{i,j}$ and then summed up. This gives the intermediate value $in_j$ as described in 2.1. This intermediate value $in_j$ is then fed through an activation function which gives the output vale $a_j$. Activation functions are separately described in the next paragraph.

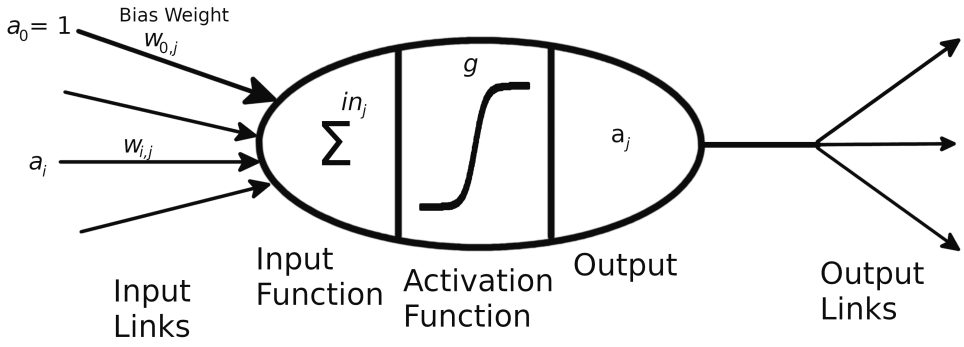



Figure 2.1: Basic description of a neuron, based on [RN16]

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{2.1}$$

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right) \tag{2.2}$$

### Activation functions

The activation function of a neuron is the part which makes it non-linear. Early NNs started with using a sigmoid as activation function. Equation 2.3 and figure 2.2 illustrate the function. This function projects all real values in a range between 0 and 1. A benefit of this function is, that it is steady and therefore, the derivation can be easily calculated. This is a useful property when using a backtracking algorithm which uses gradient descent.

In recent years, many NNs use the ReLU function as activation function (e.g. [RFB15]). As one can see in equation 2.4 and figure 2.3, it is far easier to compute and generates results of similar quality.

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

$$g(x) = max(0, x) \tag{2.4}$$

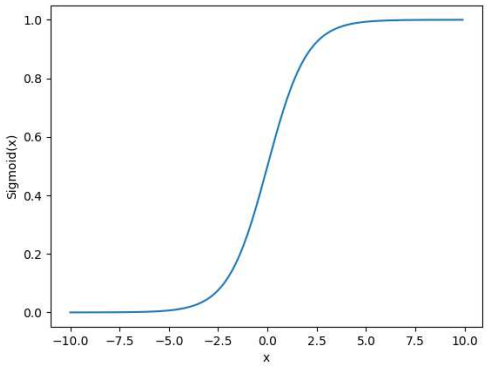



Figure 2.2: Sigmoid function

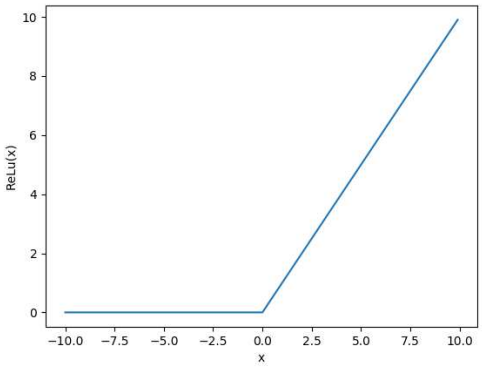



Figure 2.3: ReLU function

### 2.2.2 2D-CNNs

#### YOLO

The YOLO network was introduced in [RDGF16]. It is designed for fast detection on videos. It achieves this by making the prediction by a single pass through the network in contrast to other approaches which work on sliding windows or region proposal-based techniques. The image is divided into blocks, where each block itself predicts a predefined number of bounding boxes plus a confidential value for the bounding box which is independent from the predicted class. Each block also predicts a conditional class probability which is used for classification. This network is available in its third version [RF18]. Figure 2.4, which is originally published in [RDGF16], shows the structure of the network in its first form, parameterized for the use with the PASCAL VOC dataset ([EVGW+10]).

#### UNET

This network is originally designed and used for biomedical image segmentation and was presented in [RFB15]. Figure 2.5 shows the structure of the network. The network first uses convolutional layers followed by upsampling layers, to get the same dimensions as the input. Shortcuts between these down- and upsampling blocks provide additional information.

### 2.2.3 3D-CNNs

In contrast to 2D-CNNs as presented in 2.2.2, 3D-CNNs try to directly exploit the depth information. This approach has only become feasible in recent years, because the data size multiplies with the depth resolution.

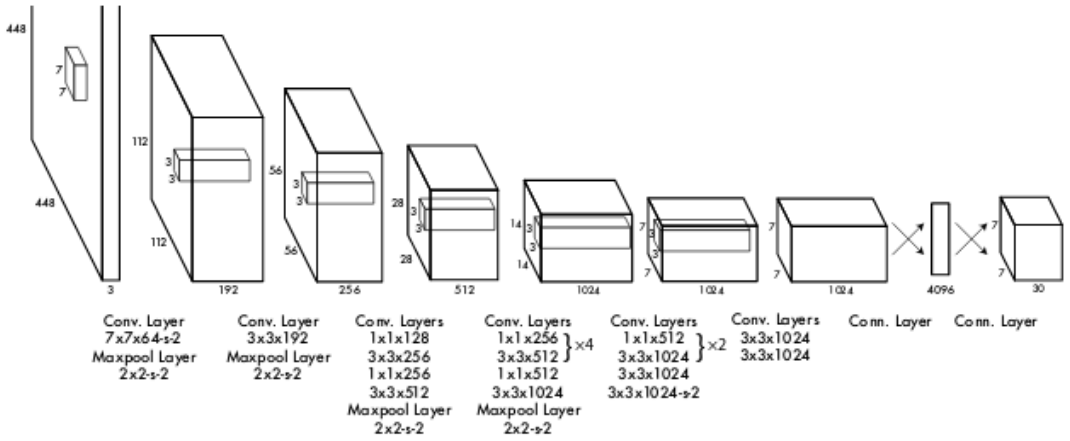



Figure 2.4: Structure of the YOLO network, configured for the use with the PASCAL VOC dataset [RDGF16]

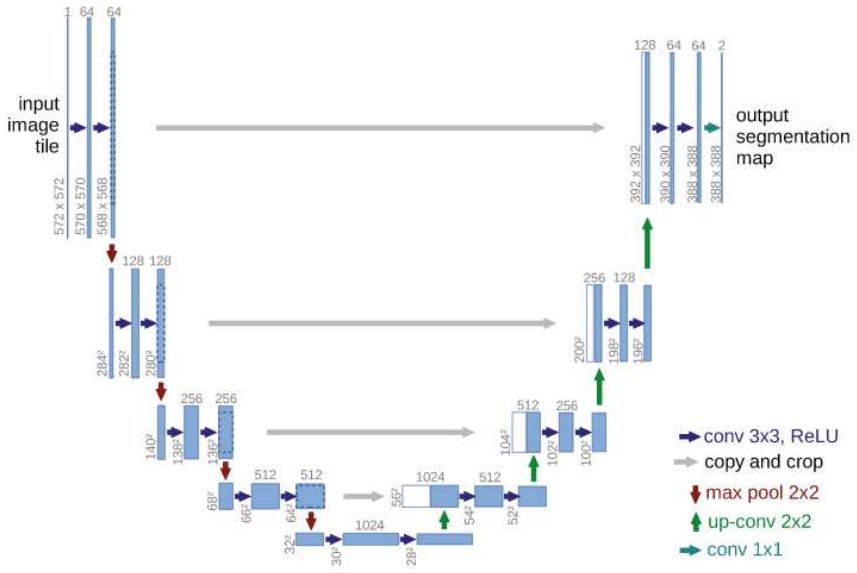



Figure 2.5: Structure of U-Net as presented in [RFB15]

### Object Grasping

In the project described in [CSDR18], a depth sensor was used to generate a point cloud which is voxilized and then fed to a 3D-CNN (see figure 2.6) to perform a grasping task with a soft hand on a robot.

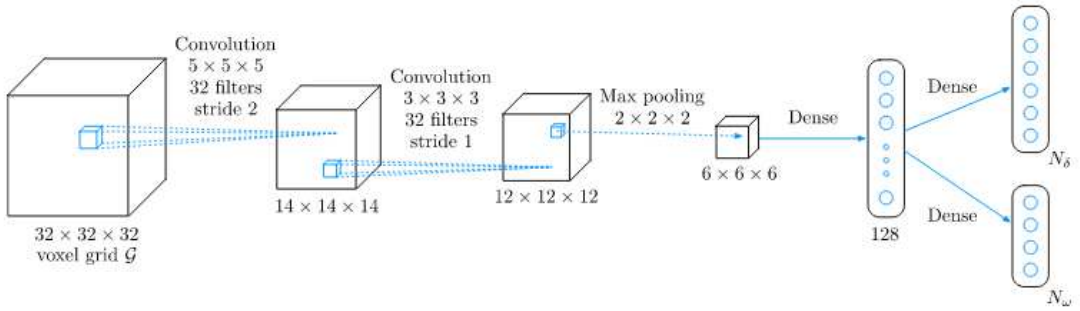



Figure 2.6: Structure of the 3D-CNN used in [CSDR18]

### VoxNet

VoxNet tries to tackle the large data structures needed for storing the objects with an occupation grid. It assigns a probability to each voxel, if it is free space or if the voxel is part of an object. This occupation grid is fed to a 3D-CNN. The basic structure is shown in figure 2.7 as it is presented in [MS15]

## 2.3 Time-of-flight

This section gives a short introduction to the basic principle of the ToF technology. One can use different approaches to exploit ToF for distance measurement. The sensor used in this thesis is based on the correlation measurement approach with the PMD pixel. Therefore, only this approach will be discussed here. Figure 2.8, which is based on [HM07], shows a simple illustration of the working principle. The capturing device consists of a sensor part and an illumination part. The illumination is usually done in the IR spectrum.

The goal of the measurement is to send out a modulated light signal, and measure the phase shift to the recorded signal which was reflected at an object. This phase shift is directly proportional to the distance of the object.

To generate one depth image, four measurements are made. Each pixel has two potential pots for integrating incoming light. With the modulation gates of the pixel one can influence in which pot electrons generated from incoming light should go.

By modulating the pixel with the same signal as the illumination, and subtracting the voltage value from one pot from the voltage value from the other, one gets the value of

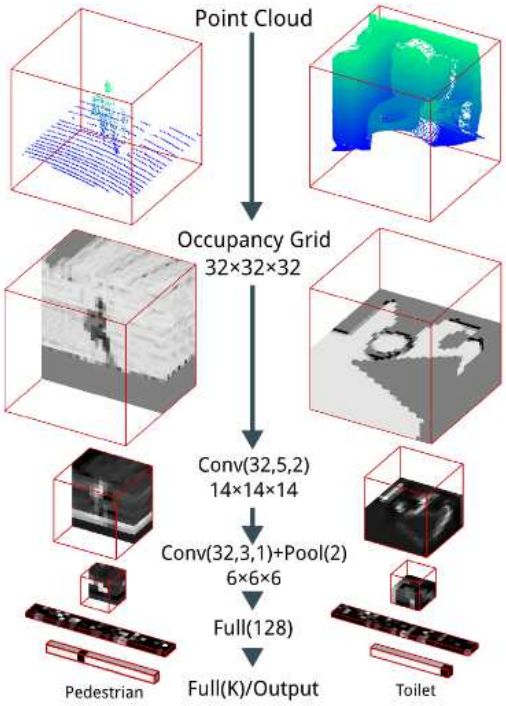



Figure 2.7: Structure of VoxNet as presented in [MS15]

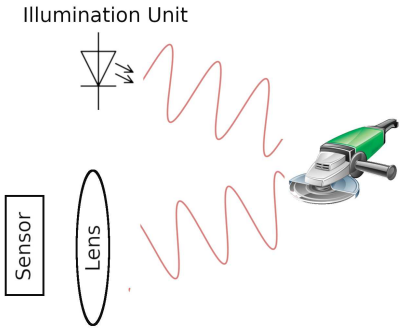



Figure 2.8: ToF principle

the correlation function (see equation 2.5) at $\tau = 0°$, where $s(t)$ is the optical received signal and $g(t)$ is the modulation signal.

$$c(\tau) = s(t) \otimes g(t) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t) * g(t+\tau)dt \tag{2.5}$$

This function correlates the sent signal and the received signal. The received signal has in theory the same frequency and signal shape, however, with an offset, changed amplitude and a phase shift.

In the following measurements, the modulation signal of the pixel (demodulation signal) is shifted by $\tau = 90°$, $\tau = 180°$ , and $\tau = 270°$. These four measurements give grid points for reconstructing the correlation function which is shown in figure 2.9 based on [HM07]. The phase shift is given by equation 2.6, the amplitude by equation 2.7 and the offset by equation 2.8 as one can read in [OLK+04]. Now the phase shift can be transformed

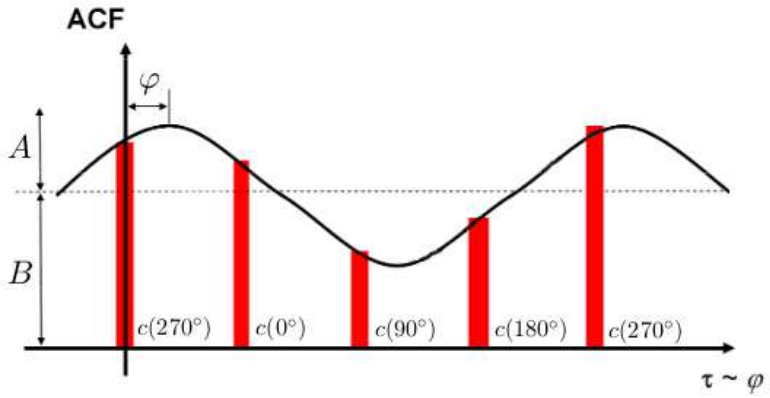



Figure 2.9: Correlation function generated from sample points

with equation 2.9 to the actual distance. $L$ as described in equation 2.10 is called the non-ambiguity distance, and limits the range of operation.

$$\varphi = atan\left(\frac{c(270°) - c(90°)}{c(0°) - c(180°)}\right) \tag{2.6}$$

$$A = \frac{\sqrt{[c(270°) - c(90°)]^2 + [c(0°) - c(180°)]^2}}{2} \tag{2.7}$$

$$B = \frac{c(0°) + c(90°) + c(180°) + c(270°)}{4} \tag{2.8}$$

$$D = L * \frac{\varphi}{2 * \pi} \tag{2.9}$$

$$L = \frac{c}{2 * f_m} \tag{2.10}$$

where:

$c$ ... speed of light
$f_m$ ... modulation frequency

# CHAPTER 3

# Methodology

## 3.1 Ideal dataset

Based on the variables gathered from the state of the art datasets, the perfect dataset should provide a high number of recorded images with high resolution, multiple objects in one instance and a decent number of categories with many different objects mapped to each category. Ideally, all instances are fully labeled with the corresponding category and all shown objects are labeled with its position and size. Moreover, the segmentation of all shown objects from the background is provided. The shown objects in the instances should be different enough to categorize them, but also, the categories should be similar enough to provide a decent complexity.

## 3.2 Method of generating the images

A common method of gathering visual data is to use preexisting images from various online sources and classify them. As stated in section 1.2, this is not possible in this thesis, because there is little to no free available raw data from time of flight cameras.

In existing publications like [MP07] a common method of recording a new dataset, which does not rely on preexisting images, is the use of a rotating table. As one can see in figure 3.1, the camera is fixed in front of the object. The object itself is placed on a plate, which can be turned by hand or by a motor. This leads to very consistent changes between the pictures. In [MP07] even the angle between each recorded image is known.

Now for this thesis, the method of the rotating table was adapted. Instead of the camera, the object will be fixed in one position for one recording sequence. During this recording sequence, the camera will be freely moved around the object to gather data from different angles, while recording with a low frame rate. This method has the advantage, that the angle is changed over two axis instead of one during recording. To

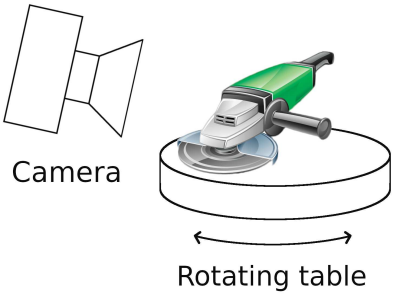



Figure 3.1: Basic structure of a recording setup with a rotating table, partially based on [fre19]

do this as systematically as a motor controlled rotating table, a complex mechanical mounting would be needed. To overcome this issue, the camera will be freely moved over the object. This has the disadvantage, that the angle change between two images will not be constant. However, this can also be seen as an advantage, because a well trained NN should be able to generalize over the given dataset and should also be able to predict such unseen angles. Therefore, the complexity of the dataset is increased. Figure 3.2 visualizes the described method.

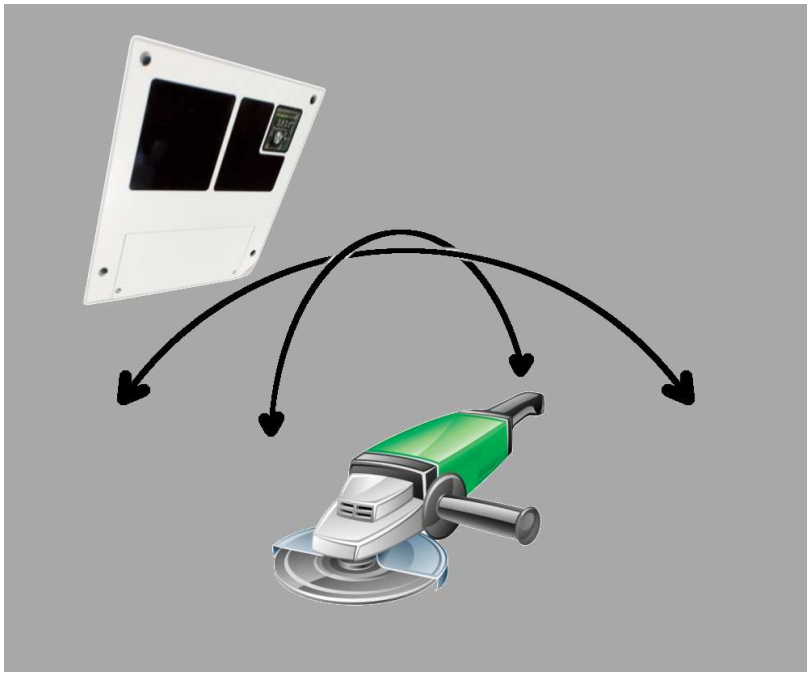


Figure 3.2: Recording of one sequence, partially based on [fre19]

## 3.3 Size of the dataset

A main question when generating a dataset is, how big it should be. As described in chapter 2 the size of the shown dataset is very different. One can state, that a high number of data points increases the quality of the dataset and makes it easier for the NN to generalize over it. There are many parameters which can be tuned to enlarge a dataset. In section 3.3.1 the parameters are explained, and in section 3.3.2 the calculation of the size dependent on these parameters is shown.

### 3.3.1 Parameters

In the following paragraphs, four parameters are explained, which will also be the main parameters for the dataset generated in this theses.

#### Categories

One of the basic parameters of a dataset mainly generated for classification, is the number of different categories. Simple datasets start with two categories [GLU12], early day datasets such as CIFAR-10 [KH+09] provide ten categories and state of the art datasets provide up to thousands of different categories (e.g. [ZJ16]). However, increasing the number of categories, must not lead to a decrease in instances at each category. Therefore, a smaller number of categories will be chosen for generating the dataset, however, the quality of the individual categories will be ensured.

#### Position

To further increase the possible different recording angles, the objects themselves will be placed in different positions. In order to get a maximum of different views, as many stable positions as possible for an object will be found and used for recording. It would be possible to further increase the number of positions, by coming up with different holding mechanisms. However, if one wants to get pictures of objects as they can be found in real life, such artificial mountings may negatively affect the quality of the dataset. Therefore, this idea was abandoned for this thesis. For choosing a new position for an object, it should also be considered that a visible difference between the position is given, otherwise the gained information for the dataset will be too small.

#### Background

Recording an object always on the same background may lead to wrong training. The NN may train itself on features on the background to classify the object. This is not intended, therefore, using different backgrounds will be essential to generate a usable dataset. Using different backgrounds also increases the number of different images which can be recorded, because one can replicate the same recording method on a new background and will get completely different images. Due to the fact, that a ToF sensor records depth

data, it will be important, that a change in the background is not only a change in color, moreover, it must also have a different structure.

#### Devices

A dataset of high quality features also many different devices in each category. Achieving this goal will be a very hard one, since getting hands on different physically available devices will afford some effort in organization, and therefore, must be a main point of consideration when choosing the categories.

### 3.3.2 Number of images

With the described approach of recording in section 3.2 and the mentioned variables above, the total number in such a generated dataset can be calculated as one can see in equation 3.1. The equation assumes, that the number of positions per device is the same over all devices in one category. Moreover, it is assumed, that all recordings are done on all backgrounds. Equation 3.2 comes in handy at recording, when one wants to know how many images should be recorded in each setup.

$$t = b * \sum_{i=1}^{i=c} d_{c_i} * p_{c_i} * r_{c_i} \tag{3.1}$$

$$i_{c_i} = \frac{t}{d_{c_i} * p_{c_i} * b * c} \tag{3.2}$$

where:

$t$ ... Number of overall images
$c$ ... Number of categories
$b$ ... Number of backgrounds
$p_{c_i}$ ... Number of positions per category $i$
$d_{c_i}$ ... Number of devices per category $i$
$r_{c_i}$ ... Number of images per recording sequence in category $i$

## 3.4 Further variables for generating a dataset

### 3.4.1 Number of objects in one picture

A decent complex dataset may contain any number of objects in one image of the dataset, as one can see in [LMB+14]. Such a dataset is hard to label, since categorizing many objects and drawing bounding boxes around them increases the labeling time per image

drastically. However, the dataset should be generated as part of this theses, therefore, the time for labeling is limited. This is why some cuts in complexity must be made. One of these cuts is, that exactly one object is recorded on one image in the dataset. In section 6.2.1 a possible extension to a multi-object-per-image dataset is outlined.

### 3.4.2 Ground truth

The time consuming part at generating a dataset for training a NN is the supply of ground truth. As shown in section 2, different datasets provide different ground truth. When recording with a ToF sensor and using the recording method described in section 3.2, one can easily provide the category of the object, because each recording sequence features exactly one object.

To provide ground truth for localization, an image, showing the segmentation of the object from the background would be the best option. However, due to the size of the planned dataset (see also section 3.3), a manual segmentation of the objects will not be feasible. To provide ground truth nevertheless, a simple localization can be achieved by marking the object in the picture with a simple bounding box. This can be done manually for a large number of images without relying on algorithms or NNs. This will deliver the best possible quality of ground truth.

### 3.4.3 Overlapping categories

Overlapping categories make a dataset even more complex. An example for overlapping categories would be if one wants to classify the color of a car. Most cars may only feature one color, however, there are cars out there, with yellow doors, black roof and pink engine cowling. To avoid overlapping categories, one can have a category called $Colorful$. The other approach would be, sorting it in the categories yellow, black and pink. The dataset generated in this thesis will not feature overlapping categories, because it would restrict the selection of the categories even more.

## 3.5 Recording technology

As already mentioned in the acknowledgments, the sensor for recording the dataset was provided by BECOM Systems. The P320 is a ToF sensor based on a sensor chip from PMD technologies. It uses the correlation of emitted and received light pulses as described in section 3.5, to calculate the distance to the objects in front of the sensor. The recorded ToF image has a resolution of 160x120 pixels. Moreover, it features an additional RGB sensor for synchronously recording a RGB picture for each captured ToF image. The data is delivered over Ethernet, and can be captured by a provided software. Figure 3.3 shows the ToF camera. It marks the position of the integrated sensors and pictures how the output of the sensors look.

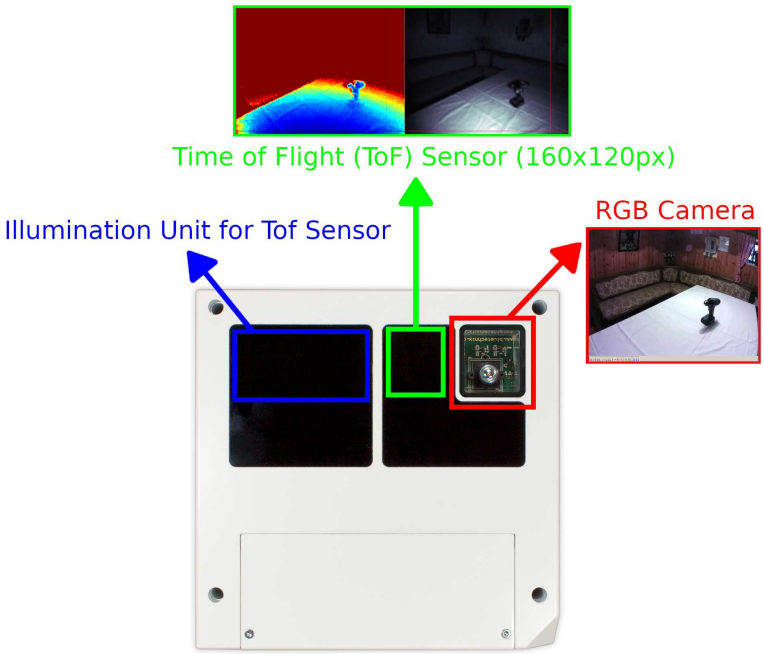



Figure 3.3: Overview of the P320 sensor system

During recording the sensor must be connected to a power supply and to the network. This limits the range for recording in contrast to RGB sensors which can be found in many mobile devices.

To get useful recording results, the following parameters have to be checked before starting recording: First, one should be aware of the non-ambiguity range of a time of flight sensor. In the default settings, the modulation frequency is set to $20MHz$. According to equation 2.10, this range is $7.5m$. When recording inside a room, this should not come into effect.

Another value which should be checked is the integration time. Integrating for a too short time period, the sensor will not be able to capture enough light and the value will be imprecise. However, integrating for too long, will cause saturation on the sensor, and the measured distance value will again be wrong. Therefore, the integration must be set to a reasonable value once the objects are defined.

Another parameter to consider in choosing the categories will be, the depth resolution of the ToF sensor. A sensor system like the P320, has a depth resolution between $0.5cm$ and $1cm$. This further limits the variety of useful objects for this dataset.

## 3.6 Evaluation metrics

### Category accuracy

As a fist, simple metric, category accuracy is used. It states, how many percent of the predicted images in a category are predicted correctly.

### IOU

As a second metric IOU is used. It provides a value for how precise the predicted bounding box matches the bounding box from the ground truth. It is defined as stated in equation 3.3

$$IntersectionOverUnion = \frac{Area_{predictedBoundingBox} \cap Area_{groundTruthBoundingBox}}{Area_{predictedBoundingBox} \cup Area_{groundTruthBoundingBox}} \tag{3.3}$$

## 3.7 Combining RGB + depth data

Due to the fact, that the ToF sensor used for generating the dataset, provides an additional RGB sensor, the combination of both images is obvious. There are a few different approaches which will be presented in this section.

### 3.7.1 Calculate point cloud

One way of combining both sensors is to use the geometry of the whole sensor to calculate an accurate colored point cloud. The basic idea is that by combining the lens parameters with the sensor dimensions, one can calculate a unit vector for each pixel of the ToF sensor. By multiplying this unit vector with the measured distance value, one can transform the value into a voxel. By knowing the distance and positioning of the RGB sensor in relation to the ToF sensor, one can transform both sensors into a global coordinate system.

For each pixel of the RGB sensor in combination with its lens, a truncated pyramid can be spread out. By intersecting the truncated pyramids with the point cloud one can assign each voxel the corresponding color recorded by the pixel on the RGB sensor.

This method works quite well, however, needs a lot of computation power. In this thesis, the classification will be done end-to-end, and therefore, this computation will be dismissed. This leads to the other two methods of combining ToF data with RGB data.

### 3.7.2 Direct combining and feeding into the network

One way of combining the data will be to stack the depth layer as 4th layer on a standard RGB image. This will preserve mostly the locality property, however, it is not accurate due to the above described transformation which must be done to get the correct values for each RGB pixel. The expectation of this approach is, that the CNN will learn the transformation and corrects it.

### 3.7.3 Split classification

The simplest approach is to train two separate CNNs, one on the RGB data and one on the ToF data and let both networks make their predictions. As a first approach, averaging over the predictions will show if a combination of both sensors can lead to better classification results.

CHAPTER 4

# Implementation

## 4.1 Preparations

### 4.1.1 Object categories

One of the first big decisions made for this thesis, was choosing the right classes of objects. First of all, all objects in the dataset must be physically available, due to the fact, that there are only few ToF generated RGB-D images freely available. This criteria excluded all kind of industrial machining equipment or industrial tools. Secondly, the objects have to be small enough to fit inside a room, because the used ToF sensor must be connected to a PC over an Ethernet cable during recording. This criteria made it impossible to use cars, buildings, ships, planes, animals and even big furniture as objects. Another aspect was, that the objects must be big enough, that they are detectable by the used ToF sensor. The resolution of the ToF sensor is between $5mm$ and $10mm$, therefore, the objects must be at least a few centimeters in each dimension. Excluding small devices prevented the use of simple hand held tools like hammers, screw drivers and wrenches. Moreover, there should be at least about 10 categories, to get a dataset of decent complexity. To avoid overtraining on one specific device, the dataset should contain at least a few different objects in each category.

All the objects shown in the datasets described in section 2.1 do not match the above mentioned criteria. Therefore, we came up with the idea to use electrical handheld power tools. These tools fulfill all requirements. As a first approach only corded electrical handheld power tools were chosen. However, getting enough devices which match this tight definition, was not possible. Therefore, battery powered tools were also included. Table 4.1 shows the different categories and a representative picture taken from the dataset recorded with the RGB sensor on the ToF sensor system.

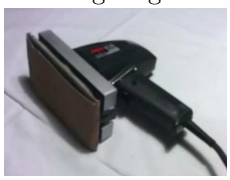
| | | | |
|---|---|---|---|
|  |  |  |  |
| Angle grinder | Battery power drill | Batt. screw driver | Big angle grinder |
|  |  |  |  |
| Chain saw | Hand planer | Hot glue gun | Jig saw |
| |  |  | |
| | Power drill | Vibration grinder | |

Table 4.1: Different power tools which are representative for their category

### 4.1.2 Backgrounds

As described in section 3.3.1, with a changing background the number of instances in the dataset can easily be increased. Therefore, three different backgrounds were chosen. Due to the limitations of the recording system described in section 3.5, all backgrounds are located in one room. The chosen backgrounds are a table, the floor of the room and a workbench. They are shown in table 4.2.

The backgrounds were chosen in such a way that they differ in color and shape. The color difference is obvious, the difference in shape can be described as follows: The floor, for example, is a flat surface, where the furniture of the room and the walls rise up on the sides. The table on the other hand, is also a flat surface, however, it ends with sharp edges, continues with a flat surface down on the floor and is followed by the rise of the walls again. Last but not least, the workbench, is surrounded by 2 walls, and two sharp edges in the front.

This will help a NN to better separate the tools from the background and to do a better generalization on the dataset. The room, where the dataset is recorded, provides bad lighting. This affects the quality of the RGB images, however, the quality of the ToF data will not be influenced, since a ToF sensor has inherently an own light source and is not dependent on ambient light (see also section 2.3).


Floor


Table


Workbench

Table 4.2: Different backgrounds on which the images were recorded

### 4.1.3 Position

To cover the tools from all directions during recording, different positions of the tools were chosen. For most of the tools three different positions could be found. As one example table 4.3 shows a big angle grinder in its three recorded positions. For tools like a power drill, only two stable positions could be found.

Moving the tools on the table without changing their position was also an option, however, the difference between the images would be too small, and therefore, this variation in position was abandoned.


First position


Second position


Third position

Table 4.3: A big angle grinder in three different positions

### 4.1.4 Tool diversity

To get a better generalization of what a tool sorted in a category can look like, different devices from different manufacturers were chosen. For some tools it was quite hard to find different models. At least two different devices for each category were used. In the raw data of the dataset they are numbered from 1 to n for distinction. Table 4.4 shows the different devices used for generating the chain saw category.


Black and Decker chain saw


Alko chain saw


KSI chain saw

Table 4.4: Different brands of chain saws, used for the chain saw category

### 4.1.5 Different angles

Due the recording method described in section 3.2, different views of an object that is not moved during a recording session, can be achieved. Table 4.5 shows some of the different angles captured during one session.

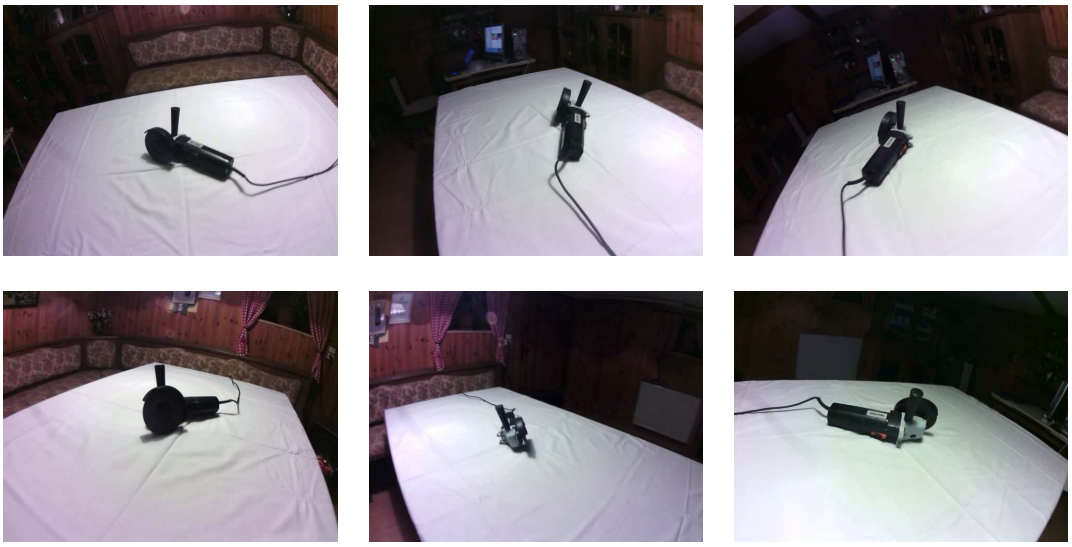


Table 4.5: Some of the different angles captured during recording

### 4.1.6 Final setup

Summing up the last sections, devices from ten categories are recorded on three different backgrounds. An overview of the number of positions and devices used during recording, is given in table 4.6. The appendix gives a bigger overview of the recorded images.

| | Angle grinder | Batt. power drill | Batt. screw driver |
|---|---|---|---|
| Positions | 3 | 3 | 2 |
| Devices | 4 | 3 | 3 |

| | Big angle grinder | Chain saw | Hand planer | Hot glue gun |
|---|---|---|---|---|
| Positions | 3 | 3 | 2 | 3 |
| Devices | 2 | 3 | 3 | 3 |

| | Jig saw | Power drill | Vibration grinder |
|---|---|---|---|
| Positions | 3 | 2 | 3 |
| Devices | 3 | 7 | 2 |

Table 4.6: Setup overview

## 4.2 Recording

### 4.2.1 Number of images

The target size of the dataset was set to about 60 000 images. One can use 80% of dataset for training, which are about 48 000 images and use 10% for validation and the last 10% for testing which are about 6 000 images. These numbers seam realistic and sufficient compared with the datasets described in section 2.1. Furthermore, it is also a realistic number of images to label manually. To get an equally balanced dataset, each of the ten categories was planned to contain about one tenth of the total number of images in the dataset. Using equation 3.2, from section 3.3.2, the number of images per recording session was calculated as one can see in table 4.7. The numbers were rounded up to the next integer value.

| | Angle grinder | Batt. Power drill | Batt. screw driver |
|---|---|---|---|
| # of images to record | 167 | 223 | 334 |

| | Big angle grinder | Chain saw | Hand planer | Hot glue gun |
|---|---|---|---|---|
| # of images to record | 334 | 223 | 334 | 223 |

| | Jig saw | Power drill | Vibration grinder |
|---|---|---|---|
| # of images to record | 223 | 143 | 334 |

Table 4.7: Wanted picture count for an equally balanced dataset with 60 000 instances

### 4.2.2 Recording session

For the recording sessions the frame rate of the ToF sensor was set to five FPS. As mentioned in section 3.5, the illumination time had to be set before starting recording. Using 500us integration time, leads to usable values in the amplitude picture and avoids over-illumination on close recordings. With the numbers from table 4.7, and the knowledge, that not all pictures recorded would be usable, the aim was to record 350 to 400 pictures per recording session. With equation 4.1, one gets the recording time for one session, which is between 70 and 80 seconds.

$$recordingTime = \frac{numberOfImages}{FPS} \tag{4.1}$$

Before starting the recording, the tool was placed on the background in one of the defined positions. A stream can be recorded using the software provided with the sensor. This stream is uncompressed and contains all the ToF data and also the RGB images, which leads to files with a few hundred megabyte in size. During the recording, the camera was moved manually over the tool as described in section 3.2. Due to the use of a video mode to capture images, combined with manual starting and stopping, the amount of captured frames varies. Figure 4.1 shows the recording software provided with the sensor.

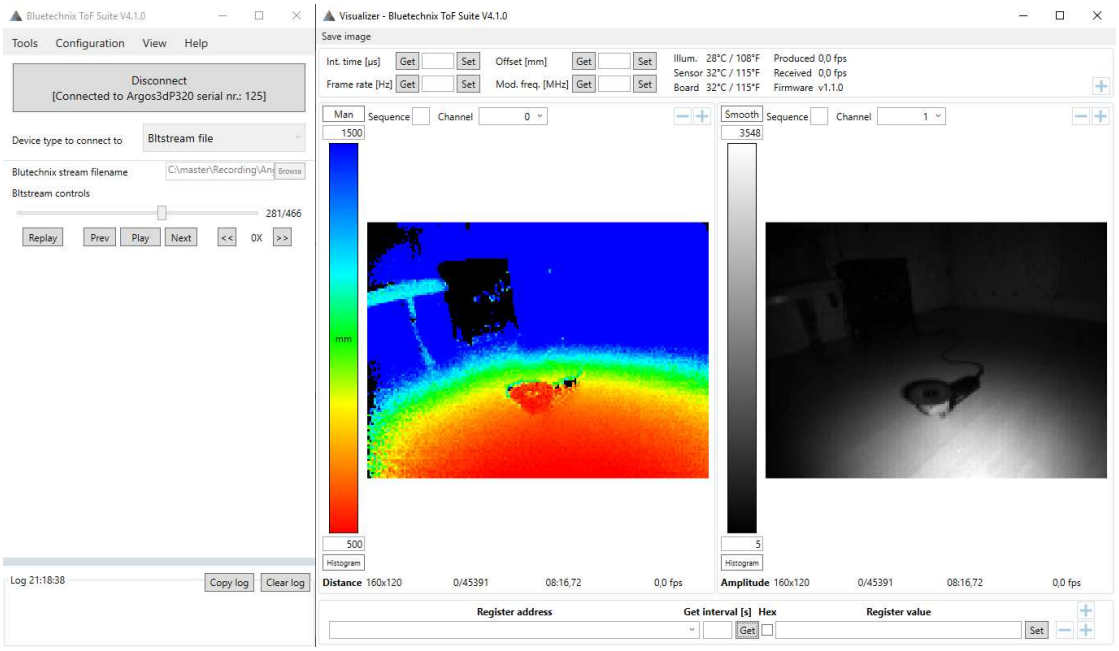



Figure 4.1: BECOM ToF Suite used for recording the data stream

### 4.2.3 Extracting data

The streams saved from the recording sessions, are in the *bltstream* format. Using the API provided for the ToF sensor it is quite simple to extract the data from it. The

extracted data was stored as CSV files. Each recorded image, consists of five channels. The ToF sensor provides an amplitude channel and a depth channel. The RGB sensor delivers the three color channels. For easier processing later on, the channels were stored in separate CSV files. To keep track of all files, the directory structure shown in figure 4.2 was chosen. The program for extracting the data is included in the appendix.

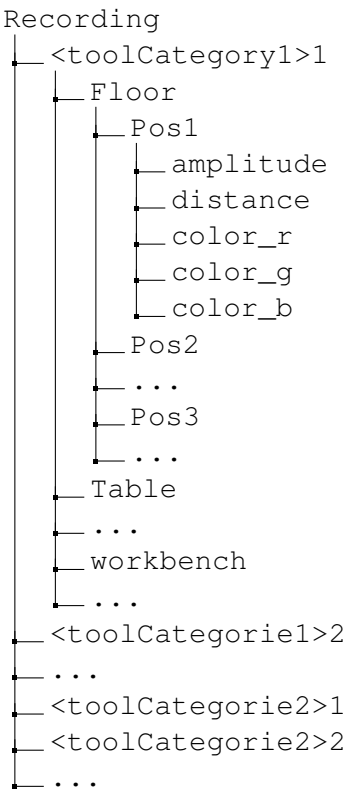



Figure 4.2: Directory Structure

## 4.3 Labeling

Due to the recording strategy as described in section 3.2 each recording session can be directly labeled with a certain class. To add additional complexity to the dataset, a localization feature in form of a bounding box was added. For labeling all images, a python script was written which is included in the appendix. It makes marking easy, however, each image was marked by hand, to provide a high quality ground truth. It is built in such a way, that only two clicks with the mouse are needed per image, for marking the corners of the bounding box. With the other hand, the space bar can be hit to get to the next image. With this program it was possible to mark all recorded images in three weeks. During this labeling process, bad images, where no object was captured, were marked as bad. For each recording session, a *pickle* file *deleted.conf* was generated,

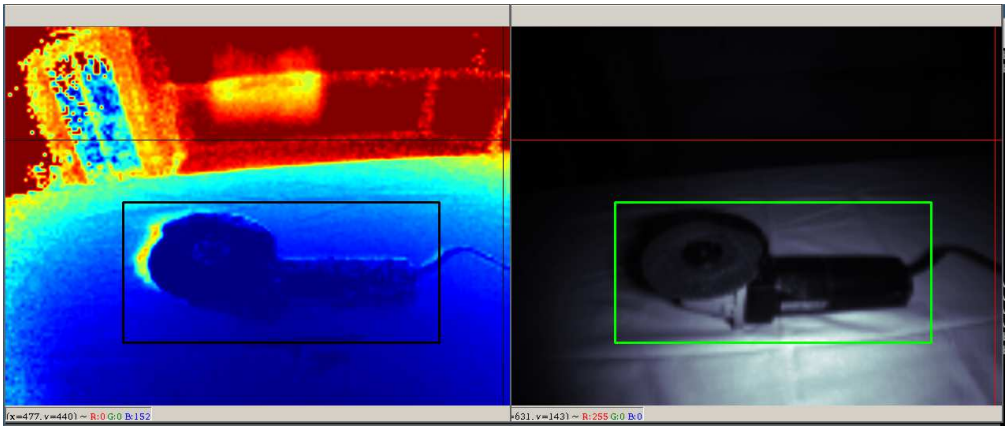


Figure 4.3: Fully labeled image

storing the info if an image is useful for further use in the dataset or not. A second *pickle* file, *labels.conf*, is generated, which stores the corners of the bounding box for each file.

Figure 4.3 shows a fully labeled image.

## 4.4 Dataset

### 4.4.1 Image Data Format

#### ToF

The depth data generated by the ToF sensor were restricted to values between 300$mm$ and 1500$mm$. These restricted values were then colored using a JET-stream coloring shown in figure 4.4. Blue represents a value of 300$mm$, red a value of 1500$mm$. The coloring in between is linear. The colored data was then stored as a standard PNG image. The resolution for the ToF data was kept at the original resolution of 160x120 pixel.

Figure 4.4: Jet coloring for depth data

#### RGB

The three channels recorded with the RGB sensor were combined and stored as a standard PNG picture. The resolution for the RGB data was reduced to 160x120 pixel to make it comparable with the ToF data. The original images can always be generated using the channels provided in the raw data.

### 4.4.2 Label data format

For storing the labeling information, a simple data format as described in [RDGF16] was used. It has one text file for each picture. Each bounding is stored as a new line. The structure of the line can be seen in table 4.8 The category is represented as an integer

| category | x position | y position | width | height |
|---|---|---|---|---|

Table 4.8: Structure of a line in a labeling file

number starting at zero. The x and y position represent the center point of the bounding box. Each file can contain more than one line, however, the dataset generated in this thesis contains only images with one object in each picture. Therefore, all generated labeling files contain only one line.

### 4.4.3 Final instance count

Table 4.9 was generated by counting all recorded and labeled images which were not marked as bad. Figure 4.5 shows, that the generated dataset is not equally distributed. However, the target size of the dataset with 60 000 images which are equally distributed over the categories can be easily extracted, because even the category with the least valid images, has more than 6 000 instances (see table 4.9).

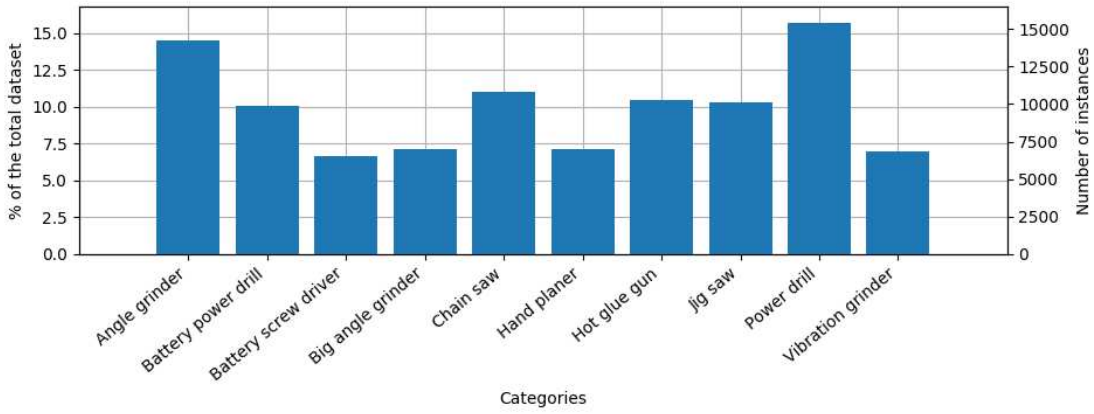



Figure 4.5: Distribution over the categories

### 4.4.4 Directory structure

With a dataset of nearly 100 000 color images and nearly 100 000 depth data images it is essential to have a clear structure for storing the images. The directory structure shown in figure 4.6 tries to organize the dataset and makes it comfortable to use. The structure is slightly based on the PASCAL Dataset ([EVGW+10]).

```
TofDataSet
├── ImageSets
│   ├── EqualTrain.txt
│   ├── EqualValidation.txt
│   ├── EqualTest.txt
│   ├── FullTrain.txt
│   ├── FullValidation.txt
│   └── FullTest.txt
├── DepthData
│   └── <ObjectCategory><NumberOfDevice><background>Pos<positionNumber>
│       <fourDigitConsecutiveNumber>.png
├── RgbData
│   └── <ObjectCategory><NumberOfDevice><background>Pos<positionNumber>
│       <fourDigitConsecutiveNumber>.png
├── Labels
│   └── <ObjectCategory><NumberOfDevice><background>Pos<positionNumber>
│       <fourDigitConsecutiveNumber>.txt
└── Recoding
    └── See section 4.2.3
```

Figure 4.6: Directory structure of the Dataset

### 4.4.5 Image sets

#### Full

The files in the directory *ImageSets* starting with *Full*, are based on the whole dataset. By using a simple random function the dataset was split up into three files. The file *FullTrain.txt* contains the filenames of 80% of the images of the dataset. The files *FullValidation.txt* and *FullTrain.txt* each contain 10% of the images of the dataset. The files contain the names of the images which are the same for ToF data and RGB data. To get the name of the corresponding label file, one has to replace the *.png* extension with an *.txt* extension. The distribution of the dataset was already shown in figure 4.5. The advantage of this dataset is, that is has nearly 100 000 labeled instances.

#### Equal

As originally intended, the dataset should have been equally distributed right after recording. To get such a distribution nevertheless, 6 000 images are randomly selected from each category. This leads to an equally distributed dataset with 60 000 labeled instances, where again 80% of the filenames are stored in *EqualTrain.txt*, and 10% each in the files *EqualValidation.txt* and *EqualTest.txt*.

| category | device | floor | | | table | | | workbench | | | # of img | # of img |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | p1 | p2 | p3 | p1 | p2 | p3 | p1 | p2 | p3 | per dev | per cat |
| Angle grinder | | | | | | | | | | | | 14220 |
| | Angle grinder1 | 466 | 428 | 451 | 658 | 347 | 306 | 409 | 395 | 424 | 3884 | |
| | Angle grinder2 | 388 | 433 | 432 | 389 | 365 | 331 | 389 | 394 | 387 | 3508 | |
| | Angle grinder3 | 429 | 391 | 407 | 345 | 364 | 368 | 355 | 367 | 375 | 3401 | |
| | Angle grinder4 | 449 | 430 | 420 | 352 | 343 | 331 | 366 | 380 | 356 | 3427 | |
| Battery power drill | | | | | | | | | | | | 9899 |
| | Battery power drill1 | 340 | 349 | 359 | 344 | 330 | 331 | 364 | 329 | 368 | 3114 | |
| | Battery power drill2 | 361 | 369 | 371 | 342 | 324 | 316 | 371 | 347 | 361 | 3162 | |
| | Battery power drill3 | 352 | 366 | 392 | 420 | 411 | 438 | 394 | 416 | 434 | 3623 | |
| Battery screw driver | | | | | | | | | | | | 6549 |
| | Battery screw driver1 | 354 | 341 | | 323 | 336 | | 372 | 339 | | 2065 | |
| | Battery screw driver2 | 373 | 391 | | 328 | 327 | | 313 | 335 | | 2067 | |
| | Battery screw driver3 | 392 | 374 | | 376 | 413 | | 461 | 401 | | 2417 | |
| Big angle grinder | | | | | | | | | | | | 6977 |
| | Big angle grinder1 | 417 | 501 | 463 | 333 | 352 | 369 | 351 | 318 | 357 | 3461 | |
| | Big angle grinder2 | 435 | 463 | 448 | 381 | 343 | 340 | 358 | 365 | 383 | 3516 | |
| Chain saw | | | | | | | | | | | | 10808 |
| | Chain saw1 | 415 | 416 | 452 | 337 | 334 | 389 | 350 | 368 | 383 | 3444 | |
| | Chain saw2 | 437 | 474 | 455 | 360 | 386 | 383 | 385 | 381 | 383 | 3644 | |
| | Chain saw3 | 525 | 462 | 436 | 366 | 368 | 396 | 418 | 359 | 390 | 3720 | |
| Hand planer | | | | | | | | | | | | 6972 |
| | Hand planer1 | 361 | 354 | 355 | 374 | 358 | 361 | 386 | 398 | 395 | 3342 | |
| | Hand planer2 | 414 | 443 | 451 | 382 | 444 | 363 | 397 | 388 | 348 | 3630 | |
| Hot glue gun | | | | | | | | | | | | 10283 |
| | Hot glue gun1 | - | 373 | 395 | 332 | 337 | 350 | 380 | 377 | 409 | 2953 | |
| | Hot glue gun2 | 349 | 423 | 402 | 373 | 386 | 414 | 413 | 568 | 455 | 3783 | |
| | Hot glue gun3 | 352 | 402 | 385 | 376 | 372 | 393 | 433 | 443 | 391 | 3547 | |

Continued on next side

| category | device | floor | | | table | | | workbench | | | # of img | # of img |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | p1 | p2 | p3 | p1 | p2 | p3 | p1 | p2 | p3 | per dev | per cat |
| Jig saw | | | | | | | | | | | | 10123 |
| | Jig saw1 | 478 | 434 | 388 | 378 | 329 | 375 | 390 | 375 | 380 | 3527 | |
| | Jig saw2 | 476 | 442 | 384 | 346 | 354 | 335 | 412 | 412 | 399 | 3560 | |
| | Jig saw3 | 411 | 457 | 415 | 349 | 347 | 330 | 377 | 350 | - | 3036 | |
| Power drill | | | | | | | | | | | | 15436 |
| | Power drill1 | 416 | 380 | | 370 | 315 | | 360 | 407 | | 2248 | |
| | Power drill2 | 323 | 346 | | 330 | 317 | | 409 | 426 | | 2151 | |
| | Power drill3 | 354 | 324 | | 334 | 380 | | 387 | 364 | | 2143 | |
| | Power drill4 | 330 | 375 | | 330 | 328 | | 371 | 379 | | 2113 | |
| | Power drill5 | 336 | 368 | | 335 | 332 | | 352 | 361 | | 2084 | |
| | Power drill6 | 353 | 344 | | 438 | 427 | | 366 | 367 | | 2295 | |
| | Power drill7 | 356 | 381 | | 415 | 450 | | 399 | 401 | | 2402 | |
| Vibration grinder | | | | | | | | | | | | 6839 |
| | Vibration grinder1 | 373 | 427 | 413 | 366 | 370 | 395 | 367 | 373 | 366 | 3450 | |
| | Vibration grinder2 | 404 | 404 | 451 | 333 | 335 | 331 | 363 | 407 | 361 | 3389 | |
| | | | | | | | | | | | Total number of pictures: | 98106 |

Table 4.9: Actual number of pictures per recording

## 4.5 Evaluation

### 4.5.1 YOLO

As a first CNN, *darknet*, an implementation of the YOLO network was used ([Ale19]). This implementation is written in *C* and uses CUDA to interact with the GPU. It is a fork of the repository maintained by the authors of the paper who first described the YOLO network ([RF18]). After setting a couple of parameters like number of classes and directory structure, which is perfectly described in the *README.txt* from the cited repository, it was ready to be trained on the dataset. For training the CNN, a google cloud instance was used, with an Nvidia P100 GPU. Training with an batch size of 64, took over night to train it on 20 000 iterations. The network is kind of over-engineered for predicting this dataset. It is designed for fast prediction on datasets where one does not know, how many objects are shown on one picture. Therefore, training took a little bit longer, however, the results are of high accuracy.

#### Data Augmentation

The *darknet* implementation has built in data augmentation. It transforms the input images in many possible ways such as rotating, sheering, cropping or brightening. It works well, however, it has no correlation to the real sensor. Especially when used with both, the ToF and RGB image, the correlation between them is destroyed.

### 4.5.2 UNET

UNET, as described in [RFB15], was used as a second approach. It was originally designed for segmentation, however, by using a softmax layer for categorization and a sigmoid layer for predicting the bounding box as a last layer, it also fitted our needs. It was implemented in python by using the Keras library. The Keras model can be found in the appendix. Training took only a few hours on the same google cloud instance as described in section 4.5.1.

#### Data augmentation

Due to the large number of instances provided by this dataset, it was not expected, that overtraining can be observed. However, while training UNET, overtraining could be observed after about 60 epochs with each consisting of 100 iterations with a batchsize of 128. After this observation, simple data augmentation was implemented. The images were rotated by $\pm 5°$. This worked very well and no overtraining could be observed again. One should mention that the data augmentation is correct when only using the ToF or the RGB data, because the rotation is simply a rotation over the optical axis of the used sensor. However, when using both images, a rotation over the optical axis on one sensor, would lead to a rotation plus a translation on the other sensor. Therefore, the used data augmentation which rotates each image over its own axis is wrong, and may influence the correlation between the two sensors a CNN may make. In this thesis, however, no

network will be used which explicitly tries to exploit this combination, therefore, this kind of data augmentation was able to be used.

Figure 4.7 and 4.8 show the data augmentation for ToF data and RGB data respectively. Figure 4.9 shows how the augmentation is done, and figure 4.10 shows how it should be done correctly.

Figure 4.7: Data augmentation ToF

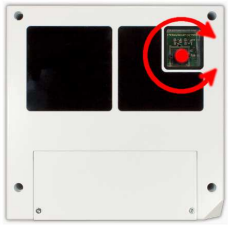


Figure 4.8: Data augmentation RGB

Figure 4.9: Data augmentation combined wrong

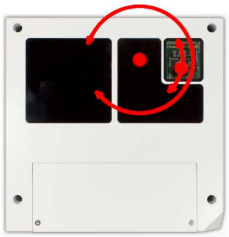


Figure 4.10: Data augmentation combined correctly

#### Normation

For training a neural network, normalization of the input data is important. For all UNET implementations, the MIN_MAX-normalization function, provided by the OpenCV library, is used to map the values of all pixels in the range from 0 to 1 and represents it as a 32bit floating point value. For the combined networks, the normalization is done for the ToF and the RGB image individually.

#### Loss

For the category prediction, the *categorical_crossentropy* function and for the bounding box prediction, the *mean_squared_error* function were used as for computing the loss. The overall loss for the network is given by $loss = 0.1 * category\_loss + 10 * bounding\_box\_loss$. The weighting was chosen this way, to compensate the small values returned by the *mean_squared_error* function. This allowed a much faster overall training, because the bounding box prediction was taken into account from the beginning.

#### Epoch

For all variants of the UNET used, one epoch is defined as 100 iterations with a batch size of 128 images. Therfore in one epoch 12 800 images are processed.

### 4.5.3 Perceptron

As a third NN, a simple perceptron was chosen. As expected, the network did not perform well. It was not even possible to let it learn anything. As a simplification, the data augmentation was turned off, and only the category output was used for training.

However, even with this setting, different learning rates, different batch sizes and different optimizers it was not possible to get more than 10%-14% accuracy out of this network. Therefore, a detailed analysis was omitted.

### 4.5.4 Network size

Table 4.10 gives a quick overview over the size of the NN. It serves as a simple metric for the complexity of it.

| | YOLO | UNET | Perceptron |
|---|---|---|---|
| Number of Trainable Weights | 61 545 895 | 7 436 048 | 2 752 526 |
| Number of Layers | 106 | 36 | 1 |

Table 4.10: Size of the trained networks

### 4.5.5 Prediction based on ToF data

Table 4.11 shows the results of the trained networks, where only the ToF images were used as input data. They were trained with the equally spread dataset as described in section 4.4.5. Figure 4.11 shows the training progress of the network.

| | YOLO | | UNET | | Perceptron | |
|---|---|---|---|---|---|---|
| | Cat | IOU | Cat | IOU | Cat | IOU |
| Train | 99.13% | 83.64% | 99.43% | 77.73% | 10%-14% | - |
| Validation | 98.79% | 82.29% | 96.57% | 76.60% | 10%-14% | - |
| Test | 98.91% | 82.40% | 96.55% | 76.35% | 10%-14% | - |

Table 4.11: Results of the trained networks with only ToF input data

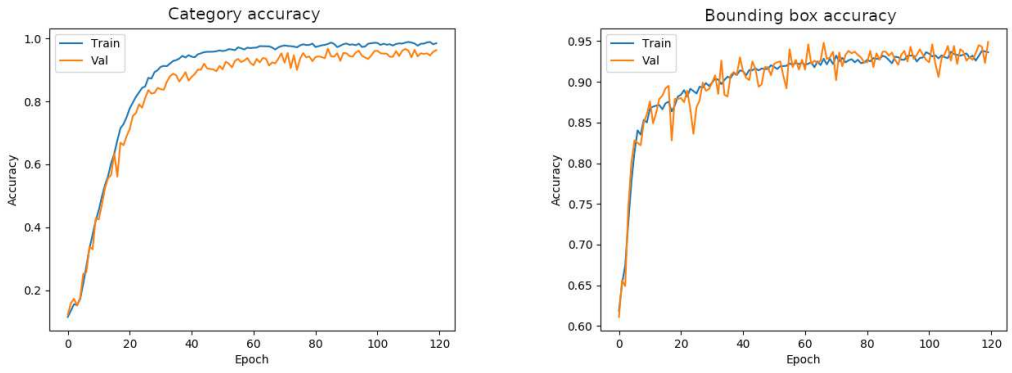



Figure 4.11: Training progress of the network (ToF as input data)

### 4.5.6 Prediction based on RGB data

Table 4.12 shows the results of the trained networks, where only the RGB images were used as input data. They were trained with the equally spread dataset as described in section 4.4.5 Figure 4.12 shows the training progress of the network.

| | YOLO | | UNET | | Perceptron | |
|---|---|---|---|---|---|---|
| | Cat | IOU | Cat | IOU | Cat | IOU |
| Train | 99.70% | 82.58% | 98.76% | 75.10% | 10%-14% | - |
| Validation | 99.46% | 81.85% | 96.00% | 73.71% | 10%-14% | - |
| Test | 99.62% | 81.58% | 95.88% | 73.77% | 10%-14% | - |

Table 4.12: Results of the trained networks with only RGB input data

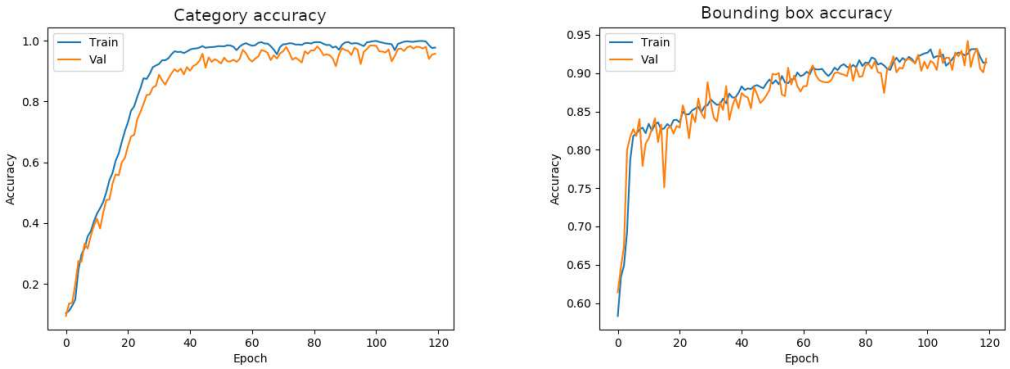



Figure 4.12: Training progress of the network (RGB as input data)

### 4.5.7 Combination of pre-trained networks

After evaluating the data generated by the ToF sensor and the RGB sensor separately, a first approach was taken to combine them. To get an idea, if there is some information hidden in the combination, an average over the predicted probabilities was used. The trained weights which were generated at training UNET as described in the last sections (4.5.5 and 4.5.6) were taken for this approach. The category is now predicted as follows: For each category, the mean value of the probability is calculated. Afterwards, the category with the highest probability is chosen. For predicting the four values of the bounding box (center x coordinate, center y coordinate, width of the box, height of the box), the average is calculated for each value individually. Therefore, no further training was necessary. Table 4.13 shows the results generated with such a network. A big disadvantage of this strategy is, that the network is now twice as big. Furthermore, this strategy did not take advantage of the existing correlation between the two input images. However, it can be shown with this approach, that there is at least some improvement possible by combining these two sensors, because all values increased in comparison to

the networks using only one image as input. Figure 4.13 visualizes the structure of this approach.

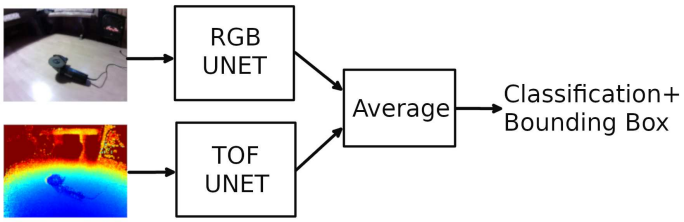



Figure 4.13: Structure of the combination of the pre-trained UNETs

| | Cat | IOU |
|---|---|---|
| Train | 99.95% | 78.82% |
| Validation | 98.97% | 77.59% |
| Test | 99.10% | 77.47% |

Table 4.13: Simple combination of two pre-trained UNETs

### 4.5.8 Train UNET on ToF and RGB data

As a final approach, UNET was trained on both images simultaneously. Therefore, both images were stacked on top of each other to get an image containing six layers. This combined image was then fed into a same structured UNET as used in section 4.5.5 and 4.5.6. Only the first layer was adapted to accept an image with 6 layers instead of 3. Table 4.14 shows the results of this approach. The results are not quite as good as the results described in section 4.5.7 where two pre-trained networks work together, however, the size of this network has not doubled in size like the network in section 4.5.7. Namely, only its first layer is doubled in size which means only an increase of 0.012% of the global size of the network. This is shown in table 4.15. For such a small increase in size, the increase in prediction accuracy, compared with the results from section 4.5.5 and 4.5.6, is remarkable. Figure 4.14 visualizes the structure of this approach. Figure 4.15 shows the training progress of the network.

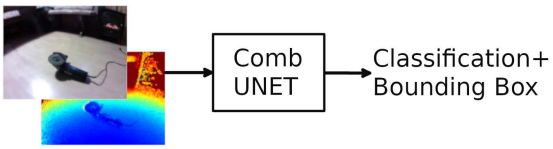



Figure 4.14: Structure of combined UNET

| | Cat | IOU |
|---|---|---|
| Train | 99.58% | 73.49% |
| Validation | 97.08% | 71.13% |
| Test | 97.40% | 71.01% |

Table 4.14: UNET trained on a combination of ToF and RGB data

| | UNET Single Image Input | UNET Combined Image Input |
|---|---|---|
| Overall size | 7 436 048 | 7 436 912 |
| Size of first layer | 896 | 1760 |
| % difference | 0.012% | |

Table 4.15: Size comparison between the UNET from section 4.5.5 and the UNET from section 4.5.8

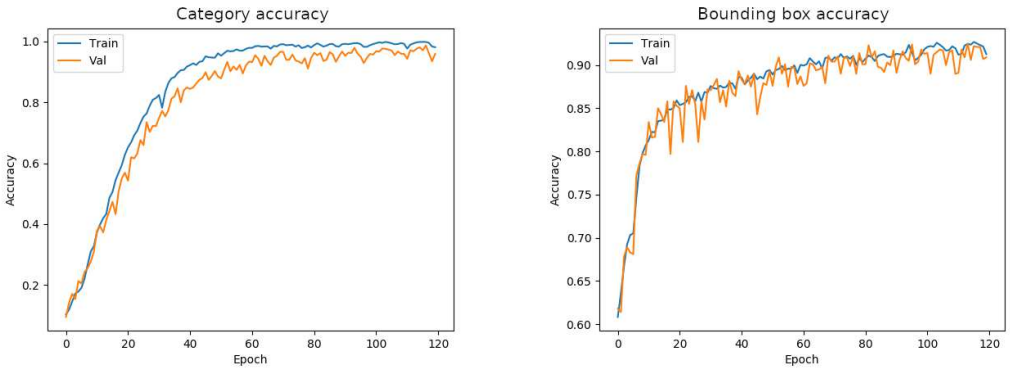



Figure 4.15: Training progress of the network (ToF and RGB as input data)

CHAPTER 5

# Critical reflection

## 5.1 Dataset

In the previous chapters, we introduced a new dataset based on ToF sensor data. It provides an equally distributed dataset with 60 000 labeled instances and overall size of nearly 100 000 labeled instances. This size is comparable with previous datasets as presented in section 2.1.2, which have between 41 000 and 80 000 labeled instances.

The number of categories provided by the datasets from section 2.1, starts with two (e.g. [GLU12]), and goes up to about 100, whereas in comparison, the dataset generated during this thesis provides 10 categories. A limitation of our dataset is, that it only provides one labeled instance per object, but at least it is guaranteed, that no other unlabeled object is present in our images. This reduces the complexity of the dataset a lot. For generating the dataset, 32 different objects, spread over the ten categories were used during recording. At least two different objects are used for each class. Whereas some existing datasets only use one object per category (e.g [MP07]), however, others contain only one instance from one object (e.g. [GLU12]).

As ground truth, a classification and a bounding box as a simple form of localization are provided in our dataset, which may be extended to provide full segmentation for all instances, however, this would have exceeded the limits of this thesis. The novelty of this thesis is, that such a big and completely labeled dataset was generated using a ToF sensor. It resolves with 120x160 pixels, which gives directly a usable input size for NNs. In comparison with many other datasets ([MP07],[MMEB18] or [MZC+19]), we tried to avoid daily life objects like mugs or fruits. Instead commonly available electrical handheld power tools were used as objects.

## 5.2 Prediction attempt

To check if the generated dataset is actually useful to design new networks which will make better use of the depth data provided by the ToF sensor, the dataset was fed to some state of the art networks. This was done by using the YOLO network and the UNET. Both of them take images as inputs. Therefore, the depth information was mapped to an image. This method provided decent results, however, it does not make full use of the depth information.

A method to overcome this issue, however this was not further analyzed during this thesis, would be to convert the depth data to a colored point cloud as explained in 3.7.1, and feed it to a 3D-CNN like Vox-Net ([MS15]). Such an attempt should make full use of the depth information.

To show, that the dataset is not too simple and provides at least some complexity, a simple perceptron was trained on the dataset. It was not possible to get any significant prediction accuracy out of it. Therefore, it can be concluded, that the generated dataset provides a certain amount of complexity.

CHAPTER 6

# Summary and future work

## 6.1 Summary

In this thesis a brand new dataset based on a ToF sensor was generated. This dataset consists of about 100 000 labeled images showing different electrical handheld power tools. 32 different objects were recorded and selected in ten categories. As additional ground truth a bounding box was manually added, to provide more complexity for prediction. To get some basic information about the dataset, the ToF sensor data and the RGB sensor data were separately fed to the YOLO network and to UNET which are both CNNs. Both sensors showed at both networks about the same accuracy at classification. However, as expected, the ToF sensor achieved slightly better results at localizing, again on both networks. To prove, that a combination of these two sensors can lead to better classification and localization, the pre-trained UNET models working on ToF data and RGB data separately, were combined through simple averaging. The simple method showed a significant improve in category prediction accuracy and slight improvement in localization. As a final approach, the ToF data and the RGB image were combined to one six layered image, and UNET was trained on this combined input data. This network performed significantly better by only increasing the size of the network marginally.

## 6.2 Future work

### 6.2.1 Extend to multi objects per instance

To extend the dataset, it would be quite easy to record similar objects on similar backgrounds and place multiple objects in one scene. A CNN like YOLO which is trained on this dataset could be used for classification. If the threshold is set low enough it will predict some of the objects. Therefore, one would not need to label the whole dataset

by hand, one only would have to remove the wrong predictions manually, which should reduce the time for generating this extended dataset drastically.

### 6.2.2 Better neural network approaches

The big networks used in this thesis had few problems to get a good prediction quality. It would be interesting, how much these networks can be reduced without loosing too much prediction quality. This threshold must exist, because it was shown in this thesis that a simple perceptron is not capable of predicting the categories nor the bounding boxes of our dataset.

### 6.2.3 Depth data prediction

Another use case for this dataset could be to only use the RGB data as input data in some kind of NN. As ground truth the depth data generated by the ToF sensor could be used. It would be interesting, if a single RGB sensor combined with a NN can predict depth data in a setting like the one given in the dataset, where only a few different backgrounds are used.

### 6.2.4 Better data augmentation

The data augmentation used for the combined approach was not correct as described in section 4.5.2. Using the provided calibration files for the ToF sensor and the RGB sensor, one can calculate the colored point cloud as described in section 3.7.1. With this colored point cloud all kinds of rotation, translation and zooming based data augmentations can be made correctly. This may help during training to get a better generalization.

# List of Figures

# List of Tables

# Glossary

**API** Application Programmable Interface. 28

**CAD** Computer-Aided Design. 7

**CNN** Convolutional Neural Network. xv, 7, 9, 11, 21, 22, 35, 42, 43, 45

**CSV** Comma Separated Value. 29

**CUDA** Computer Unified Device Architecture. 35

**FPS** Frames Per Second. 28

**GPU** graphic processing unit. 1, 35

**IOU** Intersection Over Union. 21, 37–40

**LIDAR** light detection and ranging. 1, 5

**NN** Neural Network. 2, 7, 8, 16, 17, 19, 24, 36, 37, 41, 44

**PMD** Photon Mixing Device. 11

**PNG** Portable Network Graphics. 30

**ReLU** Rectified Linear Unit. 8, 9, 45

**RGB** red green and blue. xv, 2, 3, 5, 6, 19–24, 28–30, 32, 35, 36, 38–40, 43–45, 47

**RGB-D** red green and blue plus depth data. 5

**SNR** Signal to Noise Ratio. 2

**ToF** time of flight (see section 2.3). 1, 2, 5, 11, 12, 17, 19–24, 28–30, 32, 35–45, 47

**YOLO** You Only Look Once. 2, 9, 10, 35, 37, 38, 42, 43, 45

# Bibliography


[Ale19] AlexeyAB. darknet. https://github.com/AlexeyAB/darknet, 2019.

[Ank] Ankur Handa, Andrey Kurenkov and Miles Brundage. *Indexing Datasets of 3D Indoor Objects*. https://sim2realai.github.io/Synthetic-Datasets-of-Objects-Part-I/.

[CSDR18] Changhyun Choi, Wilko Schwarting, Joseph DelPreto, and Daniela Rus. Learning object grasping for soft robot hands. *IEEE Robotics and Automation Letters*, 3(3):2370–2377, 2018.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[fre19] freedesignfile. Different power tools vector graphics. https://all-free-download.com/free-vector/download/different_power_tools_vector_graphics_524456_download.html License: Creative Common, 2019.

[GDL+17] Kent Gauen, Ryan Dailey, John Laiman, Yuxiang Zi, Nirmal Asokan, Yung-Hsiang Lu, George K Thiruvathukal, Mei-Ling Shyu, and Shu-Ching Chen. Comparison of visual datasets for machine learning. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 346–355. IEEE, 2017.

[GLU12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012.

[HM07] Dipl-Ing Bianca Hagebeuker and Product Marketing. A 3d time of flight camera for object detection. *PMD Technologies GmbH, Siegen*, 2007.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[MMEB18] Roberto Martín-Martín, Clemens Eppner, and Oliver Brock. The rbo dataset of articulated objects and interactions, 2018.

[MP07] Pierre Moreels and Pietro Perona. Evaluation of features detectors and descriptors based on 3d objects. *International journal of computer vision*, 73(3):263–284, 2007.

[MS15] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[MZC+19] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[OLK+04] Thierry Oggier, Michael Lehmann, Rolf Kaufmann, Matthias Schweizer, Michael Richter, Peter Metzler, Graham Lang, Felix Lustenberger, and Nicolas Blanc. An all-solid-state optical range camera for 3d real-time imaging with sub-centimeter depth resolution (swissranger). In *Optical Design and Engineering*, volume 5249, pages 534–545. International Society for Optics and Photonics, 2004.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RF18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RN16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[WWY+19] Yingqian Wang, Longguang Wang, Jungang Yang, Wei An, and Yulan Guo. Flickr1024: A dataset for stereo image super-resolution. *arXiv preprint arXiv:1903.06332*, 2019.

[ZJ16] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.

# Appendix

## Dataset

The whole dataset is available on GitHub: `https://github.com/TheBeMu/ToF-Dataset`

### Angle grinder




### Battery power drill

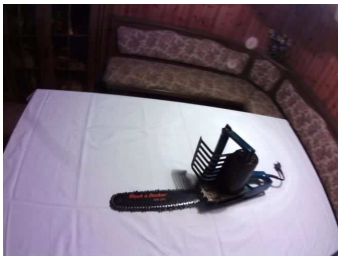


### Battery screw driver



### Big angle grinder


### Chain saw



### Hand planer


### Hot glue gun



### Jig saw



### Vibration grinder


## Power drill

## Extraction tool

```
//Based on the example.cpp file provided by BECOM Systems GmbH

#include <stdio.h>
#include <string.h>
#include <bta.h>
#include <sys/time.h>
#include <math.h>
#include <time.h>

static void mkdir(const char *subpath, const char *path);

static void *frameQueue = 0;

static void BTA_CALLCONV frameArrivedEx(BTA_Handle handle, BTA_Frame *frame) {
    BTA_Frame *frameClone;
    BTAcloneFrame(frame, &frameClone);
    BFQenqueue(frameQueue, frameClone);
    return;
}

static void storeTofData(BTA_Frame *frameToProcess,
        const char *path,int picCnt){
    char filePath[1024];
    sprintf(filePath,"%sdistance/%04d.csv",path,picCnt);
    printf("%s\n",filePath);
    FILE *dist = fopen(filePath,"w");
    sprintf(filePath,"%samplitude/%04d.csv",path,picCnt);
    printf("%s\n",filePath);
    FILE *amp = fopen(filePath,"w");

    void* distData = frameToProcess->channels[0]->data; //Distance
    void* ampData = frameToProcess->channels[1]->data;  //Amplitude

    int pixel;
    int xRes = frameToProcess->channels[0]->xRes;
    for(int y=0;y < frameToProcess->channels[0]->yRes; y++){
        for(int x=0;x < frameToProcess->channels[0]->xRes; x++){
            pixel = ((uint16_t *)distData)[x + y*xRes];
            fprintf(dist,"%d ",(int)pixel);
            pixel = ((uint16_t *)ampData)[x + y*xRes];
            fprintf(amp,"%d ",(int)pixel);
```

```
            }
        }
        fclose(dist);
        fclose(amp);
    }

    static void storeRgbData(BTA_Frame *frameToProcess,
            const char *path,int picCnt){
        char filePath[1024];
        sprintf(filePath,"%scolor_r/%04d.csv",path,picCnt);
        printf("%s\n",filePath);
        FILE *color_r = fopen(filePath,"w");
        sprintf(filePath,"%scolor_g/%04d.csv",path,picCnt);
        printf("%s\n",filePath);
        FILE *color_g = fopen(filePath,"w");
        sprintf(filePath,"%scolor_b/%04d.csv",path,picCnt);
        printf("%s\n",filePath);
        FILE *color_b = fopen(filePath,"w");

        void* colorData = frameToProcess->channels[2]->data; //Color

        int xRes = frameToProcess->channels[2]->xRes;
        for(int y=0;y < frameToProcess->channels[2]->yRes; y++){
            for(int x=0;x < xRes; x++){
                uint8_t r,g,b;
                r = ((uint8_t *)colorData)[3 * (x + y*xRes)];
                g = ((uint8_t *)colorData)[3 * (x + y*xRes) + 1];
                b = ((uint8_t *)colorData)[3 * (x + y*xRes) + 2];
                fprintf(color_r,"%d ",r);
                fprintf(color_g,"%d ",g);
                fprintf(color_b,"%d ",b);
            }
        }

        fclose(color_r);
        fclose(color_g);
        fclose(color_b);
    }

    static void processingFunction(int totalFrameCount,const char *path){
        static int picCnt=0;
        BTA_Frame *frameToProcess;
        while(BFQdequeue(frameQueue, &frameToProcess, 500)
```

```
            == BTA_StatusOk){
        picCnt ++;

        printf("\n==========================================\n");
        printf("=== New Pic ===\n");
        printf("==========================================\n");

        storeTofData(frameToProcess,path,picCnt);
        storeRgbData(frameToProcess,path,picCnt);

        BTAfreeFrame(&frameToProcess);
    }
}

int main(int argc,char* argv[]) {

    if(argc != 2){
        printf("USAGE: ./%s <Path to bltStream file>\n",argv[0]);
        return 0;
    }

    BTA_Config config;
    BTAinitConfig(&config);
    config.deviceType = BTA_DeviceTypeBltstream;
    config.bltstreamFilename = (uint8_t *)argv[1]; // (ASCII coded)
    config.frameArrivedEx = &frameArrivedEx;

    BTA_Handle handleBltstream;
    BTAopen(&config, &handleBltstream);

    float totalFrameCount;
    BTAgetLibParam(handleBltstream,
            BTA_LibParamStreamTotalFrameCount,
            &totalFrameCount);

    // Initialize frameQueue for later use with size of whole stream
    BFQinit(totalFrameCount, BTA_QueueModeDropOldest, &frameQueue);

    //Playback with max speed to fill up queue with whole stream
    BTAsetLibParam(handleBltstream,
            BTA_LibParamStreamAutoPlaybackSpeed, 10000);

    //Make path structure
```

```
    //cut away the record.bltstream
    argv[1][strlen(argv[1])-20]='\0';

    mkdir("distance/",argv[1]);
    mkdir("amplitude/",argv[1]);
    mkdir("color_r/",argv[1]);
    mkdir("color_g/",argv[1]);
    mkdir("color_b/",argv[1]);

    processingFunction(totalFrameCount,argv[1]);

    BTAclose(&handleBltstream);
}

static void mkdir(const char *subpath, const char *path){
    char cmdBuffer[1024];
    strcpy(cmdBuffer,"mkdir ");
    strcat(cmdBuffer,path);
    strcat(cmdBuffer,subpath);
    printf("Executing command: %s\n",cmdBuffer);
    system(cmdBuffer);
}
```

## Labeling tool

```
#!/usr/local/bin/python3
import numpy as np
import cv2
from os import listdir
from os.path import isfile, join
import sys
import time
import pickle
import os
import copy

def loadImage(path,y_size,x_size):
    #print("Load: " + path);
    f=open(path)
    img=f.read()
    img=list(map(int,img.split()))
    img=np.asarray(img)
    img=img.reshape(y_size,x_size)
    f.close()
    return img

def scaleValue(img,minimum,maximum):
    img=np.clip(img,minimum,maximum)
    img=img-minimum;
    img=(img/img.max())
    return img

def normImage(img):
    img = cv2.normalize(src=img, dst=None, alpha=0,
            beta=255, norm_type=cv2.NORM_MINMAX,
            dtype=cv2.CV_8UC1)
    return img

def sizeNormColor(img,scale,colormap):
    img = cv2.resize(img,(0,0),fx=scale,fy=scale)
    img = normImage(img)
    img = cv2.applyColorMap(img,colormap)
    return img

def loadDistImage(path,scale=1):
    dist=loadImage(path,120,160)
```

```
    dist=scaleValue(dist,300,1500)
    #dist=scaleValue(dist, dist.min(), dist.max())
    dist=sizeNormColor(dist,scale,cv2.COLORMAP_JET)
    return dist

def loadAmpImage(path,scale=1,autoRange=True):
    amp = loadImage(path,120,160)
    if autoRange:
        amp = scaleValue(amp,amp.min(),amp.max())
    else:
        amp = scaleValue(amp,4*amp.min(),amp.max()/4)
    amp = sizeNormColor(amp,scale,cv2.COLORMAP_BONE)
    return amp

def loadColorImage(current,scale=1):
    r = loadImage(color_r_directory+current,480,640)
    g = loadImage(color_g_directory+current,480,640)
    b = loadImage(color_b_directory+current,480,640)

    r = normImage(r)
    g = normImage(g)
    b = normImage(b)

    color_image=cv2.merge((b,g,r));
    return color_image


class bidirectionalIterator(object):
    def __init__(self, collection,deleted,labeled):
        self.collection = collection
        if deleted == []:
            self.deleted = [False] * len(collection)
        else:
            self.deleted = deleted
        if labeled == []:
            self.labeled = [((-1,-1),(-1,-1))] * len(collection)
        else:
            self.labeled = labeled

        if (len(self.collection) != len(self.deleted) or
                len(self.collection) != len(self.labeled)):
            print("Deleted Len: " + str(len(self.deleted)))
            print("Data len: " + str(len(self.collection)))
```

```
            print("Labeled len: " + str(len(self.labeled)))

            raise DataCorruptedError

            #Try to repair
            index = 0
            for data in self.collection:
                try:
                    self.deleted[index]
                except:
                    self.deleted.append(False)
                try:
                    self.labeled[index]
                except:
                    self.labeled.append([((-1,-1),(-1,-1))])

                index += 1

            print("Deleted Len: " + str(len(self.deleted)))
            print("Data len: " + str(len(self.collection)))
            print("Labeled len: " + str(len(self.labeled)))

        self.index = 0

    def current(self):
        return self.collection[self.index]

    def next(self):
        try:
            self.index += 1
            _ = self.collection[self.index]
        except IndexError:
            self.index -= 1
        return self.collection[self.index]

    def prev(self):
        self.index -= 1
        if self.index < 0:
            self.index = 0
        return self.collection[self.index]

    def delete(self):
        self.deleted[self.index] = True
```

```
    def restore(self):
        self.deleted[self.index] = False

    def getDeletedList(self):
        return self.deleted

    def getLabelList(self):
        return self.labeled

    def setLabel(self, x1,y1,x2,y2):
        self.labeled[self.index] = ((x1,y1),(x2,y2))

    def removeLabel(self):
        self.labeled[self.index] = ((-1,-1),(-1,-1))

    def isDeleted(self):
        return self.deleted[self.index]

    def getLabel(self):
        return self.labeled[self.index]

    def totalPics(self):
        return len(self.collection)

    def currentPic(self):
        return self.index + 1

    def numberOfMarkedPics(self):
        cnt = 0
        for i in range(0,len(self.collection)):
            if(self.deleted[i] == True or
                    self.labeled[i] != ((-1,-1),(-1,-1))):
                cnt += 1
        return cnt
    def isLastPic(self):
        return self.index == len(self.collection)-1

    def __iter__(self):
        return self

def crossOutImg(img):
    cv2.line(img, (0, 0),
```

```
            (img.size//(len(img)*3),len(img)), (0,0,255), 2)
    cv2.line(img, (img.size//(len(img)*3), 0),
            (0,len(img)), (0,0,255), 2)
    return img

def mouseCallback(event, x, y, flags, param):
    global currentx
    global currenty
    global drawRoi

    currentx = x
    currenty = y
    if event == cv2.EVENT_LBUTTONDOWN and drawRoi == False:
        directoryIterator.removeLabel()
        directoryIterator.setLabel(x,y,-1,-1)
        drawRoi = True
    elif event == cv2.EVENT_LBUTTONDOWN and drawRoi == True:
        drawRoi = False
        roi = directoryIterator.getLabel()
        directoryIterator.setLabel(roi[0][0],roi[0][1],x,y)


##############################################
### M A I N ##################################
##############################################

currentx = -1
currenty = -1

drawRoi = False

cv2.namedWindow("amplitude")
cv2.setMouseCallback("amplitude",mouseCallback)

if len(sys.argv) != 2:
    print("USAGE: " + sys.argv[0] + " /path/to/position/folder")
    exit();

print("Using Path: " + sys.argv[1]);

dist_directory     = sys.argv[1] + "distance/"
amp_directory      = sys.argv[1] + "amplitude/"
color_r_directory = sys.argv[1] + "color_r/"
```

```
color_g_directory = sys.argv[1] + "color_g/"
color_b_directory = sys.argv[1] + "color_b/"

files = [fil for fil in listdir(dist_directory)
        if isfile(join(dist_directory,fil))]
files.sort();

scale = 4

#for current in files:
deletedList = []
if os.path.isfile(sys.argv[1] + "deleted.conf"):
    with open(sys.argv[1]+"deleted.conf",'rb') as fp:
        deletedList = pickle.load(fp)

labeledList = []
if os.path.isfile(sys.argv[1] + "labels.conf"):
    with open(sys.argv[1]+"labels.conf",'rb') as fp:
        labeledList = pickle.load(fp)


directoryIterator = bidirectionalIterator(files,
        deletedList,labeledList);
current = directoryIterator.current()

dist = loadDistImage(dist_directory + current,scale)
amp = loadAmpImage(amp_directory + current,scale)
color_image = loadColorImage(current)

cv2.imshow('distance',dist)
cv2.moveWindow('distance',1,1);
cv2.imshow('amplitude',amp)
cv2.moveWindow('amplitude',625,1);
#cv2.imshow('color',color_image);

while(1):
    #help window
    helpWindow = np.ones((200,1260,3), np.uint8)
    helpWindow = helpWindow * 255
    cv2.putText(helpWindow,
            'Total Pics: '+str(directoryIterator.totalPics()),
            (0,25), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,0),2,cv2.LINE_AA)
    cv2.putText(helpWindow,
```

```
            'Current: '+str(directoryIterator.currentPic()),
            (0,50), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,0),2,cv2.LINE_AA)
    cv2.putText(helpWindow,
            'Already Marked: '+str(directoryIterator.numberOfMarkedPics()),
            (0,75), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,0),2,cv2.LINE_AA)
    cv2.putText(helpWindow,
            'Press: A ... brighter Pic, S ... Save, D ... mark as deleted',
            (0,125), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,0),2,cv2.LINE_AA)
    cv2.putText(helpWindow,
            '       R ... remove mark as deleted, ESC ... Save and Exit',
            (0,150), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,0),2,cv2.LINE_AA)
    cv2.putText(helpWindow,
            '       Space ... Next Pic, Backspace ... Previous Pic',
            (0,175), cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,0),2,cv2.LINE_AA)
    cv2.imshow('helpWindow',helpWindow)
    cv2.moveWindow('helpWindow',1,550);
    cv2.moveWindow('distance',1,1);
    cv2.moveWindow('amplitude',625,1);


    #Handle inputs
    kbk = cv2.waitKeyEx(33)
    if kbk == 13 or kbk == 32:
        roi = directoryIterator.getLabel()
        if(roi[1] == (-1,-1) and roi[0] != (-1,-1)):
            directoryIterator.setLabel(roi[0][0],roi[0][1],currentx,currenty)
            drawRoi = False
        current = directoryIterator.next()
        dist = loadDistImage(dist_directory + current,scale)
        amp = loadAmpImage(amp_directory + current,scale)
        color_image = loadColorImage(current)
    elif kbk == ord('e'):
        #seach for next unmarked pic
        while(1):
            current = directoryIterator.next()
            if ((directoryIterator.getLabel() == ((-1,-1),(-1,-1))) and
                directoryIterator.isDeleted() == False):
                break;
            if directoryIterator.isLastPic() == True:
                break;

        dist = loadDistImage(dist_directory + current,scale)
        amp = loadAmpImage(amp_directory + current,scale)
```

```
        color_image = loadColorImage(current)
    elif kbk == 8:
        current = directoryIterator.prev()
        dist = loadDistImage(dist_directory + current,scale)
        amp = loadAmpImage(amp_directory + current,scale)
        color_image = loadColorImage(current)
    elif kbk == ord('f'):
        if(drawRoi == False):
            directoryIterator.removeLabel()
            directoryIterator.setLabel(currentx,currenty,-1,-1)
            drawRoi = True
        else:
            roi = directoryIterator.getLabel()
            directoryIterator.setLabel(roi[0][0],
                    roi[0][1],currentx,currenty)
            drawRoi = False
            current = directoryIterator.next()
            dist = loadDistImage(dist_directory + current,scale)
            amp = loadAmpImage(amp_directory + current,scale)
            color_image = loadColorImage(current)


    if kbk == ord('r'):
        directoryIterator.restore()
    if kbk == ord('d'):
        directoryIterator.delete()


    if kbk == ord('a'):
        amp = loadAmpImage(amp_directory + current,scale,False)

    if kbk == ord('s') or kbk == 27:
        with open(sys.argv[1] + "deleted.conf", 'wb') as fp:
            pickle.dump(directoryIterator.getDeletedList(),fp)
        with open(sys.argv[1] + "labels.conf", 'wb') as fp:
            pickle.dump(directoryIterator.getLabelList(),fp)

    if kbk == ord('w'):
        head,tail = os.path.split(sys.argv[1])
        head,tail = os.path.split(head)
        head,tail = os.path.split(head)
        head,tail = os.path.split(head)
        cv2.imwrite("./" + sys.argv[1].replace("/","")+
```

```
                current.replace(".csv",".png"),color_image)
    if kbk == 27:
        break

    #draw additions

    dist_modified = copy.deepcopy(dist)
    amp_modified = copy.deepcopy(amp)
    color_image_modified = copy.deepcopy(color_image)

    roi = directoryIterator.getLabel()

    cv2.line(amp_modified,(currentx,0),(currentx,120*scale),(0,0,255),1)
    cv2.line(dist_modified,(currentx,0),(currentx,120*scale),(0,0,0),1)
    cv2.line(amp_modified,(0,currenty),(160*scale,currenty),(0,0,255),1)
    cv2.line(dist_modified,(0,currenty),(160*scale,currenty),(0,0,0),1)

    if(roi != ((-1,-1),(-1,-1))):
        #valid roi
        if(roi[1] == (-1,-1)):
            cv2.rectangle(amp_modified, roi[0],
                    (currentx,currenty), (150,255,0),2)
            cv2.rectangle(dist_modified, roi[0],
                    (currentx,currenty), (50,50,50),2)
        else:
            cv2.rectangle(amp_modified, roi[0],
                    roi[1], (0,255,0),2)
            cv2.rectangle(dist_modified, roi[0],
                    roi[1], (0,0,0),2)

    if directoryIterator.isDeleted():
        dist_modified = crossOutImg(dist_modified)
        amp_modified = crossOutImg(amp_modified)
        color_image_modified = crossOutImg(color_image_modified)


    cv2.imshow('distance',dist_modified)
    cv2.imshow('amplitude',amp_modified)
    cv2.imshow('color',color_image_modified);
```

## UNET model (Keras)

```
import numpy as np
import os
import skimage.io as io
import skimage.transform as trans
import numpy as np
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as keras


def perceptron(input_shape,pretrained_weights=None):
    inputs = Input(input_shape)

    #Downsampling
    conv1 = Conv2D(32,(3,3),activation="relu",padding="same")(inputs)
    conv1 = Conv2D(32,(3,3),activation="relu",padding="same")(conv1)
    max1 = MaxPooling2D()(conv1)

    conv2 = Conv2D(64,(3,3),activation="relu",padding="same")(max1)
    conv2 = Conv2D(64,(3,3),activation="relu",padding="same")(conv2)
    max2 = MaxPooling2D()(conv2)

    conv3 = Conv2D(128,(3,3),activation="relu",padding="same")(max2)
    conv3 = Conv2D(128,(3,3),activation="relu",padding="same")(conv3)
    max3 = MaxPooling2D()(conv3)

    conv4 = Conv2D(256,(3,3),activation="relu",padding="same")(max3)
    conv4 = Conv2D(256,(3,3),activation="relu",padding="same")(conv4)
    max4 = MaxPooling2D()(conv4)

    conv5 = Conv2D(512,(3,3),activation="relu",padding="same")(max4)
    conv5 = Conv2D(512,(3,3),activation="relu",padding="same")(conv5)
    drop5 = Dropout(0.5)(conv5)
    max5 = MaxPooling2D()(drop5)

    #Upsampling
    up6 = Conv2D(256,2, activation="relu", padding="same")(
            UpSampling2D(size=(2,2))(max5))
    merge6 = Add()([max4,up6])
```

```
conv6 = Conv2D(256,3,activation = "relu",padding="same")(merge6)
conv6 = Conv2D(256,3,activation = "relu",padding="same")(conv6)

up7 = Conv2D(128,2, activation="relu", padding="same")(
        UpSampling2D(size=(2,2))(conv6))
merge7 = Add()([max3,up7])
conv7 = Conv2D(128,3,activation = "relu",padding="same")(merge7)
conv7 = Conv2D(128,3,activation = "relu",padding="same")(conv7)

up8 = Conv2D(64,2, activation="relu", padding="same")(
        UpSampling2D(size=(2,2))(conv7))
merge8 = Add()([max2,up8])
conv8 = Conv2D(64,3,activation = "relu",padding="same")(merge8)
conv8 = Conv2D(64,3,activation = "relu",padding="same")(conv8)

up9 = Conv2D(32,2, activation="relu", padding="same")(
        UpSampling2D(size=(2,2))(conv8))
merge9 = Add()([max1,up9])
conv9 = Conv2D(32,3,activation = "relu",padding="same")(merge9)
conv9 = Conv2D(32,3,activation = "relu",padding="same")(conv9)

conv9 = Conv2D(2,3, activation = "relu", padding="same")(conv9)
flat9 = Flatten()(conv9)
categorie = Dense(10, activation = 'softmax', name="categorie")(flat9)
boundingBox = Dense(4, activation = 'sigmoid', name="boundingBox")(flat9)

model = Model(input = inputs, output = [categorie,boundingBox])

model.compile(loss={"categorie": "categorical_crossentropy",
    "boundingBox": "mean_squared_error"},
    loss_weights={"categorie": 0.1, "boundingBox": 10.0},
    optimizer = Adam(), metrics = ['accuracy'])

model.summary()

if(pretrained_weights):
    model.load_weights(pretrained_weights)

return model
```