MASTER'S THESIS 2023

Evaluation of Spot as an instrument to provide autonomous data collection

Amanda Zarkout, Johan Lindberg

ISSN 1650-2884
LU-CS-EX: 2023-21

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-21

Evaluation of Spot as an instrument to provide autonomous data collection

Utvärdering av Spot som ett instrument för autonom datainsamling

Amanda Zarkout, Johan Lindberg

Evaluation of Spot as an instrument to provide autonomous data collection

(Performing object detection using YOLO and ROS at construction sites)

Amanda Zarkout

am5425za-s@student.lu.se

Johan Lindberg

jo4875li-s@student.lu.se

June 19, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Volker Krueger, volker.krueger@cs.lth.se
Mathias Haage, mathias.haage@cognibotics.com

Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

The construction industry is one of the least digitized in the world. In this thesis, a method for automatic digitization using autonomous unmanned ground vehicles (UGVs) is utilized for frequent data gathering. A problem is handling the amount of data such a method produces. This thesis investigates the use of machine learning techniques in order to automate the analysis of gathered data. Techniques are chosen to be possible to perform online on the UGV as an onboard semantic service but also for offline data analysis.

The UGV used in this thesis was Spot from Boston Dynamics. During the thesis data was collected by Spot on a weekly basis on a construction site project at Vipan in Lund. Two program plugins for Spot were developed, allowing for data gathering and data analysis using Spot on construction sites. To enable the automatic interpretation of gathered information, the availability of Swedish construction site datasets was explored. It was found that very little such material is publicly available. The performance of two public datasets was therefore explored, as well as a pre-study on a collection of new datasets. A method of using small annotated datasets for the analysis of specific tasks was evaluated in the thesis.

Improved digitization on construction sites results in more efficient feedback for day-to-day progression. Some tasks at construction sites that today are performed by workers can be risky, repetitive, dirty, and involve heavy lifting. Using tools such as robots to automate these tasks not only makes construction sites more efficient but also creates a safer work environment.

Keywords: Boston Dynamics Spot, construction robotics, machine learning, inspection

Acknowledgements

Our gratitude goes to all the people involved in this master's thesis who made it possible. First off would we like to thank Mathias Haage (Cognibotics) and Volker Kruger (Dept. Computer Science, LTH) for their valuable support as supervisors during this time. Secondly, would we like to express a big thank you to Ola Nilsson (Cognibotics) for giving us inputs and leads on valuable information about the robot. Jarkko Erikshammar (Dept. Industrial and Environmental Construction, LTU) and Lars Stehn (Professor and Head of Subject, LTU) for their helpful knowledge about construction sites. Lastly, a big thank you to our examiner Elin Anna Topp (Dept. Computer Science, LTH) that was available for advice during this master's thesis.

Contents

Abbreviations

AP **A**verage **P**recision
mAP **m**ean **A**verage **P**recision
YOLO **Y**ou **O**nly **L**ook **O**nce
HTML **H**yper**T**ext **M**arkup **L**anguage
RGB **R**ed **G**reen **B**lue
OS **O**perating **S**ystem
SDK **S**oftware **D**evelopment **K**it
ROS **R**obot **O**perating **S**ystem
BIM **B**uilding **I**nformation **M**odel

List of Terms

RPC protocol stands for Remote Procedure Call and is a software communication protocol. It can be used to construct distributed client-server based applications.

WSL Windows Subsystem for Linux is a feature in Windows that lets you run a Linux distribution in addition to Windows desktop. One advantage over other virtual machines is that WSL can access files in windows.

Overfitting is the behavior of when a machine learning model shows great results on the data it is trained on but fails to perform at a nearly equal level on new data.

Supervised learning is a machine learning method that indicates that a network trains on data and learns from its pattern.

Chapter 1
Introduction

1.1 Motivation and background

Construction projects are prone to errors that quickly add up to large unnecessary costs and project delays. An example of these errors can be misplaced tools, materials, and instruments or even doors and windows built in the wrong place. In 2018 the Swedish authority Boverket conducted a study, requested by the Swedish government, that mapped these errors. The yearly costs were estimated to be in the range of 80 - 110 billion Swedish kronor [3], more available information for project steering and overview of the workplace could reduce some of these costs. Nowadays, construction site projects are inspected manually by humans who regularly walk around the site and inspect it, then brief the workers. This regular maintenance can introduce human errors and result in more workload. By instead using Spot, a mobile quadruped robot, these inspections could be made more systematic and autonomous and therefore be made more often. These regular inspections by Spot could hopefully detect errors sooner, thereby removing the fault handling costs and making the working process more efficient.

1.2 Purpose

There are plenty of advantages when making data gathering during inspection autonomous. Autonomous inspections enable workers to focus on other tasks instead of manually inspecting the area, making their work more efficient and reducing human error. Detecting errors sooner by replacing these manual inspections with more frequent autonomous inspections can result in cost savings and improved productivity at the construction site.

In hazardous environments, this could also improve worker safety. For clarification, the information provided by the more frequent autonomous inspections could help workers avoid places where loose objects are placed or move loosely placed objects to a less crowded area [25].

The purpose of this master thesis is to evaluate how well Spot as an instrument can be used when gathering data autonomously, as well as how image data gathered by Spot on construction sites can be used in computer vision functionalities such as object detection. This report also investigates publicly available datasets, including construction site images, to determine the mean Average Precision (mAP) for a trained model. The real-time aspect is discussed but not investigated any further (not included in the conclusion).

1.2.1 Research questions

Based on Section 1.2, the following research questions have been formulated, in order to fulfill the purpose.

- What is the best practice to use a neural network such as You Only Look Once (YOLO) on Spot?
- Can any relevant object for construction work be detected on the Swedish construction site called Vipan when using a YOLO network trained on Microsoft Common Objects in Context (MSCOCO) data?
- What publicly available datasets are suitable for training a YOLO network and analyzing image data from the Swedish construction site called Vipan?
- What is the mean Average Precision (mAP) of a YOLO network trained on data taken from a construction environment?
- What is required for a dataset when detecting relevant objects at the construction site Vipan?

1.3 Limitations

The initial scope was to decide on the relevant objects to detect and find an appropriate training dataset suitable for construction sites. A decided object detection network is used and trained with the chosen appropriate datasets. This study does not contain other methods to improve the mAP for a YOLO network such as fine-tuning hyper-parameters during training or applying other image processing methods. Worth emphasizing is that the collection of data is limited to a specific and smaller construction site area in Sweden and thereby is the performance of the trained model in a global aspect not investigated in this thesis.

1.4 Method and outline

This section describes every chapter in short. Please note that the work made during this master's thesis has overall been experimental.

Chapter 2 introduces the robot named Spot from Boston Dynamic and other relevant information such as an overview of the software utilized, the selected YOLO network, and the publicly available datasets used to train the network. The chapter ends with the relevant background about metrics which is used to compare and evaluate the trained models of our chosen network.

In Chapter 3, the method and implementation of our solution are described. This section starts with the ROS implementation followed by the selected network (YOLOv7). The chapter also presents important aspects when training and testing three different models of YOLOv7.

Experiments and results during this master's thesis are given in Chapter 4. This section describes how Spot's autowalk missions worked, as well as a frame rate experiment for Spot's hand camera. The performance score in the form of the mAP is presented for the three models, both during training and testing.

The final implementation is then presented and the research questions are discussed. The mAP scores are analyzed here for the three models.

Finally, a conclusion is made of this thesis where the research questions are answered and future work is discussed.

1.5 Related work

Some related works in the field of construction have been found in this thesis and by investing in modern technologies such as robotics, Drone Technologies (DI), the Internet of Things (IoT), and artificial intelligence (AI), construction companies can improve productivity and safety while reducing costs. Furthermore, several datasets have been developed and used to train AI models that can detect various objects on construction sites. In the academic world there has also been some research including construction site applications.

In the master's thesis "Building dense reconstructions with SLAM and Spot" [23] the writer explored dense recognition to reduce errors on construction sites with the help of the exact same model of Spot. This thesis included helpful information about the robot's different hardware parts and an insight into the working process at construction sites.

In another master's thesis "Construction Supervision with Augmented Reality" [12] the writer used partly the mobile robot Spot from Boston Dynamics for supervision tasks at construction sites with simulations in Gazebo. Gazebo is software that provides a plat-

form for simulating and testing robots in a virtual environment.

In the article "SODA: A large-scale open site object detection dataset for deep learning in construction" [9] the authors describe their building of a comprehensive image dataset for a construction site. The category selection for this dataset includes 15 selected objects and it gave us valuable information about our algorithm selection.

The article "Development of an Image Data Set of Construction Machines for Deep Learning Object Detection" [44] is an article describing a case study on developing a dataset valuable for construction sites. The dataset is called ACID and is focused on 12 different types of construction machines. To demonstrate the datasets functionality the authors trained four object detection models on the developed dataset and tested the models performance. The article also includes the methodology for collecting and curating a dataset for deep learning applications.

In the paper "Fast Personal Protective Equipment Detection for Real Construction Sites Using Deep Learning Approaches" [42] published in 2021 the authors dedicated a high-quality dataset to detect Personal Protective Equipment PPE for the goal to improve safety at construction sites. This open-source dataset includes images of six objects to detect. These objects are helmets (in four colors), persons, and vests. This paper also focuses on training and analyzing different models of YOLO with the constructed dataset they called CHV (Color Helmet and Vest).

"Internet of Things (IoT) based Smart Helmet for Construction" [25] is an article describing a helmet that is used to provide safety, management, and efficiency at a construction site. Two sensors are enabled on this smart helmet, one GPS sensor and another for gas and smoke detection. Real-time signals are constantly sent from these sensors wireless to a single computer.

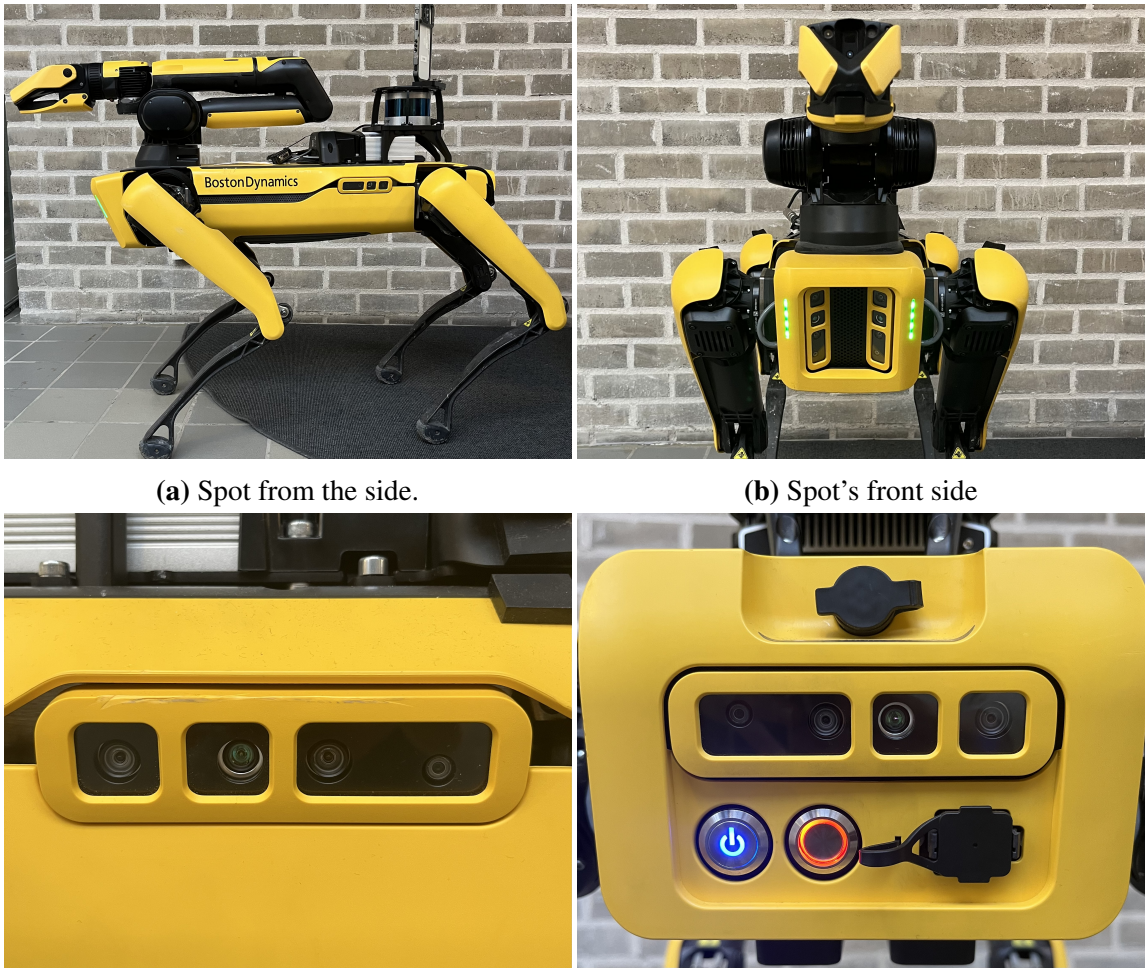
In the study "Applications of Drone Technology in Construction Industry: A Study 2012-2021 " by Gayatri Mahajan [21] the author gives a comprehensive background of the use of drones in the construction industry. For example are drones used for optimization and safety improvements. Both the evolution of drone technology and its adoption in the construction industry such as its advantages and disadvantages are explained. The impact when applying drone technology in the construction industry is also treated.

Chapter 2

Background

2.1 Spot - An introduction

Spot is a robot by Boston Dynamics. It uses five projected stereo cameras around its body and three integrated sensors in the mounted arm, a Time-of-Flight camera, an inertial measurement unit and a 4k RGB camera. Communication can be handled with both WiFi and Ethernet, allowing to collect data off and on the field. The robot has a maximal speed of 1.6 m/s. Different movement modes such as crawl, sit and walk, both in stairs and on other terrains can be adjusted on the tablet when controlling it manually. The robot is able to avoid obstacles in all modes automatically. Autowalk missions can be set by putting fiducials, nodes which will help the robot locate its position, it will then sense its way by using the depth information from the Lidar or the stereo cameras. These autowalk missions are saved as maps on the tablet and can be replayed on demand. Spot has a lithium-ion battery that is latched underneath its four legs. A fully charged battery lasts for about 90 minutes under normal operation and 180 minutes when the robot sits down [35]. It takes approximately two hours to fully charge the battery. In this master's thesis a specific model of Spot was used called Explorer.



(a) Spot from the side.

(b) Spot's front side

(c) One stereo camera from Spot's side.

(d) Power and motor button.

Figure 2.1: Figures of Spot and its hardware features.

2.1.1 Payload configuration

Payloads such as the Spot Arm and an industrial PC called Spot CORE can be mounted to better fit the customers' needs. In addition to these two payloads, a Velodyne VLP-16 LiDAR is mounted on top of the Spot CORE. Also for extra data collection, an iPhone 13 Pro is mounted on top of the Lidar, see Figure 2.2b. Detailed descriptions of the used payloads are given below. The total weight of payloads must not exceed 14kg. The robot has two DB12 payload ports on its back that supply power, communication, time-synchronization, and safety system integration.

(a) DB12 port on Spot.

(b) Payloads on Spot.

(c) Hand camera.

(d) Front cameras.

Figure 2.2: Spots cameras, payloads and port.

Spot CORE Spot CORE is a small PC used for additional computing capability, data interfaces, and network [36]. The hardware consists of an i5 Intel® 8th gen with 16GB of RAM and a 512GB SSD. The operating system installed is Ubuntu 18.04 LTS 64-bit. The CORE has an auto power setting which is turned on when the blue colored button on the robot's rear side is pressed, see Figure 2.2d. Boston Dynamics offers 3 types of Spot CORES, Spot CORE(the one used in this master thesis), Spot CORE AI, and Spot CORE I/O. Spot CORE comes with a Cockpit installation, and a web-based graphical interface for servers. Cockpit makes interacting with the CORE more intuitive by providing easily accessible information on the CPU usage, network setup, and an overview of docker images and containers, see Figure 2.3.

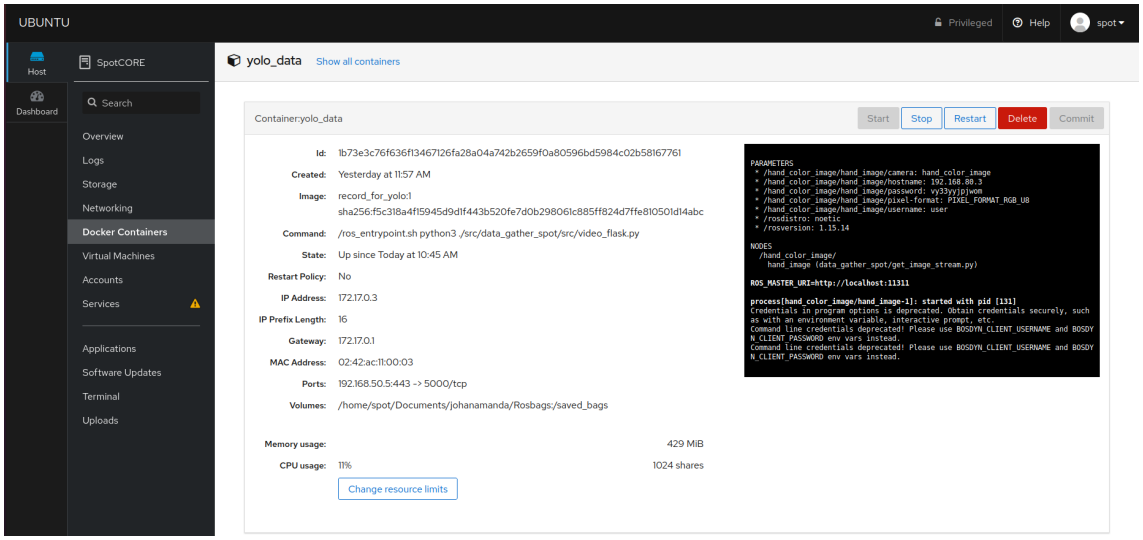


Figure 2.3: Interface for Docker in Cockpit

Spot Arm An arm with a gripper is applied to Spot. Inside the gripper Spot's three integrated sensors are placed, see Figure 2.2c). The arm has six degrees of freedom and the upper half of the gripper can be opened 90°. This lets Spot perform tasks such as opening doors and extending the reach of the robot. It also contains the only built-in color camera.

2.1.2 Cameras

Multiple cameras are available both internally and externally.

Stereo Cameras Five stereo cameras are built into the body where every stereo camera includes a "fisheye" camera and a depth camera. These cameras have a relatively short baseline and are useful for depth computation. As they only capture grayscale images they are not used in this master thesis apart from early testing.

Hand sensors The Spot hand is equipped with three sensors, an 4K RGB Max camera field of view (FOV), an Inertial Measurement Unit (IMU) and a Time of Flight-camera (ToF) [6].

IPhone camera An iPhone 13 Pro can be mounted on top of the LiDAR for extra data collection, see Figure 2.2b. The camera can record in 25-60 frames per second in both HD and 4K. In this thesis, the iPhone is set to 30 frames per second and HD. This frame rate was set due to the frame rate experiment that is presented in Section 4.

2.1.3 Software Development Kit

Boston Dynamics provides a Software Development Kit (SDK) to make it easier to develop applications and software for running directly on Spot. It is available as a Python API, the software used in this thesis relies on SDK version 3.2. The API is divided into

three main parts; core, robot, and autonomy [7]. Core handles low-level functions such as authentication which is needed to access higher-level functions. The robot part of the SDK handles for example data acquisition and basic control of the motors. The image sources are used to capture multiple images which are accessed through Spot with a single RPC(Remote Procedure Call) request [37]. The autonomy part handles higher-level calls like choreography and has not been explored in this thesis. The choreography SDK contains an API protocol and a Python client library to communicate with Spot. This includes functions like automatic docking and programmable dance routines. The interested reader is referred to [10].

2.1.4 Communication with Spot

A network connection is needed for communicating with Spot. This can be done by an Ethernet cable in the RJ-45 port or through the onboard WiFi access point. The Spot CORE is connected to Spot's network through the DB-12 payload connection. Network ports from Spot are forwarded to the Spot CORE, see Figure 2.4.

Description	Robot port	Target	Protocol
Standard Forwards 1	20000 + [22, 80, 443]	192.168.50.5:[22, 80, 443]	TCP

Figure 2.4: Some of the forwarded network ports from Spot to its payloads, Figure taken from [33]

2.2 Software

Python 3.8 was used in this master's thesis as it is the latest version supported by Boston Dynamics Python client library [34].

2.2.1 OpenCV

OpenCV is a large open source computer vision library with over 2500 optimized algorithms [4]. These algorithms range from simple ones like rotating images to more complicated ones such as corner detectors and encoders. The used Python interface OpenCV uses its own image object that stores pixel information in a format of the type numpy array [15]. OpenCV enables various types of data structured matrices. A colored image uses three 8-bit unsigned values per pixel to describe the color channels, red, green, and blue. OpenCV uses the color order BGR (Blue Green Red). The namespace *cv* is used to define the classes and functions in the OpenCV library. Examples of those functions are the *blur()* function and a Gaussian filter function. Video analysis is also a great area of use. Operations on a video such as writing, viewing, and reading are possible as well as extracting individual frames [4]. The specific version used for OpenCV in this master thesis is 4.6.0.

2.2.2 ROS

ROS 1 is used with the distribution Noetic Ninjemys. The Robot Operating System or ROS is, although its name, not really an operating system in the traditional sense but rather a communications layer above the host operating system [26]. ROS is designed with a peer-to-peer topology to avoid the necessity of a central data server which can be problematic in systems with multiple hosts. This topology is made possible with the lookup mechanism called master which lets different processes find each other at runtime. These processes are in ROS called nodes and are basically software modules that perform computations. Nodes in turn can communicate by publishing and subscribing to topics where messages are sent. Messages are strictly typed data structures where standard primitive types such as integers and booleans are supported, as well as arrays.

ROS bridge to OpenCV

ROS uses its own message format to send image data in topics. This data type is not always practical in applications using the computer vision library OpenCV as these are not compatible. ROS does provide an interface between ROS and OpenCV called CVBridge, this makes it easy to convert back and forth from ROS image messages to OpenCV images [22], see Figure 2.5.

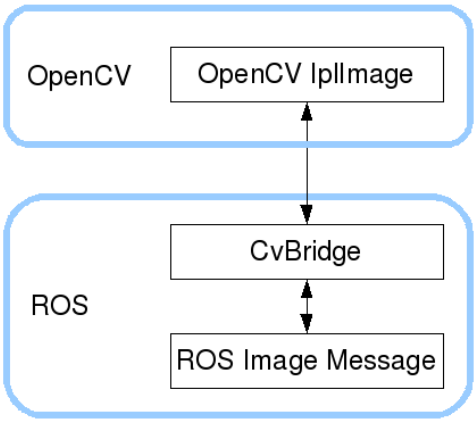


Figure 2.5: Connection between OpenCV and ROS [22].

Rosbags

The ROS package rosbag can be used with simple command-line instructions to collect data published in active ROS topics. From these rosbags the data can be extracted for later use [39]. Useful commands can be to compress/decompress bagfiles and encrypt/decrypt them. The key commands used in this master thesis are:

- **rosbag record -a** Record a bag file by subscribing to a specified topic. The flag -a means that all topics are being recorded.
- **rosbag play <bag name>.bag** Play back the contents of one or more bag files in a time-synchronized fashion.
- **rosbag info** Summarize the contents of one or more bag files. Below in Figure 2.6 is an example of what this can look like.

```
path:        2022-12-02-12-33-30.bag
version:     2.0
duration:    3:54s (234s)
start:       Dec 02 2022 13:33:30.20 (1669984410.20)
end:         Dec 02 2022 13:37:24.35 (1669984644.35)
size:        6.8 GB
messages:    8918
compression: none [7746/7746 chunks]
types:       rosgraph_msgs/Log            [acffd30cd6b6de30f120938c17c593fb]
             sensor_msgs/Image            [060021388200f6f0f447d0fcd9c64743]
             vision_msgs/Detection2DArray [402071f61477de256df9f1aa45e6e4c8]
topics:      /rosout                         5 msgs    : rosgraph_msgs/Log           (2 connections)
             /spot_hand_rgb                5951 msgs    : sensor_msgs/Image
             /yolov7/yolov7                1481 msgs    : vision_msgs/Detection2DArray
             /yolov7/yolov7/visualization  1481 msgs    : sensor_msgs/Image
```

Figure 2.6: An example output when using the command rosbag info. The recorded topics are listed at the bottom.

2.2.3 Docker

Docker is an open-source project for packaging software into isolated containers. Unlike many other virtual machines Docker, shares the Linux kernel with the host machine. Docker must therefore be based on a Linux system or subsystem such as Windows Subsystem for Linux. The four major parts of the docker used in this project are files, images, containers, and volumes. The Dockerfile provides simple instructions that define how the image should be built. These instructions are written in a syntax very similar to shell scripts in a Linux environment. This makes them relatively easy for users to both read and write [2]. Images in Docker are built on top of base images available on Docker hub, a website similar to GitHub. Conveniently, many ROS distributions are available as base images. Instructions are then given in the Dockerfile to install and set up all necessary software as well as how to copy the local directories needed for the project [43]. With the Dockerfile ready a simple command builds the image that then can start a container on the local machine or be zipped and transferred to a different host. This ability to test and later transfer makes it possible to develop on a separate machine before deployment to the intended machine [2]. Network ports can be forwarded from the host to the container, this lets the container communicate via the network. Files created in a Docker container are normally only available inside that container. By using Docker volumes it is possible to save data to a directory that is shared by the container and the host's file system [8].

2.2.4 Flask

Flask is an easy-to-use framework for developing web applications. A Flask app is started in a Python script and web browsers can then send requests. These requests can in turn activate methods within Python [13]. Web applications can be rendered using HTML from which one can send requests to the server by pressing buttons. Flask comes with a built-in server, this is not suitable for production but small-scale projects can use it to host the Flask app.

2.3 Convolutional Neural Network

CNN or Convolutional Neural Networks is a supervised deep learning algorithm. A supervised learning technique uses labeled input data and output data. The convolutional neural network is then trained on a labeled dataset (see Section 2.4 for the definition of a dataset). Based on the characteristics of new data, the CNN model will be able to classify similar data after training. An example of supervised learning can be training a CNN that after training is capable to classify objects in images into different categories, such as a dog or a cat. The architecture of a typical neural network is inspired by our human brain cells called neurons [29]. The network has a collection of these artificial neurons, also called units, see Figure 2.7. These neurons in deep neural networks such as CNN:s are often structured and combined in an input layer, an output layer, and some hidden layers, see Figure 2.8.

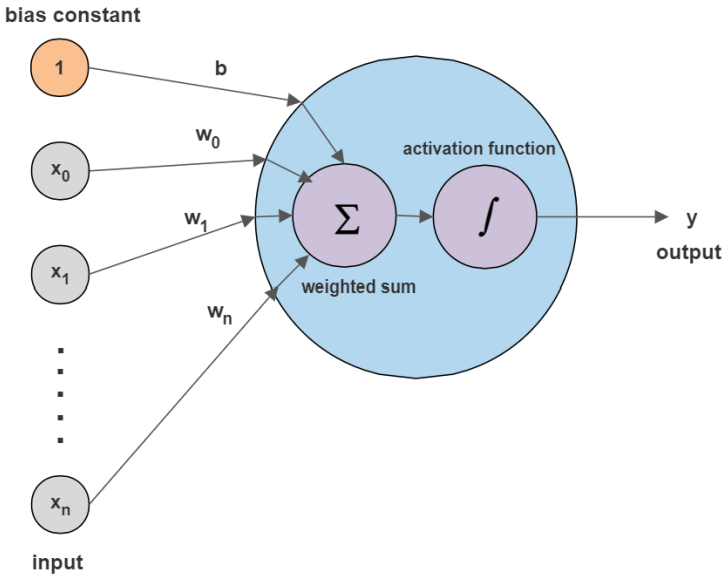


Figure 2.7: Architecture of a general unit. Highlighting input values ($x_0, x_1, ..., x_n$), bias (b), weights (w), weighted sum($\sum$), activation function ($\int$), and output signal (y). Observe that a neuron can have multiple output values pointed to other neurons.

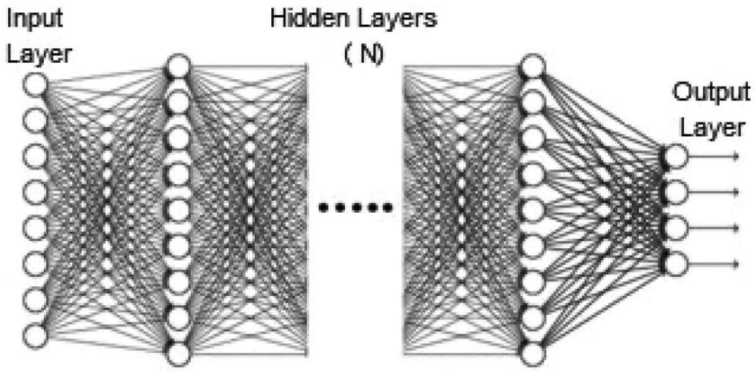


Figure 2.8: A deep neural network(Sarker, Iqbal H. p.6, Figure 5b) [30]).

A general CNN model can be split into 4 parts (these 4 parts refer to the hidden layers of the deep learning model, see Figure 2.7), convolutional layers, pooling layers, activation function, and fully connected layers, see Figure 2.8 for an overview.

Convolutional layer is used to recognize features in an image. Different filters, including weights, are used in the different layers with convolutional operations to create a so-called feature map that reflects where in the image there is an activation of local features.

Pooling layer or sub-sampling layer, is used to reduce the number of trainable parameters, which in turn reduces the computation amount by the neural network to process. Pooling also preserves spatial invariance. Max-pooling is one example of a pooling technique where the output will be downsampled, see Figure 2.9.

Activation function decides whether a neuron's input is essential or not and thereby activates it. Worth mentioning is that these neurons in the hidden layers are locally connected, meaning it only takes a patch of the image and looks at it. Examples of some of these functions are Sigmoid and Rectified Linear Unit (ReLU). ReLU introduces nonlinearity into the system and makes it more efficient under training.

Fully connected layer can be used for different tasks. For example, a fully connected layer can combine these features created by the previous layers and prepares the CNN for classification tasks [16]. The neurons in a fully connected layer are connected to all the other neurons in both the previous layer and the next layer.

In the subject of using object detection, this structure of the network dramatically reduces parameter space and optimizes the training process. Today CNNs are used in many computer vision applications such as image classification, object detection, etc. Implementations of CNN into network architectures have been widely used [16]. For example, the YOLO network is built with convolutional layers, see Figure 2.11.

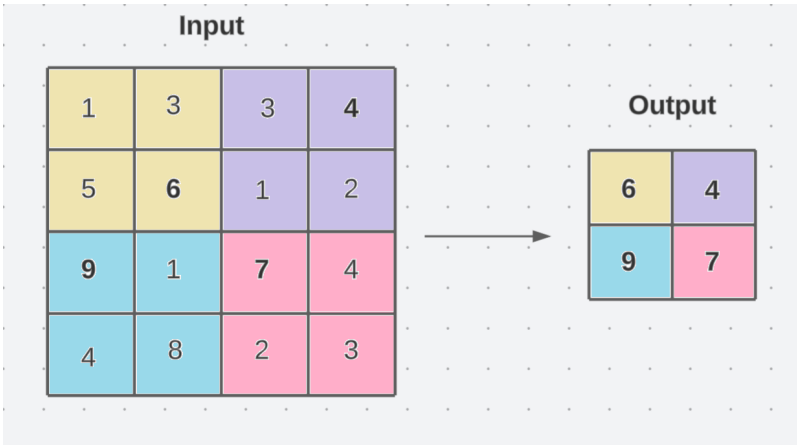


Figure 2.9: Example of the input and the resulting output using max-pooling in 2D. The operation provides a 2x2 filter with a stride of 2 where the stride size is how many steps the filters will shift over the input.

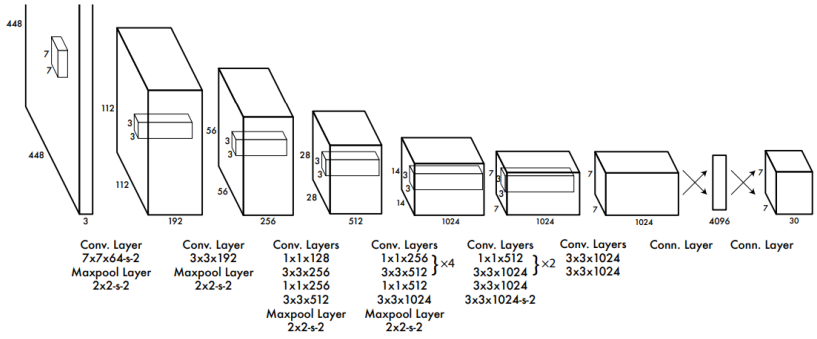


Figure 2.10: Here is a picture of YOLOv1(Redmond et al., p.3, Figure 3 [27])

2.3.1 Faster RCNN: Towards Real-Time Object Detection

Faster RCNN (Region-based Convolutional Neural Network) was a candidate when selecting an appropriate object detection model. This network is an extension of the first RCNN and can be used to perform object detection tasks. The article "Faster-RCNN: Towards Real-Time Object Detection with Region Proposal Networks" [31] describes the two-stage object detection model. The two-stages first begin with a Region Proposal Network, RPN, that proposes relevant regions in an image likely to include any object to detect. The second stage takes the information from the RPN stage and as a result, gives the necessary information to detect objects. This artificial neural network is known for its high accuracy in object detection tasks but has the disadvantage of being slower than for example YOLOv7 (see Section 2.3.2).

2.3.2 YOLO: Unified Real-Time Object Detection

From the objective of this master's thesis, a subgoal has been to perform real-time object detection using different trained models on the robot's portable PC (Personal Computer). The YOLO (You Only Look Once) model, known for its exceptional performance in real-time object detection, has been a big part of this study. This network is designed to quickly identify objects in an image and is a so-called single-stage object detector [32]. Object detection models with two stages, such as Faster R-CNN, are also available but only near real-time functionality (The interested reader is referred to this article [31] to know more about Faster R-CNN). A first-stage object detector, such as YOLO, is designed to predict all objects in one evaluation [27]. YOLO is bad at predicting objects in new environments since the data the network trains on is hard to make general. The model also makes more localization errors (compared to Fast R-CNN) but is less likely to predict false positives in the background. The model reasons globally about the images when making predictions, meaning it looks at the whole image at test time. The approach for detecting objects can be divided into 3 steps [27]:

1. Resize
2. Run convolutional network
3. Non-max suppression

To explain the steps above, the system starts with a re-scale of the images with a dimension of 428x428. The next step is to run the image through a convolutional network once before making non-max suppression to erase the multiple bounding boxes detected by the network. See the subsection below to know more about non-max suppression.

YOLO has a big family of object detection models. The model used in this master's thesis is YOLOv7 with the architecture of the tiny variant. The tiny variant has 6.1 million parameters and uses leaky ReLU as the activation function while the other variants of YOLOv7 use SiLU as the activation function [41]. As a result of fewer layers, the tiny model predicts faster but is less accurate.

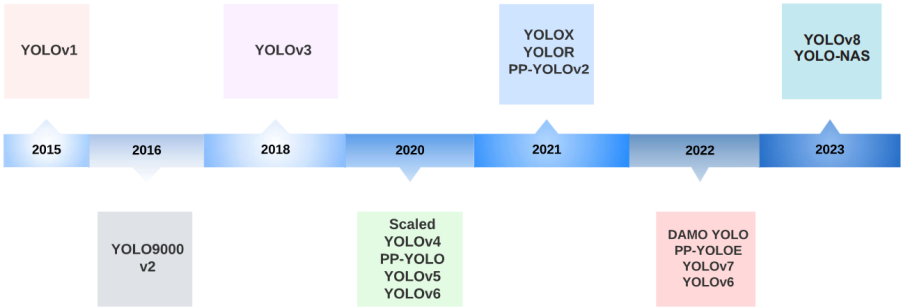


Figure 2.11: Timeline of YOLO versions. Observe that YOLO version 8 was released later in this master's thesis (Terven and Cordova-Esparaza., p.2, Figure 1 [38]).

Bounding boxes

The model uses a technique of dividing the image into grid cells to predict the bounding boxes. A demonstration of this technique is interpreted in Figure 2.12 where an example image is partitioned into a 4x4 grid. Observe that the grid size can be set by the developer and is not fixed. A grid cell is responsible for detecting and locating the object contained within it. For clarification, objects are contained in a grid cell if their centers are placed in that grid cell. The model detects an object with the help of bounding boxes and one grid cell can predict multiple bounding boxes. A bounding box is a colored box in the image enclosing the object. Figure 2.13 shows an example of bounding boxes. Each bounding box has five parameters and additional class probabilities which are provided as a vector for the model. These vectors have the same appearance as the vector given in Figure 2.12. The first five parameters include the box coordinates of its center in (x, y), the width w of the box, the height h of the box, and at last the confidence score. The confidence score is a score that tells us the accuracy of that specific class in combination with the accuracy that the box contains any object at all. Formally this score can be expressed with Equation 2.1 taken from the original article [27]. During test time the model makes use of Equation 2.2 where $Pr(Class_i|Object)$ are the conditional class probabilities [27]. Multiplying the conditional class probabilities with Equation 2.1 results in a right hand side of class-specific confidence scores calculated for a grid cell regardless of the number of predicted bounding boxes. Observe that in Equation 2.2 is the Intersection Over Union method is applied (IOU_{pred}^{truth}) when making these calculations. This method is described in more detail in Section 2.3.2. The resulting vector will have the same appearance as in Figure 2.12.

$$Pr(Object) * IOU_{pred}^{truth} \tag{2.1}$$

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \tag{2.2}$$

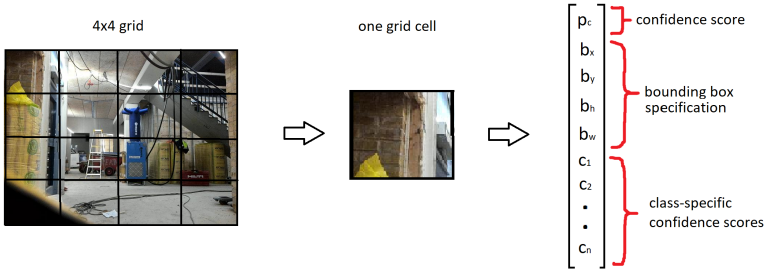


Figure 2.12: Example of one vector for a grid cell. One grid cell can have multiple bounding boxes with calculated bounding box specifications and confidence score. The class-specific confidence scores are the same for all alternative boxes.

Non-max Suppression

As discussed in the earlier section this algorithm can predict multiple bounding boxes created for one object and non-max suppression solves this problem by leaving the most appropriate and accurate bounding box as a result. The technique is the final step of the YOLO algorithm and involves filtering out the redundant bounding boxes in two steps. The non-max suppression method is applied for each category (or class) specified by the developer.

1. First, it discards all the bounding boxes with a confidence score lower than a specific threshold.

2. While there are any remaining boxes around an object, the algorithm will pick the box with the largest confidence score and set it as a prediction (also called the ground truth). The IoU (Intersection over Union) method is then applied which is a similarity measurement to compare 2 shapes against each other, in our case two bounding boxes. The IoU method is calculated by taking the intersection area I over the union area U, see Equation 2.3 [28]. The equation can also be explained as taking the overlap between two bounding boxes A and B (corresponding to letters A and B in Equation 2.3 where one of these two letters is the ground truth box) and dividing it by the total area of them both together. A high IoU value implies that 2 boxes overlap more than a lower IoU value. If the IoU for a specific box goes over a certain threshold, the technique will suppress that box. After several iterative rounds of comparison and rejection, ideally, only one bounding box will remain for an object. An example of the result can be seen in Figure 2.13. A threshold of the overlap is decided manually by the developer. A higher threshold will result in more accurate detection because the predicted bounding box will have a larger area of intersection with the ground truth box.

Figure 2.13: Example of the result using non-max suppression when detecting a car [**?**].

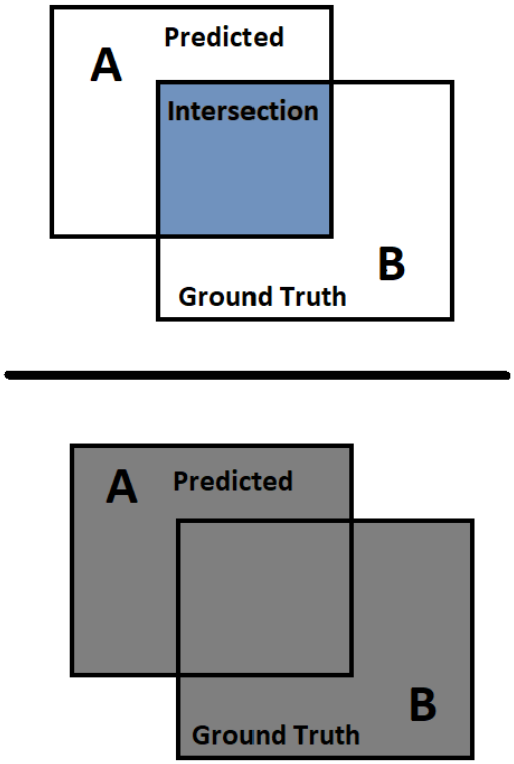


Figure 2.14: Illustration of the IoU which refers to Equation 2.3. The blue area in the numerator is the area of overlap while in the denominator, the gray area is the area of union.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{|A \cup B|}{|A \cap B|} = \frac{|I|}{|U|} \tag{2.3}$$

2.4 Datasets

Datasets are a collection of images provided for some machine learning models to learn from. A dataset is split into three parts; a training set, a validation set, and a test set. The training and validation parts can be used during the training of a deep learning model such as the YOLOv7 network. The test set is later used for testing the model with its already provided weights. A network model such as YOLOv7 can either be used in a pre-trained state or trained on a suitable dataset. A pre-trained state means that the weights are already provided and are used for the network to detect (see Section 2.3 for a definition of weights). A short summary of the datasets used in this thesis is given below.

2.4.1 MSCOCO

MSCOCO stands for Microsoft Common Objects in Context and is a dataset including 80 object types and 2,5 million labeled instances in 328K images. These images contain

everyday scenes of objects in a natural context and focus on segmenting individual object instances [20]. A per-instance segmentation technique was used to more accurately locate objects in the images. Below, the reader can find some of the annotated images in this dataset 2.15. Details can be found in [20].

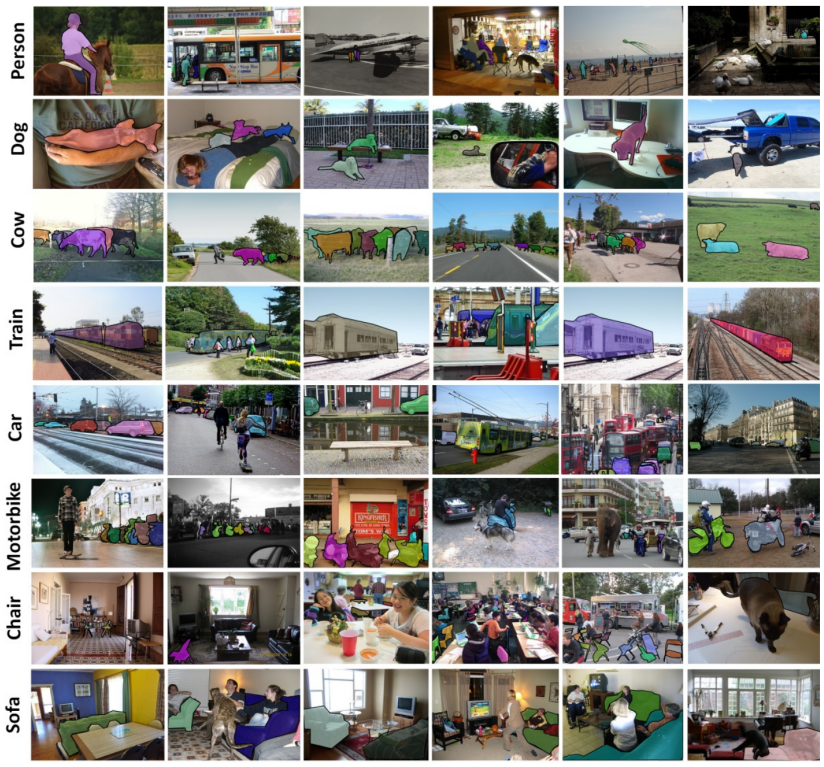


Figure 2.15: Here is a picture of some of the classes in the dataset taken from [20].

2.4.2 SODA

SODA, or Site Object Detection dAtaset, is a dataset containing images taken from construction sites in Guangzhoy, China [9]. The set includes just under 20000 images and around 286000 annotated objects. The objects are annotated using bounding boxes and the dataset is fully prepared to be used when training a YOLO network. Fifteen different objects are used as classes, examples of them can be seen in Figure 2.16. SODA is divided into 4 categories and 15 labels visible in Table 2.1.

Figure 2.16: Examples of the classes included in SODA [9].

Table 2.1: Table of the four categories and 15 objects in SODA.

Category	Labels				
Worker	person	west	helmet		
Material	brick	wood	board	rebar	scaffold
Machine	handcart	cutter	ebox	hopper	hook
Environment	fence	slogan			

2.5 GPU Accelerated computing

Since the beginning of the 21th century GPU accelerated computing has been made more and more accessible through software environments such as NVIDIA's CUDA. The big advantage of performing computations on a GPU rather than a CPU is the ability to parallelize the tasks [24]. Even though the family of YOLO networks is designed to run on GPUs, attempts have been made to create versions that function with CPUs. Both possibilities will be explored in this thesis and will be referred to as online (on the robot) and offline (on a separate computer with a GPU).

2.6 Performance measurements

Table 2.2: Table of the terms used in the formulas below.

Term	Explanation
TP	- true positives, meaning the quantity of correctly classified data
FP	- false positives, meaning the quantity of wrongly classified data
TN	- true negatives, meaning the quantity of the correctly classified non-data
FN	- false negatives, meaning the quantity of wrongly classified non-data

To understand how good predictions an artificial neural network is achieving some performance measurements will be applied. Some terms used in the formulas are explained in Table 2.2.

Precision is the accuracy of the positive predictions [14]. The positive predictions are the data with one specific label (class). An example can be looking at the classification of predicted dogs when the dataset includes all kinds of animals (cats, dogs and hamsters etc.). The formula can be expressed by:

$$precision = \frac{TP}{TP+FP} \tag{2.4}$$

Recall (also called sensitivity) is instead, explaining with the same example as above, out of all dog truths how many the classifier got right. The formula is:

$$recall = \frac{TP}{TP+FN} \tag{2.5}$$

f1-score is a combination of precision and recall (also called the harmonic mean). It gives more weight to low values and measures the balance between precision and recall. The formula is:

$$f1 - score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{TP}{TP\ \frac{FN+FP}{2}} \tag{2.6}$$

Observe that a high f1-score is given when precision and recall are similar in value to each other.

AP (Average Precision) is basically the area under the precision-recall curve (precision on the y-axis and recall on the x-axis.). This performance score is calculated class-wise.

mAP (mean Average Precision) takes the mean of all classes average precision. Some commonly used expression is mAP@0.5 and mAP@05-0.95. It indicates the decided IoU-threshold explained in Section 2.3.2.

Confusion matrices are used to illustrate how many predicted items or classes were correctly and incorrectly predicted in the form of a table. Figure 4.4 shows an example of how such a table could appear. The vertical axis presents what the model predicts and the horizontal the actual true classes. Observe that the confusion matrices in Section 4 are normalized and are given in percent ranging between [0-1] with 1 as 100%. The confusion matrices given in the result of this master's thesis have an extra column and row with background- False Negatives and False Positives which shows the performance of background objects in images both missed by the network and wrongly classified as them.

Chapter 3

Method

This chapter describes the developed software and how the different tools are used. In Figure 3.1 the general structure is shown.

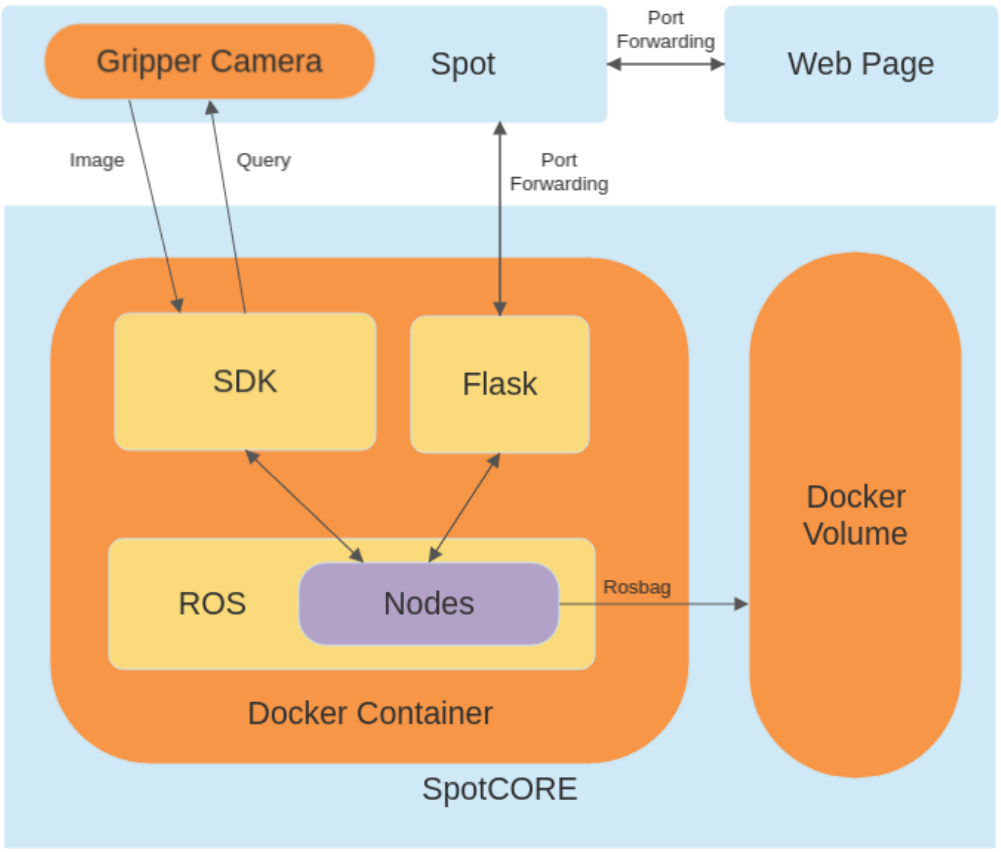


Figure 3.1: Structure of implemented software on Spot

3.1 ROS packages

Three ROS packages were developed, the Data collection package records image data from Spot's gripper camera and stores it in a rosbag. The Offline-YOLO package performs object detection on recorded data without needing access to Spot. The final Package, Online-YOLO, is deployed onto Spot CORE and performs live object detection. All python scripts mentioned in this section are included in Appendix A.2. The software is available upon request.

Data collection The package contains the Spot SDK, Flask app, and a ROS node called get_image_stream.py. This node creates a bosdyn client, robot, and image client through the SDK. Authentication credentials are passed in as arguments in the ROS launch file. Images from the hand color camera are then queried at a rate of 30 hertz and bridged from cv2 images to ROS image messages and finally published in the topic /spot_hand_rgb. The ROS structure is shown in Figure 3.2.

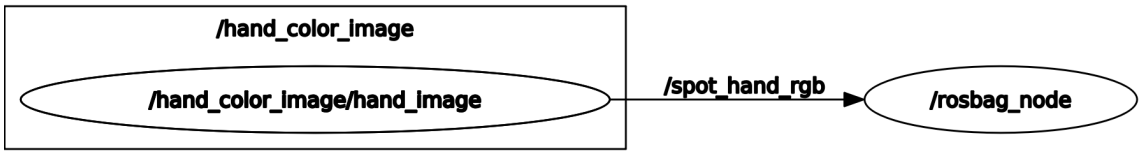


Figure 3.2: Graph of the ROS nodes and topics in the Data collection package

Offline-YOLO The purpose of this package is to perform detection on a separate computer. This is needed partly because the image data from the iPhone is not in any way connected to ROS. But it also allows data gathered by the gripper camera to be processed with a faster frame rate of 30 fps thanks to GPU-accelerated computers.

As there is no video feed from the robot in this case the data needs to be published by other methods. For data gathered using the Data collection package this is done by simply playing the rosbag. Video files from the iPhone require an additional python script. We called this script pub_video.py, it uses the OpenCV function *VideoCapture* to read the video file frame by frame. Each frame is then published in the ROS topic /spot_hand_rgb.

The YOLO network node is based on a git project [11] where most changes were made in the launch file that starts the YOLO script (detect_ros.py). This node subscribes to /spot_hand_rgb and performs the detection according to the theory in Section 2.3. Images with added bounding boxes are then published in a new topic called /yolov7/yolov7/visualization.

To view or save the new images two additional Python scripts were written, one which displays images (show.py), and another that saves them to an mp4 file (save.py). Both of these scripts subscribe to the topic published by YOLO.

Online-YOLO The final package is made to be deployed on Spot CORE and performs the detection live. It is a combination of the first two packages. Images are queried from the SDK and published as in the Data collection package, the YOLO node described in the Offline-YOLO package performs the detection. This results in two topics where image messages are being published, both are recorded in a rosbag, see Figure 3.3.

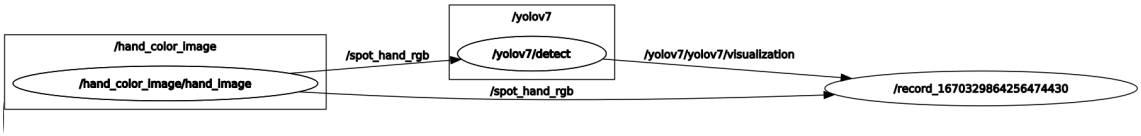


Figure 3.3: Graph of ROS nodes and topics in the online package

3.2 Docker

Docker was used to develop and test the software on a separate machine as deploying to Spot after each change would be very time-consuming. The Docker images were built using a ROS-noetic base image that includes Ubuntu 20.04 as well as instructions on how the ROS workspace should be setup. The image also comes with Python 3.8.10 installed, onto which all necessary Python packages, including Spot SDK, were installed using pip. Observe that the Python library in the SDK is used to collect sensor data from Spot. A network port was exposed to make communication with the Flask app possible. A Python script that starts the Flask app and hosts it through the built in server was set as the standard command.

As both the Data collection and Online-YOLO package are to be deployed onto the Spot CORE, Docker images are built for each package. The specific commands used to deploy the software can be found in Appendix D.

Human machine interface - Flask app

A Flask app was included in each docker container to simplify the control when using software on the robot. Making the app accessible to other devices on the network requires a public IP address. As the Flask app is hosted inside the docker container which does not have its own public IP, network ports were forwarded, see Table D.1. Shortcuts to these IP addresses are saved on the tablet to make them easily accessible.

Data collection The Data collection app shown in Figure 3.4 has two pages with one button on each page. Clicking the green Start button launches get_image_stream.py and creates a rosbag that records data from all Ros topics. Clicking the red stop button will kill and save the Rosbag using the current time as name and store it in the Docker volume, it also kills the data collection. By clicking the button on one page the other page is loaded, this makes it possible to record multiple Rosbags.

Online-YOLO This Flask app functions mostly in the same way as the first. When pressing Start Rosbag in Figure 3.5a data collection, the YOLO node and a Rosbag are started and data is recorded. Stop in Figure 3.5b saves the Rosbag and the home page is loaded. Detect Photo button takes one photo and uses Yolo to detect objects and then shows the result directly in the Flask app, see Figure 3.6.

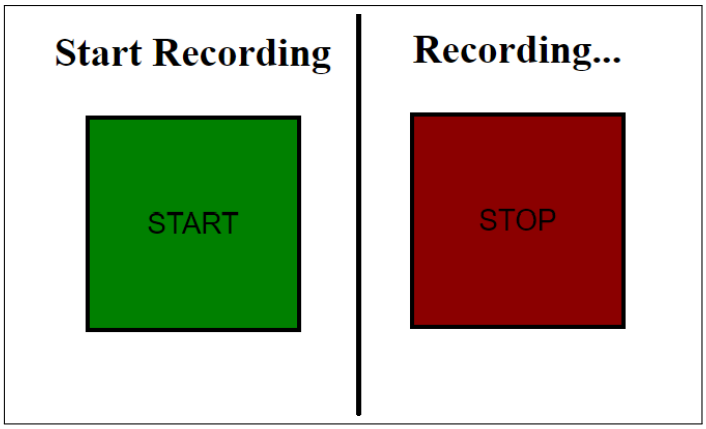


Figure 3.4: Flask app in the Data collection container

(a) Home page with the two options

(b) Button to stop the recording

Figure 3.5: Flask app in the Online-YOLO container

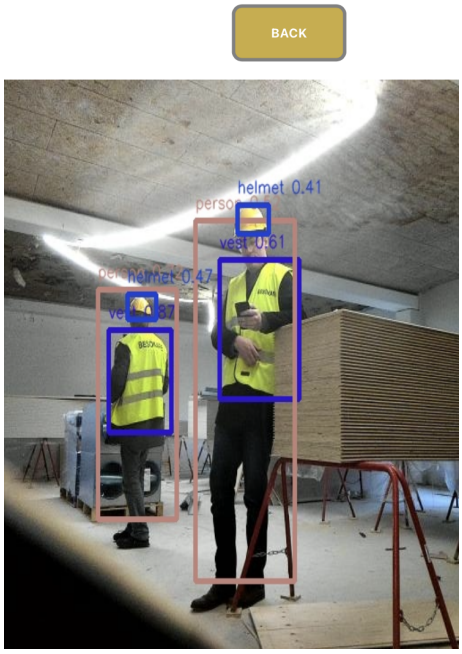


Figure 3.6: A photo taken by Spot's hand camera, viewed on the Flask app.

3.3 The selected YOLOv7 network

As informed in the background was YOLO the most appropriate decision for this master's thesis. This is because the real-time performance was an aim of this study and therefore was speed essential before accuracy. The Faster RCNN network did have greater detection accuracy compared to the YOLO network but lacked in speed instead. The chosen

version 7 of YOLO was decided because it was the version with the highest accuracy and speed according to its article [41].

All training and offline testing of YOLO was done on a desktop PC with an NVIDIA GeForce RTX 3070 graphic card.

3.3.1 Selection of Datasets

As mentioned in Section 2.4, the two publicly available datasets used are MSCOCO and SODA. MSCOCO is a staple in object detection and is often used to test these kind of neural networks. As MSCOCO does not include many classes found on construction sites we needed to find a more suitable dataset. When searching for such datasets the requirements were:

- Objects must be labeled with bounding boxes in such a way that a YOLO network can be trained
- The majority of images must come from construction sites
- The annotated objects should be found at Vipan

Google was used with the search terms "construction site dataset", "yolo dataset construction site" and "yolo dataset construction material". The different datasets are shown in Table 3.1. All these datasets fulfill the first two requirements. After studying the categories and images in the datasets it was clear that SODA included the most objects that could be found at Vipan. Additionally it was of special interest to detect material at the construction site according to experts in the field from LTU.

Table 3.1: Construction datasets annotated for YOLO

Dataset	Categories	Images	Classes
SODA [9]	Worker, Material, Machine and Environment	19846	15
CHV [42]	Color helmet and Vest	1330	6
ACID [44]	Large Construction Machines, Vehicles	10000	10
MOCS [45]	Large Construction Machines, Vehicles	41668	13

To be able to test the SODA dataset on images from Vipan, a custom dataset needed to be annotated. This same dataset was also used to train a YOLO network that was evaluated.

3.3.2 Annotate data

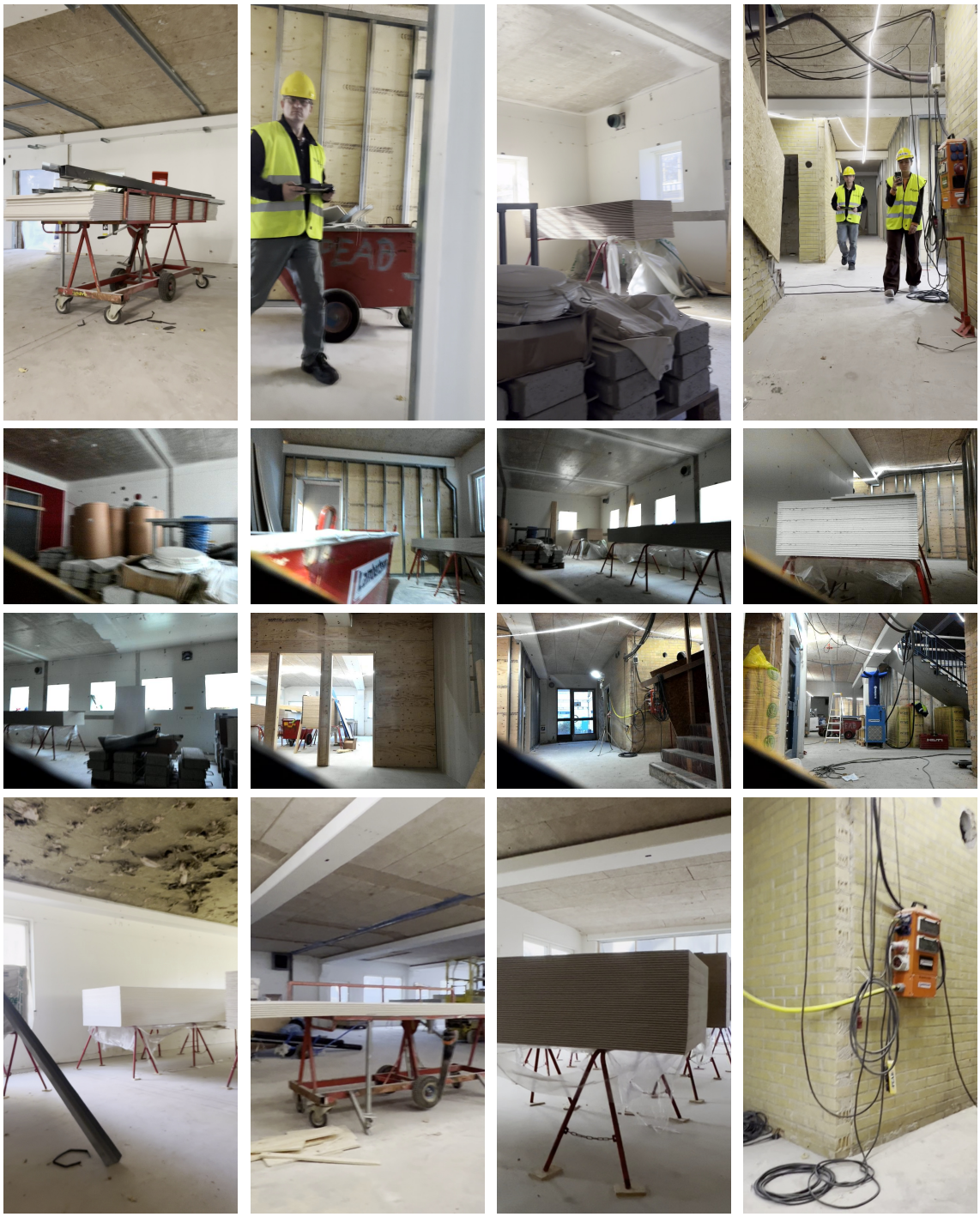


Figure 3.7: Example images of the test data at Vipan.

To annotate data, 148 pictures were selected from four rosbags and two iPhone videos which were extracted into images. Further annotation of 50 images was provided for testing the networks against each other (Vipan and SODA weighted networks). The extracted iPhone pictures were in JPG format and the rosbags were extracted in PNG format. Worth

mentioning is that both the rosbags and iPhone videos were selected on different weeks and thereby in various stages of the project at the construction site building. A drawback that comes from taking all data from a small area such as Vipan is that the variation in the scenery is reduced. This leads to a not so general network that will not perform well on data taken at a different site. To increase the general performance of the network, images were selected with variations in the following areas; content, time period, angles of objects, and lighting conditions. For a more generalized dataset, this list of rules could include different phases of construction, image sources, and time of day. See Figure 3.7 for example images of these. The program used to annotate was a git repository and can be found here [17]. See Figure 3.9 for an example of how the boxes were drawn.

The rules for annotating the data are mostly taken from the SODA article [9]:

- The boxes were drawn so that they enclose the whole object. An example can be found in the upper left three smaller pictures in Figure 3.8.
- Objects that are not visible or covered with at least 50% on the images were not annotated. An example of this kind of scenario can be found in Figure 3.8 to the right. The boxes that are red are not supposed to be annotated but the blue one is correctly annotated.
- Objects that are too blurry are not included as an annotation. The reason that the blurry images are excluded depends on the difficulty to annotate objects but also on the training effect of the deep learning model.
- Objects of the same class that are put in the same area but separated in for example boxes are not annotated as one object. See the example in Figure 3.8 in the two left images.

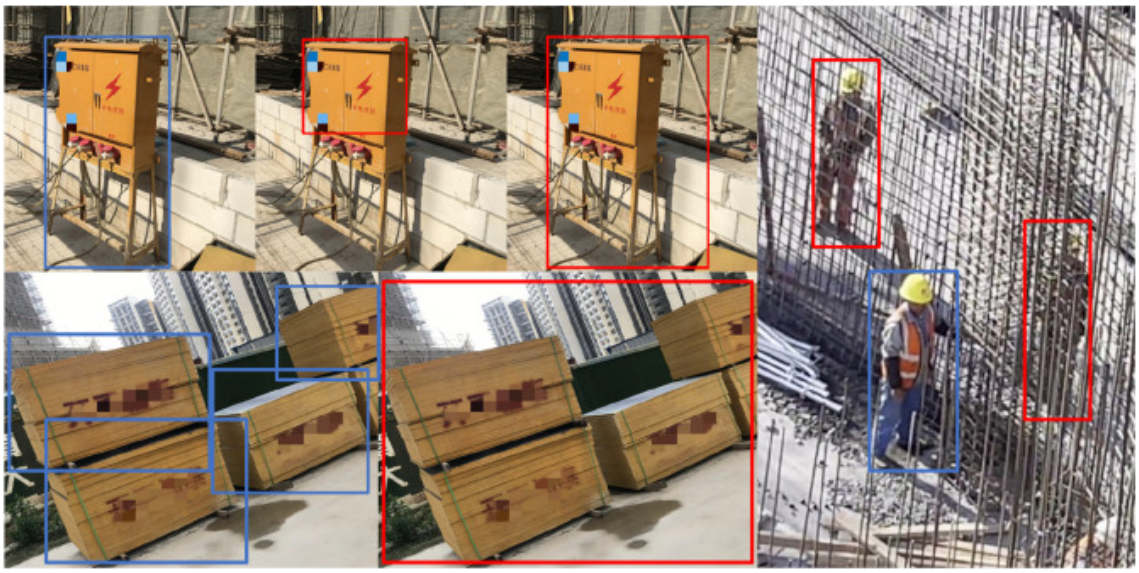


Figure 3.8: Example of annotation standard taken from SODA article (Duan et al., p.7, Figure 7) [9].

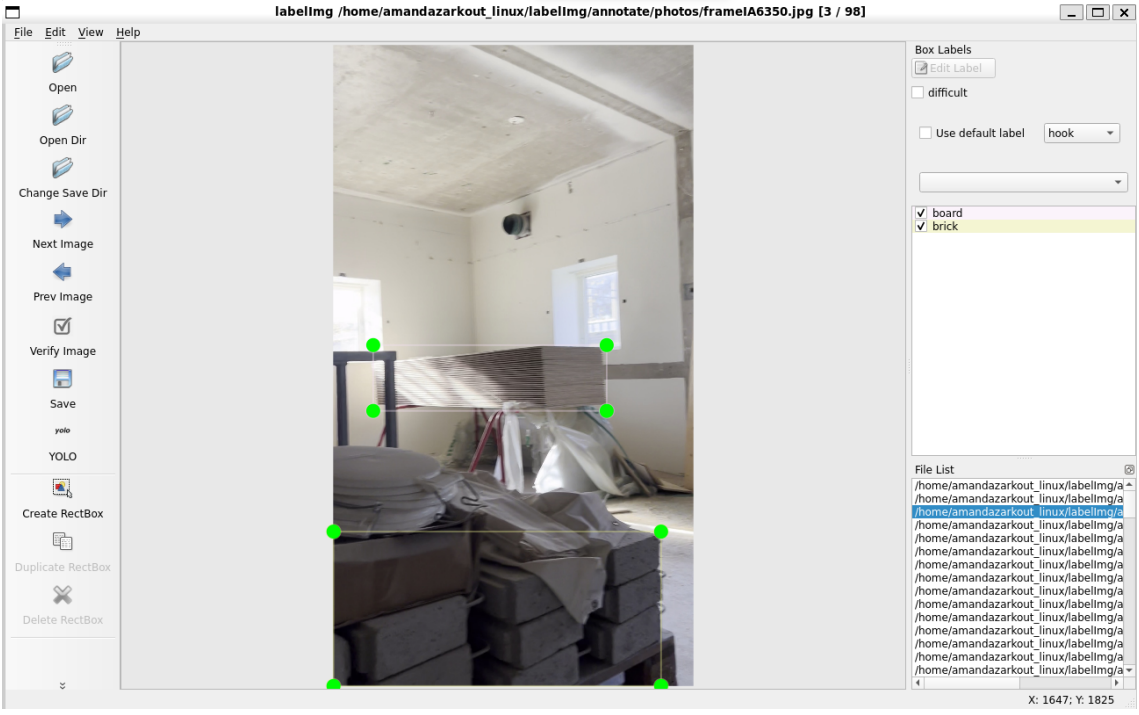


Figure 3.9: Example of the labelImg program. The text files created after every box were chosen in YOLO and the classes were fixed in a specific order.

3.3.3 Training

Two networks were trained during this master's thesis, the first using a dataset called SODA and a second on a custom dataset. Both used the tiny model of YOLO as this performs better on CPUs. Table 4.4 and 4.5 were given after running *train.py*, the commands can be found in Appendix E.

Custom dataset was later used for training after providing annotations to test the SODA network on our own data at Vipan. The difference is the changed batch size (looking at the amount of data of 148 versus the SODA dataset of around 20 000 images).

When training the custom dataset 148 images were annotated and divided into 123 training images and 25 validation images. Below is the division of the annotated categories.

Table 3.2: Table of the division of labels in validation data.

Class	Labels
All	49
Person	4
Helmet	4
Vest	4
Scaffold	0
Fence	0
Board	18
Wood	4
Rebar	3
Ebox	3
Hopper	5
Brick	4
Handcart	0
Hook	0
Slogan	0
Cutter	0

3.3.4 Test

Testing both the SODA dataset and custom dataset was done with *test.py* from the git repository [11]. These tests were done on 50 different annotated images from Vipan. Observe that the mAP results are dependent on the IOU threshold which is set to 0.65. From *test.py* were Table 4.6 and 4.7 provided and given in Section 4.

Test results with data only taken from the construction site at Vipan. See Table 3.3 for the division of labels in the test data.

Table 3.3: Table of the division of labels in test data.

Class	Labels
All	131
Person	23
Helmet	18
Vest	22
Scaffold	3
Fence	5
Board	19
Wood	14
Rebar	8
Ebox	3
Hopper	8
Brick	5
Handcart	3
Hook	0
Slogan	0
Cutter	0

Chapter 4
Experiments and Results

4.1 Autowalk missions

As described in Sect. 2.1 the autowalk missions are recorded and saved as maps on Spot. A hardware limit restricted the use of the arm during autowalk missions for the Explorer model of Spot. This restriction included the image data stream from the RGB camera. Due to this, all data gathered was done during the manual control of Spot.

4.2 Frame rate experiments

In early tests, it was noticed that the frame rate fluctuated when the recording was done at too high a rate. It was necessary to find a reliable frame rate that produced the same frame rate set in Python. This was tested by creating three Docker images with the Data collection package inside. Rosbags were then recorded and the frame rate was calculated by dividing the number of frames by the duration. Table 4.1 shows the result. Furthermore, the decided frame rate was set to 30 frames per second (fps) in all future Docker images.

Table 4.1: Tested frame rates and resulting fps

Set fps	Duration (s)	Frames	Resulting Fps
15	216	3249	15.0
30	85	2563	30.2
45	176	6501	36.9

Further tests were done to ensure that the robot's movement and actions did not impact the fps. Rosbags were recorded using the Docker image set to 30 fps.

Table 4.2: Resulting frame rates when Spot performs different actions

State	Duration (s)	Frames	Resulting Fps
Motor off	54	1614	29.9
Stand	40.8	1218	29.9
Walk around	32.6	967	29.7
Arm manipulation	116	3462	29.8

Table 4.3 shows four additional frame rates. The first and second are calculated directly from rosbags recorded using the Online-YOLO package. The online detection on Spot's CPU managed to perform inference at around six fps. After looking at the recorded videos and with the speed that Spot moves, it is doubtful that objects are missed. A bigger problem is the actual detection of objects, this will be discussed further in later sections. While YOLO is running on Spot the image capture rate was lowered by a small amount. Test three and four were made with GPU acceleration on a separate desktop. All four tests were done using the YOLOv7 model trained on the SODA dataset.

Table 4.3: Table of experiments both offline and online using YOLO. YOLO with GPU is done offline and YOLO on Spot and data capture is done online.

Situation	Duration (s)	Frames	Resulting Fps
YOLO on Spot	680	4304	6.33
Data capture during YOLO	680	18927	27.8
YOLO on GPU (rosbag)	74	2222	30.0
YOLO on GPU (iPhone)	362	10861	30.0

4.3 Data collection at Vipan

Some examples of the collected data can be seen in Figure 4.1 and Figure 4.2. Due to the shakiness in the arm and iPhone on Spot when moving, some images were blurry. During the manual data collection Spot moved in different angles to get different perspectives (or varied data) of the relevant objects at the construction site.

Figure 4.1: Images taken by the RGB camera in the gripper on Spot.

.

Figure 4.2: Images taken by the iPhone mounted on Spot.

4.3.1 Building a dataset

In total, just under 200 images were annotated. All images were collected passively and did not add any extra working hours. Due to the small variance in images at Vipan it was difficult to find a wide range of different scenes. This led to around eight hours of work in finding somewhat unique images. Annotating objects also took around eight hours.

4.4 Offline detection

The offline detection and initial tests were done with the third package covered in Section 3.1. MP4 files recorded with an iPhone were used as initial tests for the YOLO network. With a GPU this detection was performed without any problems at a speed of 30 fps. As the network at this point is trained using MSCOCO the objects detected included for example people, chairs, tables and potted plants, see Figure 4.3 below.

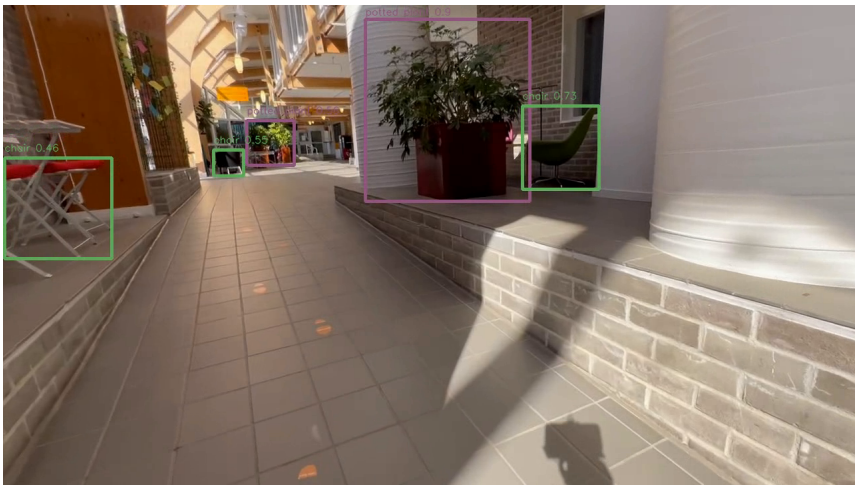


Figure 4.3: Screenshot from the first video with detected objects.

4.5 Performance of YOLOv7

As stated in Section 3.3.3, two neural networks were trained, one utilizing the SODA dataset and the other using the custom dataset (the custom dataset includes data from Vipan). Additionally, another model was initialized with pre-trained weights of MSCOCO. In Section 4.5.1 the training outcomes are presented followed by the testing results in Section 4.5.2. Metrics such as precision, recall, and mAP were utilized to evaluate the performance of the models. Additionally, confusion matrices were used as a performance measurement. These confusion matrices have normalized values ranging between zero and one. The values in the confusion matrices will be interpreted as percentages where one cell value in the matrix represents the percentage of an instance classified as a particular class when viewed from the y-axis.

4.5.1 Training

In the subsections below the confusion matrices and tables obtained after training a YOLOv7 model on two datasets, the SODA dataset and the Vipan dataset are presented. It begins with the confusion matrix generated by training the model on SODA and then moves on to the interpretation of the customized dataset, Vipan. Each model was trained for 300 epochs where an epoch is referred to as a complete iteration of the complete training data.

YOLOv7 Model Trained with the SODA Dataset

The confusion matrix is visible in Figure 4.4 and the results presented in this matrix are highly promising. The high percentage values observed along the diagonal, ranging between 70-90%, indicate that the model performs well in accurately predicting true objects. The *hook* detection stands out as a particularly strong example of this with a remarkable 98% in identifying the true *hooks* in the images. In nearly all the classes, both the false negative (FN) and false positive (FP) values for the background are low. Having a small amount of FN and FP is positive because that indicates a low rate of wrongly classified objects, meaning that the model in a small amount missed classes to detect and also did not detect anything else that was not that class.

The results from Table 4.4 state that the YOLOv7 model achieved an overall precision of 58.2%, recall of 70.3%, a mAP@.5 of 63.3% and a mAP@.5:.95 of 35.2%. The object/class *hook* dominated all metrics for its competitive classes. The lowest precision value lies at 36.2% and belonged to the *brick* class. The *fence* class has the lowest recall value at 48.6%.

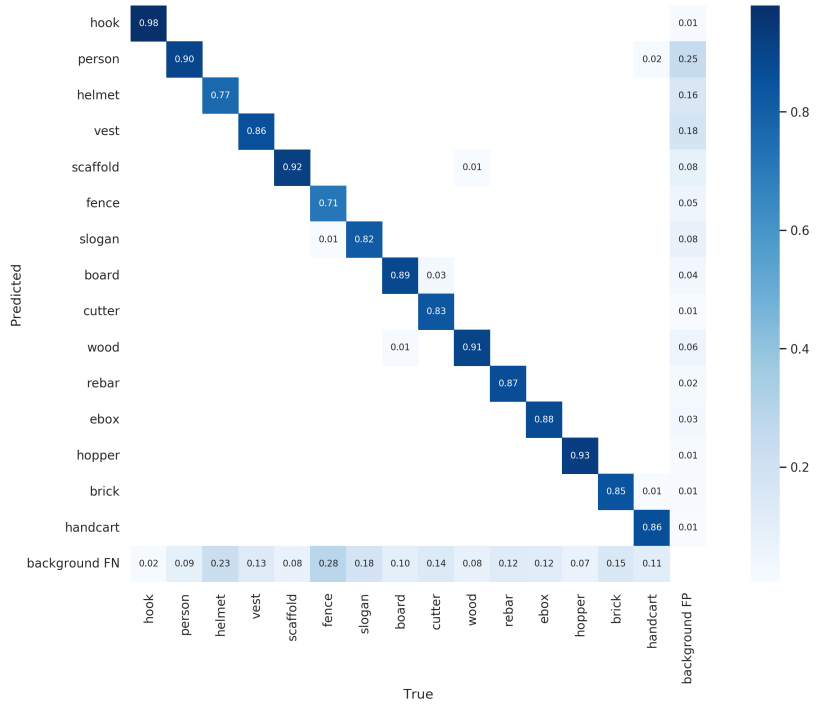


Figure 4.4: The resulting confusion matrix after training the network with the SODA dataset. Observe that the background FN and FP are included here.

Table 4.4: Result of the training data for the model trained with the SODA dataset.

Class	Precision	Recall	mAP@.5	mAP@.5:.95
all	0.582	0.703	0.633	0.352
hook	0.866	0.936	0.932	0.555
person	0.671	0.708	0.693	0.33
helmet	0.668	0.535	0.57	0.235
vest	0.645	0.636	0.647	0.312
scaffold	0.53	0.787	0.681	0.423
fence	0.483	0.486	0.447	0.224
slogan	0.535	0.591	0.548	0.267
board	0.616	0.826	0.745	0.477
cutter	0.48	0.671	0.517	0.273
wood	0.526	0.804	0.699	0.412
rebar	0.629	0.666	0.7	0.492
ebox	0.513	0.693	0.559	0.238
hopper	0.503	0.805	0.671	0.431
brick	0.362	0.582	0.415	0.252
handcart	0.696	0.815	0.672	0.355

YOLOv7 Model Trained with the Vipan Dataset

The confusion matrix is visible in Figure 4.5. Compared to the model trained on the SODA dataset this one is not equally promising. Although it has very encouraging accuracy for objects such as *brick*, *board*, *vest*, *helmet*, and *person*, it fails to detect certain classes that are never visible, such as *scaffold*, *fence*, and *slogan*. The background false negative (FN) is also calculated very high for the *wood* (75%) and *rebar* (100%) which indicates that the model is bad at predicting these classes.

The results from Table 4.5 indicate that the YOLOv7 model achieved an overall precision of 49.5%, recall of 63.6%, a mAP@.5 of 64.1% and a mAP@.5:.95 of 27.5%. The object/class *person* and *helmet* took the lead in the precision score meanwhile the recall was better for the classes *ebox* (referring to an electrical box) and *vests*. The *rebar* got zero percent of all the metrics. Observe that not all 15 classes of SODA are visible in the table and that depends on the training set not including every class.

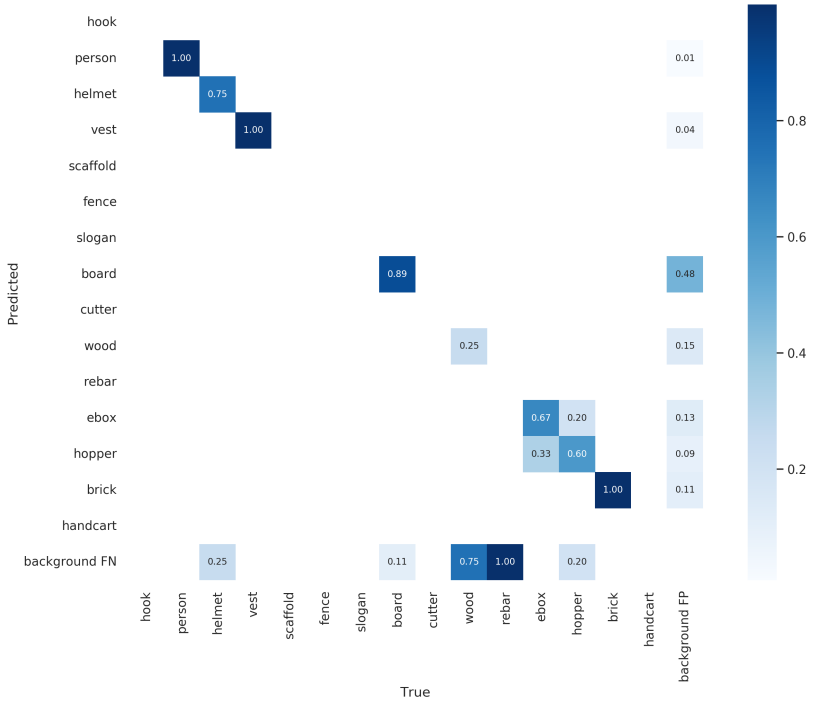


Figure 4.5: The resulting confusion matrix of a network trained on the Vipan dataset with the SODA classes. Observe that the background FN and FP is included here.

Table 4.5: Result of the training data for the model trained on the Vipan dataset.

Class	Precision	Recall	mAP@.5	mAP@.5:.95
all	0.495	0.636	0.641	0.275
person	1	0.75	0.995	0.335
helmet	1	0.702	0.995	0.414
vest	0.773	1	0.995	0.64
board	0.37	0.722	0.41	0.0939
wood	0.213	0.25	0.25	0.148
rebar	0	0	0	0
ebox	0.332	1	0.641	0.236
hopper	0.459	0.8	0.834	0.389
brick	0.306	0.5	0.649	0.219

4.5.2 Test

The three different trained models of YOLOv7 were tested on data collected at Vipan. The test data is the same for all the models to get the best comparison between them. In the YOLOv7 model trained with Vipan data, the training and test data were quite equal due to the limited area at the construction site. Both confusion matrices and tables including metrics are included in this section.

YOLOv7 Model Pre-trained with the MSCOCO Dataset

The YOLOv7 model pre-trained on MSCOCO was not used much in this master's thesis but was explored with data from Vipan. Persons and bottles were the most common objects detected by the pre-trained network, however, laptops also occasionally appeared. See Figure 4.6 for example.

Figure 4.6: Example of a YOLOv7 model detecting objects in an image from Vipan.

YOLOv7 Model Trained with the SODA Dataset

The confusion matrix is presented in Figure 4.9. Observe the high score of background FN of the classes *board*, *wood*, *rebar*, *ebox*, *hopper*, *brick*, and *handcart*, which mean that the model missed detecting these true objects in the test images. Also, one more thing to keep in mind is that *scaffold*, *fence*, *slogan*, and *cutter* do not have any values. The reason for this behavior is likely due to the fact that some classes in the test data have not been annotated at all or in a very small amount, which partly explains the difference in performance for testing and training. The model's predictions for the classes *person*, *helmet*, and *vest* show high percentages around 90%. These high percentage values in the diagonal (for the classes *person*, *helmet*, and *vest*) indicate that the model is good at detecting these classes. Observe that the *person* class has high background false positive (FP) and that indicates that the model wrongly detects persons in a large amount. An example of an FP of the category person is visible in Figure 4.7. In some scenarios, a vest can make the network predict a person even if there is not one there. As many of the persons in SODA have vests on them this results in a bias against persons without vests on.

Figure 4.8 shows when the model is detecting two unseen images. The first image (Figure 4.8a) displays the detection of boards, which is correctly identified with boxes that are accurately placed, although the confidence score of the detected box in the middle is lower. In the second image (Figure 4.8b) the model detects both persons, helmets, and vests but fails to recognize the handcart. The not detected handcart can be explained by the article on the SODA dataset showing that it is one of the classes with a low occurrence rate when producing the dataset (meaning there are not many images of a handcart in this dataset) [9].

The results from Table 4.6 indicate that the YOLOv7 model achieved an overall precision of 77.6%, recall of 22.8%, a mAP@.5 of 26.4% and a mAP@.5:.95 of 13.8%.

Having a high precision and a low recall usually indicates that the model is predicting correctly but in a small proportion of the positive outcomes (positive outcomes are all the annotated objects in the images of the test data). This behavior was also noticed when observing the video material during this thesis. The classes that have all the metrics in the table set to zero are *scaffold* and *ebox* and that explains the zero value of truth positives (TP).

Figure 4.7: Example of an detection on Spot with the YOLOv7 model trained with the SODA dataset. The model fails to detect a person and detects a board with a vest as a person instead.

(a) Image with the detected categories board and wood

(b) Image with the detected categories helmet, vest, person and handcart

Figure 4.8: Two example images when the network detects.

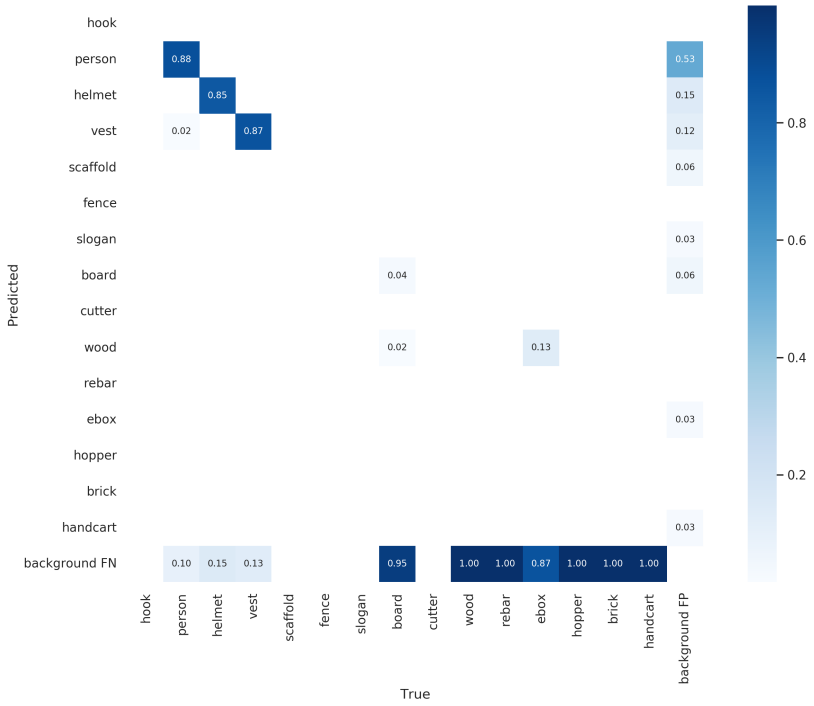


Figure 4.9: The resulting confusion matrix tested on Vipan data. This result is from the YOLOv7 model trained with the SODA dataset.

Table 4.6: Result of the test data. This result is from the YOLOv7 model trained on the SODA dataset.

Class	Precision	Recall	mAP@.5	mAP@.5:.95
all	0.776	0.228	0.264	0.138
person	0.905	0.826	0.926	0.468
helmet	0.83	0.889	0.841	0.359
vest	1	0.863	0.981	0.648
scaffold	0	0	0	0
fence	1	0	0	0
board	0.575	0.158	0.192	0.101
wood	1	0	0.0984	0.0504
rebar	1	0	0.125	0.025
ebox	0	0	0	0
hopper	1	0	0	0
brick	1	0	0	0
handcart	1	0	0	0

YOLOv7 Model Trained with the Vipan Dataset

When evaluating the confusion matrix in Figure 4.12 these are high values at the diagonal for *person*, *helmet*, and *vest*. High values of background false negatives (FN) are seen vis-

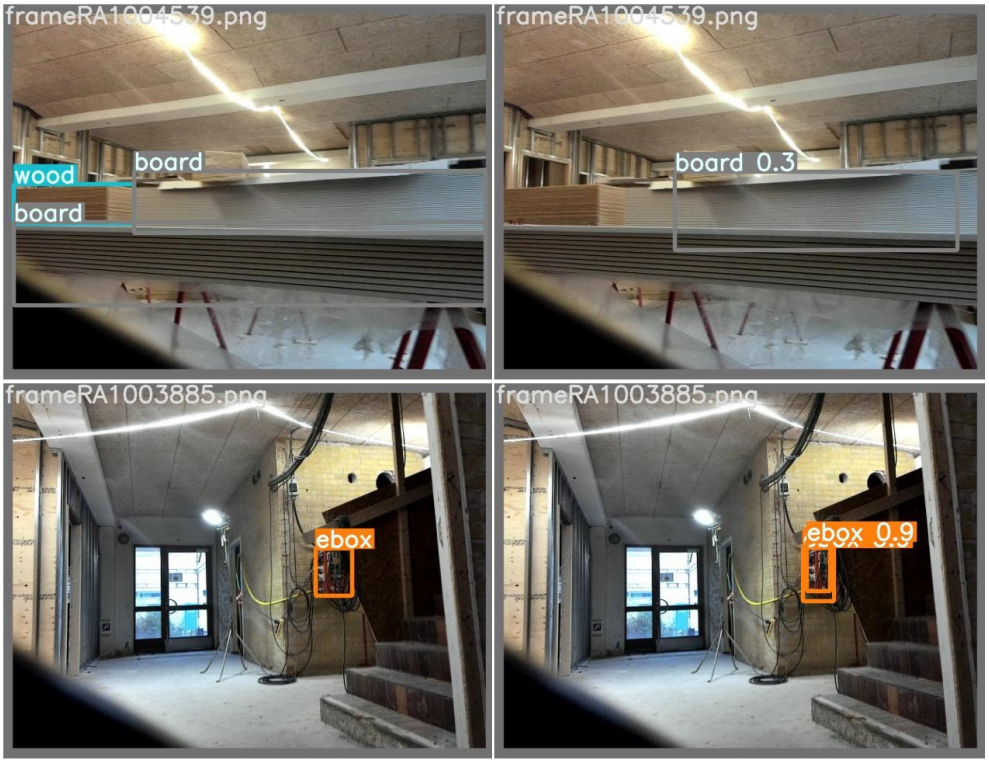
ible in the matrix for *scaffold* and *fence*. Good performance is to be seen for the classes *board*, *wood*, *hopper*, and *brick* where the normalized value lies within 50-60% at the diagonal (which means that the predicted objects are actually the true objects). Classes such as *hook*, *slogan*, *cutter* have no visibility in the confusion matrix due to no annotations at all in the test and the training data.

The images in Figure 4.11 are exactly the same as in Figure 4.8. Figure 4.11a shows an example of when the model detects multiple bounding boxes around one board. Observe that in the left-hand side corner of Figure 4.11b the model is detecting a board where it is not any object but at the same time handles to detect persons, helmets, and vests with appropriate bounding boxes. It was expected that the handcart was not detected in Figure 4.11b due to the training set including a few handcart labels, which also explains the 100% FN for the handcart in the confusion matrix in Figure 4.12.

In Table 4.7 the performance measurements are visible and the model achieved an overall precision score of 52.2%, a recall score of 43.9%, a mAP score of 29.3% at an IoU threshold of 0.5, and a mAP score of 10.1% at IoU thresholds between 0.5 to 0.95. Observe that *scaffold*, *fence*, and *handcart* have a precision of one and a recall of zero. The highest precision value of all classes is the *person* class and the class *vest* has the highest recall score.

In Figures 4.8 and 4.11 the model trained on SODA and our dataset have performed inference on the same images. This shows that the model trained on Vipan data can distinguish between the two quite similar classes *board* and *wood*, while the model trained on SODA does not.

While given the model trained on Vipan data, it is not unexpected that it performed better on the test data based on the mAP@.5 score. An important notice is that the test data come from the exact same environment as the data that were used to train the model. Even though the overall mAP@.5 score in Table 4.6 and 4.7 are quite similar the performance on the material classes is higher in our dataset. The large difference in training and testing scores indicates that the model (trained on data at Vipan) is somewhat overfitting.



(a) Image labeled before training

(b) Image with detected bounding boxes

Figure 4.10: Overview of the sensor system as worn by the subjects.

(a) Image with categories board and wood

(b) Image with the categories helmet, vest, person and handcart

Figure 4.11: Two example images when the network detects.

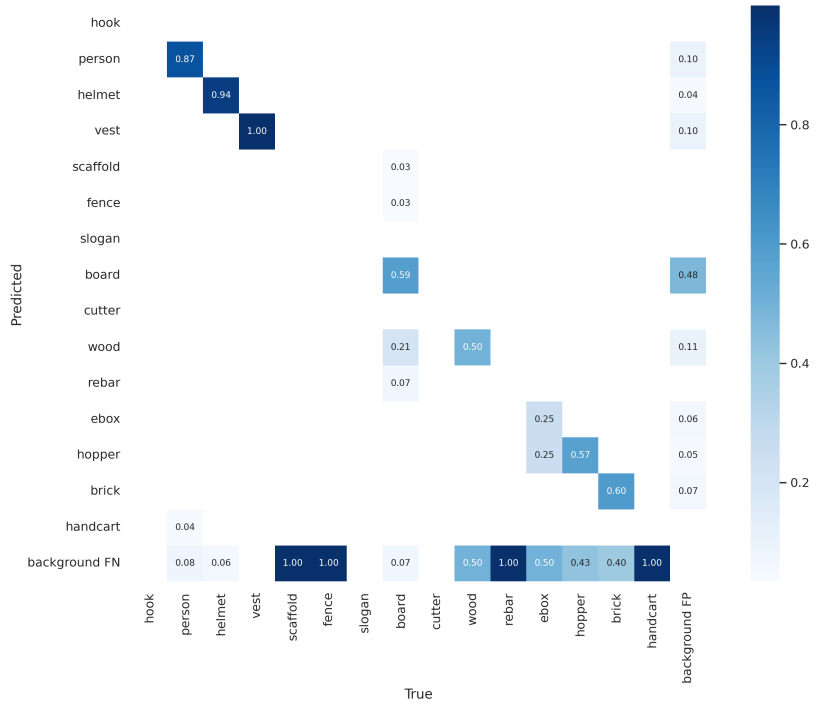


Figure 4.12: The resulting confusion matrix of a network trained on the custom dataset and tested on the custom dataset from Vipan.

Table 4.7: Result of the test data at Vipan. This result is from the YOLOv7 model trained with the Vipan dataset.

Class	Precision	Recall	mAP@.5	mAP@.5:.95
all	0.522	0.439	0.293	0.101
person	0.558	0.769	0.678	0.246
helmet	0.491	0.804	0.596	0.204
vest	0.434	0.955	0.753	0.375
scaffold	1	0	0	0
fence	1	0	0	0
board	0.102	0.895	0.206	0.0578
wood	0.146	0.286	0.158	0.0506
rebar	1	0	0	0
ebox	0.06	0.333	0.335	0.0335
hopper	0.348	0.625	0.429	0.196
brick	0.12	0.6	0.36	0.0535
handcart	1	0	0	0

4.6 Construction datasets

Looking at the construction site datasets in Table 3.1 they mostly focus on large machines and vehicles. Only SODA includes any type of material, it does however mostly contain images from outdoors. Using Spot for data collection is most likely to be done in indoor environments as other means of collection are more suited outdoors. This shows the need for a dataset specialized in material from indoor environments. Making a custom dataset can be costly, we invested 16 hours in creating our small set, and creating SODA took around 1860 working hours. It does however have the advantage of full control over images, objects, and size. This control makes it possible to customize the dataset to better suit the construction project.

Chapter 5

Discussion

5.1 Autonomous data collection using Spot

The compatibility issue with the arm and autowalk limits the way Spot can be used to collect data today (Section 4.1). The limitation is set by Boston Dynamics on the version of Spot used in this thesis. This can be solved in a few different ways. There is a more powerful model of Spot, called Enterprise, that does not restrict the use of the arm during autowalks. Data collection can also be done with another camera source that is not dependent on the arm. For example, the newer models of Spot are equipped with RGB cameras around its body.

Spot in combination with ROS has the large advantage that messages sent are easily time synced with messages from other ROS packages, for example geolocation. This would allow adding the detected objects in a 3D environment and comparing them to a project plan.

Spot with the Online-YOLO package achieved a frame rate of six fps (see Table 4.3). When comparing to another research using object detection and YOLO with CPU they accomplished 7.7 fps (with another CPU) [40]. Comparing the 7.7 fps from the article with the six fps achieved in this master's thesis their results are close to each other. Lastly, it is worth implying that the connection between fps and real-time can vary depending on the application's characteristics, meaning that real-time is decided depending on the detecting objects [19]. For example, having an application that has the goal of detecting materials at a construction site that is standing still requires fewer fps than an application that wants to detect falling objects fast. On the other hand, a higher fps results in more collected data, which can be valuable as well.

5.2 YOLOv7 on Spot data: A Model Evaluation

Three models were tested on the data collected at Vipan and the significant results that were given in Section 4.5 are discussed.

Model trained with the MSCOCO dataset

The model pre-trained on the MSCOCO dataset was tested on Vipan data as mentioned in Section 4.5.2. As a result, most of the 80 classes are not visible at all on the construction site. Objects such as materials and machines on a construction site are not included in the MSCOCO dataset and thereby the behavior is expected. The only somewhat relevant class detected by MSCOCO at Vipan was persons. It is possible that MSCOCO would be more suited for outdoor construction sites as it can detect cars and trucks.

Model trained with the SODA dataset

The classes that gave very bad results, such as *board*, *wood*, *rebar*, *ebox* (electrical box), *hopper*, *brick*, and *handcart* can be explained partly by looking at Figure 5.1. In SODA, for instance, the handcart has bigger wheels than the ones in Sweden and the ebox has yellow color and a different shape.

Another point is the reliability of this test data captured at the construction site Vipan, mostly because this test data at Vipan is too small in comparison to the SODA dataset and does not have a variety of classes included. This is partly explained by looking at Table 3.3, some of the classes are labeled three times or not at all. Therefore it would be of interest to expand this dataset even more and with a better variation. The low number of labels in the test data can therefore explain why some classes have a precision of one and a recall of zero. The precision of one describes that these classes have a low rate of FP. But on the other hand, a recall of zero indicates a high rate of FN.

Comparing Figure 5.1 with Figure 3.7 some observations can be made. The images in the SODA dataset have the majority of images taken outside and at bigger construction site projects with different equipment (drones, camera images taken by humans, etc.). The test data includes one specific project which results in a not so general test of how the SODA network actually works on a variety of construction site projects in Sweden. Maybe the SODA network would perform better at another construction site project in a different environment.

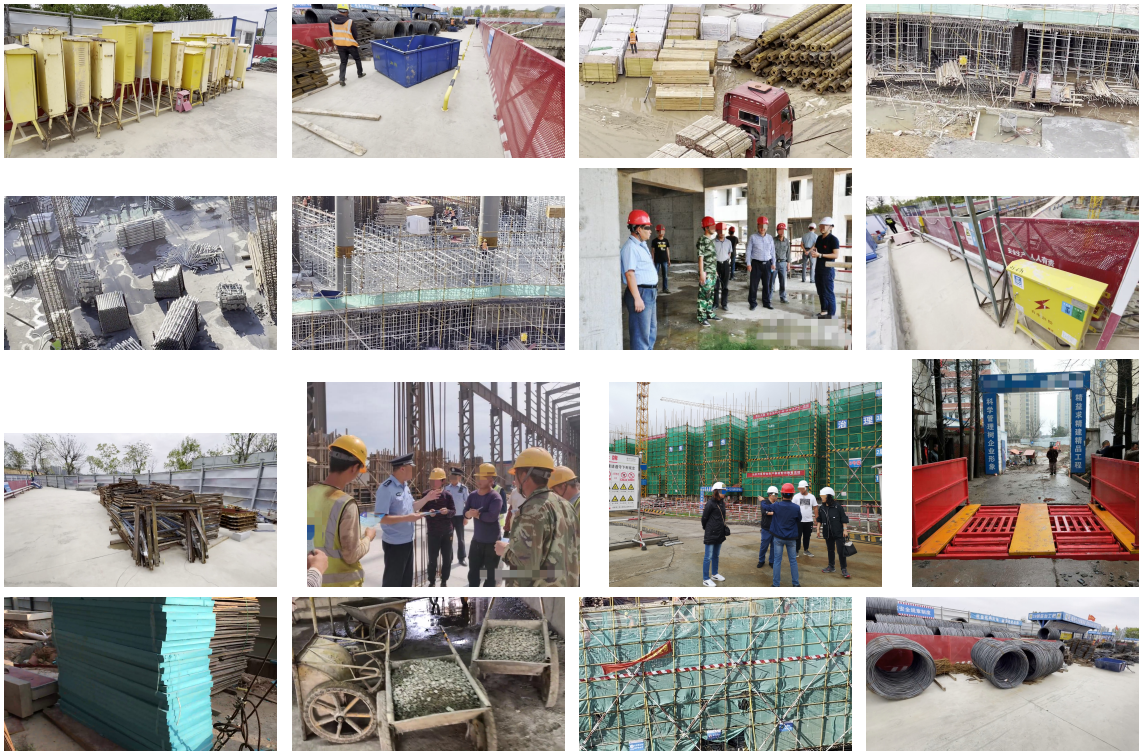


Figure 5.1: Example images included in the SODA dataset.

Model trained with the Vipan dataset

A reason for the network not predicting well can lie in the smaller size of this dataset. The data collected is not general and accurate according to SODAs categories and it is collected in a short period of time in a small area. It would therefore be of more interest to test this network in another construction site project in Sweden to check its generality.

Although the model trained with the Vipan dataset is far from perfect, some observations have been made for potential improvements. For example, looking at the images in Figure 4.10b (especially the upper image), the model is detecting the *ebox* class two times at one object. This is depending on the IoU threshold, explained in Section 2.3.2. Lowering the IoU threshold can thereby solve the multiple box issue. Notice that lowering the IoU will result in the network suppressing boxes of multiple objects of the same class near each other.

5.3 Construction datasets

Deciding the number of images in a dataset (the size) depends on the scenario (for example what kind of environment the object detection will occur and also the number of categories to annotate). Therefore, when developing a custom dataset, studying other successful datasets is to be considered. As mentioned in Section 2.4.2, the SODA dataset includes around 20000 images and 1000 annotations for each category (15 labels). The

article about another dataset with the headline "ImageNet: A Large-Scale Hierarchical Image Database" [5] aimed an average of 600 images per category in their dataset (the visually decided objects to detect). Moreover, deciding the size of the dataset thereby depends not only on the number of images in the dataset but also on the number of annotations in combination with how many classes there are decided to detect. In other words, the dataset is sufficient in terms of the quantity and quality of the data.

There is a lot of complexity when building a customized dataset and there are a lot of techniques out there such as data augmentation. Image data augmentation saves person-hours when building a dataset in the sense that it increases the diversity and the quantity of a dataset. MSCOCO, for instance, was built using segmentation techniques [20].

5.4 Drawbacks

The decision to use YOLOv7 on Spot's portable PC was made without knowing that it just included a CPU. This discovery was made too late in this master's thesis and could have influenced the decision of the selected version of YOLO. Thereby Spot CORE AI (mentioned in Section 2.1.1) would be of relevance when implementing real-time object detection. Spot CORE AI includes a GPU and would improve the frames per second. An experiment showing frames per second using YOLO with both a GPU and a CPU is made in the following article [40].

Chapter 6
Conclusion

This chapter aims to answer the five research questions of this thesis, and thereby, fulfill the purpose of evaluating how well Spot can be used when gathering data autonomously as well as how image data from Spot can be used in computer vision functionalities. The questions are answered with support and arguments from the results and analysis that were conducted throughout this research project.

6.1 Answering research questions

With the two software modules developed for this thesis, Spot can perform autonomous data collection as well as object detection. Docker has proven to be required when implementing new software on Spot, and Robot Operating System was a handy way of solving the communication and data storage. While Flask was not directly required, it helped in creating an interface for human robot interaction. Based on the drop in frame rate when running YOLO directly on the Spot CORE it can be concluded that it should only be done on demand, and not as a continuous execution. Hence the first research question has been answered.

The MSCOCO dataset was not used much in this master thesis due to the lack of detection of objects on Vipan. The YOLOv7 model pre-trained on MSCOCO data was detecting some objects at Vipan such as *laptop*, *person*, and *bottle*. This is very few compared to its 80 classes. The only relevant detection for construction work is persons, which is not enough to make it viable. Thereby the research question of whether a model trained on MSCOCO data can detect any relevant objects on a Swedish construction site is answered.

The third research question was about finding any publicly available datasets that would be suitable for training a YOLO network on the Swedish construction site called

Vipan. Different publicly available datasets were investigated and the decision ended up using the SODA dataset to train a YOLOv7 network. A lot of materials were identified at the construction site at Vipan and the relevance of detecting materials at a construction site was discussed during a meeting with persons specialized in the area. Three other discovered datasets presented in Section 3.3.1 were relevant to construction sites, but they did not compete with the SODA dataset, which had the most identified objects at Vipan. At Vipan, some of the classes were present in the CHV dataset, including yellow helmets and vests, but not any other objects. The CHV dataset is therefore incomplete due to the absence of materials. There were two other datasets mentioned in Table 3.1 (ACID and MOCS) that included large construction machines that were not used in an indoor environment at Vipan, so they were not selected as a suitable dataset.

Two models of YOLOv7 were trained on data taken from a construction environment, one with the SODA dataset and the other including images from Vipan. The SODA dataset was best suited for the indoor environment at Vipan and many of the objects were included in the SODA classes. Objects such as *vests*, *helmets*, and *boards* were detected from the test data. As a large number of images in SODA are taken outside, it is not best suited for the indoor environments tested in this thesis. This led to a mAP@.5 of 0.264 and a mAP@.5:.95 of 0.138 for the YOLOv7 model trained on SODA data and tested with images captured at Vipan. These mAP results are too low and thereby a more specialized network is needed for the construction site at Vipan. Keep in mind that the mAP takes the mean of all classes so if one should just investigate the classes it performed well such as *person*, *helmet*, and *vest* the mAP would have been much higher. As mentioned in the results it would be of interest to test SODA with expanded test data from Swedish construction sites. For example, images taken outdoors and with greater variance will lead to a more reliable mAP to calculate the performance of a YOLOv7 model trained on the SODA dataset. The second model with data taken from a construction environment is the customized model (trained on Vipan data). This model performed better than the network trained on the SODA dataset when looking at the mAP scores. When tested on data from Vipan the resulting mAP scores landed in a mAP.5 of 0.293 and in a mAP.5:.95 of 0.101. As mentioned in the results this is not a perfect indicator of the suitability of the network, the similarity of train and test data leads to a very specialized model that probably would not perform as well on another construction site. However, the intent was not to create a perfect dataset and model, but rather investigate the workload and process needed in the future. While looking at the positive side, increasing the quantity and quality of the image data in the customized dataset could improve the mAP scores. To summarize, the mAP given above answers the fourth research question.

When answering what is required for a dataset in the scenario of this work some things are noted (research question 5). Firstly, building a custom dataset that could detect any objects from Vipan took 16 invested hours to complete. A larger variance in the image data and more people involved would result in a decent network. Images collected in various contexts under specific conditions, different materials, and different manufacturers' equipment contribute to a great deal of variance in the data. Professional competence is needed to correctly annotate and analyze the objects in the images taken from a construction site. Optimization in the form of fine-tuning hyperparameters could make our

network even better but this was as mentioned in Section 1.3 not included in this master's thesis. These adjustments could produce a network with higher accuracy and result in a more systematic and autonomous way of performing inspections on construction sites. Another aspect is that YOLOv7 has a complex architecture that shows great performance in the form of high accuracy. But this comes with a trade-off because a complex model can also be too complex for its data which can result in overfitting [1]. Including more data for the model to train on can prevent overfitting and this can be done by either collecting new data or using data augmentation.

Finally, we believe that Spot is an excellent addition to the data collection toolbox at construction sites. With the possibility of performing object detection online to avoid getting caught behind obstacles, it has the advantage of needing less human interaction than other robots. Its arm allows for a wide range of camera angles which is important when creating a varied dataset.

6.2 Future Research

A number of ideas for future research emerged during this master's thesis.

- Image processing technology and machine analysis. This includes using Spot to compare two images from the same position at different times to notice changes.
- Forming datasets for different phases during a building process. Example of this could be to dive into a more thorough detection of the material type by creating a more specialized dataset.
- Comparison against a building plan. Using Spot to detect faults in construction, for example counting the number of drilled holes.
- Evaluating different machine learning techniques. This could include evaluating offline solutions and comparing them to other networks
- A goal for Spot could be to tag the objects that cause the previously mentioned errors, map them by geolocation software (An example of geolocation software can be this master's thesis [23]). This data can later be used in a Building Information Model (BIM) service (for example in a 3D digital representation of a building or other build site plans) [21].

References

[1] Khaled Alomar, Halil Ibrahim Aysel, and Xiaohao Cai. Data Augmentation in Classification and Segmentation: A Survey and New Strategies. *Journal of Imaging*, 46(9):1–26, 2022.

[2] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[3] Boverket. Kartläggning av fel, brister och skador inom byggsektorn, 2018. https://www.boverket.se/globalassets/publikationer/dokument/2018/kartlaggning-av-fel-brister-och-skador-inom-byggsektorn.pdf. [Accessed: 28-Oct-2022].

[4] Ivan Culjak, David Abram, Tomislav Pribanic, Hrvoje Dzapo, and Mario Cifrek. A brief introduction to OpenCV. In *2012 proceedings of the 35th international convention MIPRO*, pages 1725–1730. IEEE, 2012.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. *Dept. of Computer Science, Princeton University, USA*, pages 1–8, 2009.

[6] Boston Dynamic Docs. Arm and Gripper Specification. https://dev.bostondynamics.com/docs/concepts/arm/arm_specification. [Accessed: 18-Oct-2022].

[7] Boston Dynamic Docs. SDK Concepts. https://dev.bostondynamics.com/docs/concepts/readme. [Accessed: 10-Nov-2022].

[8] Docker docs. Volume. *https://docs.docker.com/storage/volumes/ .[Accessed: 07-Nov-2022]*, 2022.

[9] Rui Duan, Hui Deng, Mao Tian, Yichuan Deng, and Jiarui Lin. SODA: A large-scale open site object detection dataset for deep learning in construction. *Automation in Construction*, 142:1–18, 2022.

[10] Boston Dynamics. SPOT CHOREOGRAPHY SDK. https://dev.bostondynamics.com/docs/concepts/choreography/readme?highlight=choreography. [Accessed: 11-Oct-2022].

[11] Lukas Ewecker. yolov7-ros, 2022. https://github.com/lukazso/yolov7-ros. [Accessed: 20-Oct-2022].

[12] Pablo Fernández Fernández. Construction Supervision with Augmented Reality. 2022. https://lup.lub.lu.se/student-papers/search/publication/9075076. [Accessed: 15-Apr-2023].

[13] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[14] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras Tensor-Flow*. " O'Reilly Media, Inc.", 2019.

[15] Raphael Holzer. Introduction - Capture a Video. https://opencv-tutorial.readthedocs.io/en/latest/intro/intro.html#capture-live-video. [Accessed: 10-Nov-2022].

[16] Sakshi Indolia, Anil Kumar Goswamib, S. P. Mishrab, and Pooja Asopa. Conceptual Understanding of Convolutional Neural Network - A Deep Learning Approach. *Procedia Computer Science*, 132:679–678, 2018.

[17] LabelImg. labelImg, 2018. https://github.com/heartexlabs/labelImg. [Accessed: 14-Dec-2022].

[18] Lambda. Lambda Stack is all the AI software you need, and it's always up to date. https://lambdalabs.com/lambda-stack-deep-learning-software. [Accessed: 13-Dec-2022].

[19] Jeonghun Lee and Kwang-il Hwang. YOLO with adaptive frame control for real-time object detection applications. *Multimedia Tools and Applications*, 81(5):36375–36396, 2021.

[20] Tsung Yi Lin, James Hays, Michael Maire, Pietro Perona, Serge Belongie, Deva Ramanan, Lubomir Bourdev, C. Lawrence Zitnick, Ross Girshick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *CoRR*, pages 1–15, 2014. https://arxiv.org/abs/1405.0312. [Accessed: 10-Oct-2022].

[21] Gayatri Mahajan. Applications of Drone Technology in Construction Industry: A Study 2012-2021. *International Journal of Engineering and Advanced Technology*, 11(1):1–17, 2021.

[22] Patrick Mihelich and James Bowman. Cvbridge-ros wiki. *http://wiki. ros. org/cv_bridge.[Accessed: 20-Oct-2022]*.

[23] Ola Nilsson. Building dense reconstructions with SLAM and spot. 2022. https://lup.lub.lu.se/student-papers/search/publication/9079252. [Accessed: 10-Dec-2022].

[24] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[25] Yaman Hooda Preeti Kuhar, Kaushal Sharma and Neeraj Kumar Verma. Internet of Things (IoT) based Smart Helmet for Construction. *Journal of Physics: Conference Series*, 1950(1):1–9, 2021.

[26] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[27] Joseph Redmond, Santosh Divvala, Ross Girchick, and Ali Farhadi. You Only Look Once: Unified Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[28] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized Intersection over Union. June 2019.

[29] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach Third E*. Pearson Education Limited.

[30] Iqbal H. Sarker. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science*, 142:104499, 2021.

[31] Ross Girshick Shaoqing Ren, Kaiming He and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Regional Proposal Network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[32] Petru Soviany and Radu Tudor Ionescu. Optimizing the Trade-off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. In *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 209–214, 2018.

[33] Boston Dynamic Support. Payload device network configuration. `https://support.bostondynamics.com/s/article/Payload-device-network-configuration`. [Accessed: 12-Dec-2022].

[34] Boston Dynamic Support. Python Library. `https://dev.bostondynamics.com/docs/python/readme`. [Accessed: 10-Nov-2022].

[35] Boston Dynamic Support. Spot battery and charging system. `https://support.bostondynamics.com/s/article/Introduction-to-the-Spot-battery-and-charger`. [Accessed: 14-Nov-2022].

[36] Boston Dynamic Support. Spot CORE payload reference. `https://support.bostondynamics.com/s/article/Spot-CORE-payload-reference#AboutSpotCORE`. [Accessed: 14-Nov-2022].

[37] Boston Dynamic Support. Understanding Spot Programming. https://dev.bostondynamics.com/docs/python/understanding_spot_programming#services-and-authentication. [Accessed: 10-Nov-2022].

[38] Juan R Terven and Diana M. Cordova-Esparaza. A Comprehensive Review of Yolo: From Yolov1 and Beyond. page 2, 2023. https://arxiv.org/abs/2304.00501. [Accessed: 7-Jun-2023].

[39] Dirk Thomas, Tim Field, Jeremy Leibs, and James Bowman. Rosbag-ros wiki. *http://wiki.ros.org/rosbag. [Accessed: 20-Oct-2022]*, 2014.

[40] Md Bahar Ullah. CPU Based YOLO: A Real Time Object Detection Algorithm. In *2020 IEEE Region 10 Symposium (TENSYMP)*, pages 552–555, 2020.

[41] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark-Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. https://arxiv.org/abs/2207.02696.[Accessed: 10-Apr-2023].

[42] Zijian Wang, Yimin Wu, Lichao Yang, Arjun Thirunavukarasu, Colin Evison, and Yifan Zhao. Fast Personal Protective Equipment Detection for Real Construction Sites Using Deep Learning Approaches. *Sensors*, 21(10), 2021. https://www.mdpi.com/1424-8220/21/10/3478.

[43] Ruffin White and Henrik Christensen. ROS and Docker. In *Robot Operating System (ROS)*, pages 285–307. Springer, 2017.

[44] Bo Xiao and Shih-Chung Kang. Development of an Image Data Set of Construction Machines for Deep Learning Object Detection. *Journal of Computing in Civil Engineering*, 35(2):05020005, 2021. https://www.researchgate.net/publication/349695505_Development_of_an_Image_Data_Set_of_Construction_Machines_for_Deep_Learning_Object_Detection. [Accessed: 20-Mar-2023].

[45] An Xuehui, Zhou Li, Liu Zuguang, Wang Chengzhi, Li Pengfei, and Li Zhiwei. Dataset and benchmark for detecting moving objects in construction sites. *Automation in Construction*, 122(8):103482, 2021.

Appendices

Appendix A
Explanation on flags

A.1 Flags in *train.py* and *test.py*

- **- -workers** how many working processes will put data into the RAM simultaneously.
- **- -device** here one sets the GPU number (ID) for training.
- **- -batch-size** the batch size is the number of samples the network will pass at one time.
- **- -data** path to the custom config file.
- **- -img** image size which the model will train on. By default is it set to 640
- **- -cfg** path to the configuration file which one provides on its own specifying the labels and the splitting of the training, test and validation data.
- **- -weights** - path to pre-trained weights.
- **- -name** provide a name to the subdirectory folder inside the *runs* directory folder in which all the training, validation, and test result are saved.
- **- -hyp** the path to the chosen YOLOv7 model which is defined in a YAML file with included parameters and hyperparameters.
- **- -epochs** is the number of epochs the network will train for.
- **- -conf** the object confidence threshold.
- **- -iou** the IOU threshold for Non-Max-Supression.
- **- -task** the task which will be performed.

A.2 Flag options to Docker

These flag options were used to build the image.

- **- -network=host** Set the networking mode for the RUN instruction during build
- **- -rm** Saves space by removing intermediate containers after a successful build
- **-t <image_name:tag>** Add a tag to the image for clarity

The following flags was used when running *Docker run*

- **-p <hosts_public_ip>:<port_on_host>:<port_exposed_in_file>** Used to forward an open port for connection with the flask app
- **-v <path_to_folder_in/host>:<path/to/folder/in/container>** Mounts a volume that makes a folder on the host accessible from within the container, used to save rosbags locally.
- **-it** Creates an interactive bash shell in the container, used so far for testing purposes
- **- -name <name_of_container>** Assigns a name to the container
- **<image_name:tag>** Name and tag of the image from which to start the container

Appendix B
Python code

B.1 detect_ros.py

```python
#!/usr/bin/python3
import torch
from models.experimental import attempt_load
from utils.general import non_max_suppression
from utils.ros import create_detection_msg
from visualizer import draw_detections

import os
from typing import Tuple, Union, List


import cv2
from torchvision.transforms import ToTensor
import numpy as np
import rospy

from vision_msgs.msg import Detection2DArray, Detection2D,
    BoundingBox2D
from sensor_msgs.msg import Image
from cv_bridge import CvBridge


def parse_classes_file(path):
    classes = []
    with open(path, "r") as f:
        for line in f:
            line = line.replace("\n", "")
            classes.append(line)
    return classes

```

```
30
31 def rescale(ori_shape: Tuple[int, int], boxes: Union[torch.Tensor, np.
   ndarray],
32             target_shape: Tuple[int, int]):
33     """Rescale the output to the original image shape
34     :param ori_shape: original width and height [width, height].
35     :param boxes: original bounding boxes as a torch.Tensor or np.array
       or shape
36         [num_boxes, >=4], where the first 4 entries of each element
   have to be
37         [x1, y1, x2, y2].
38     :param target_shape: target width and height [width, height].
39     """
40     xscale = target_shape[1] / ori_shape[1]
41     yscale = target_shape[0] / ori_shape[0]
42
43     boxes[:, [0, 2]] *= xscale
44     boxes[:, [1, 3]] *= yscale
45
46     return boxes
47
48
49 class YoloV7:
50     def __init__(self, weights, conf_thresh: float = 0.5, iou_thresh:
   float = 0.45,
51                  device: str = "cuda"):
52         self.conf_thresh = conf_thresh
53         self.iou_thresh = iou_thresh
54         self.device = device
55         self.model = attempt_load(weights, map_location=device)
56
57     @torch.no_grad()
58     def inference(self, img: torch.Tensor):
59         """
60         :param img: tensor [c, h, w]
61         :returns: tensor of shape [num_boxes, 6], where each item is
   represented as
62             [x1, y1, x2, y2, confidence, class_id]
63         """
64         img = img.unsqueeze(0)
65         pred_results = self.model(img)[0]
66         detections = non_max_suppression(
67             pred_results, conf_thres=self.conf_thresh, iou_thres=self.
   iou_thresh
68         )
69         if detections:
70             detections = detections[0]
71         return detections
72
73
74 class Yolov7Publisher:
75     def __init__(self, img_topic: str, weights: str, conf_thresh: float
       = 0.5,
76                  iou_thresh: float = 0.45, pub_topic: str = "
   yolov7_detections",
77                  device: str = "cuda",
```

```
78                  img_size: Union[Tuple[int, int], None] = (640, 640),
79                  queue_size: int = 1, visualize: bool = True,
80                  class_labels: Union[List, None] = None):
81         """
82         :param img_topic: name of the image topic to listen to
83         :param weights: path/to/yolo_weights.pt
84         :param conf_thresh: confidence threshold
85         :param iou_thresh: intersection over union threshold
86         :param pub_topic: name of the output topic (will be published
    under the
87             namespace '/yolov7')
88         :param device: device to do inference on (e.g., 'cuda' or 'cpu
    ')
89         :param queue_size: queue size for publishers
90         :visualize: flag to enable publishing the detections visualized
     in the image
91         :param img_size: (height, width) to which the img is resized
    before being
92             fed into the yolo network. Final output coordinates will be
     rescaled to
93             the original img size.
94         :param class_labels: List of length num_classes, containing the
     class
95             labels. The i-th element in this list corresponds to the i-
    th
96             class id. Only for viszalization. If it is None, then no
    class
97             labels are visualized.
98         """
99         self.img_size = img_size
100        self.device = device
101        self.class_labels = class_labels
102
103        vis_topic = pub_topic + "visualization" if pub_topic.endswith("
    /") else \
104            pub_topic + "/visualization"
105        self.visualization_publisher = rospy.Publisher(
106            vis_topic, Image, queue_size=queue_size
107        ) if visualize else None
108
109        self.bridge = CvBridge()
110
111        self.tensorize = ToTensor()
112        self.model = YoloV7(
113            weights=weights, conf_thresh=conf_thresh, iou_thresh=
    iou_thresh,
114            device=device
115        )
116        self.img_subscriber = rospy.Subscriber(
117            '/spot_hand_rgb', Image, self.process_img_msg
118        )
119        self.detection_publisher = rospy.Publisher(
120            pub_topic, Detection2DArray, queue_size=queue_size
121        )
122        rospy.loginfo("YOLOv7 initialization complete. Ready to start
    inference")
```

```
123
124     def process_img_msg(self, img_msg: Image):
125         """ callback function for subscriber """
126         np_img_orig = self.bridge.imgmsg_to_cv2(
127             img_msg, desired_encoding='passthrough'
128         )
129 
130         # handle possible different img formats
131         if len(np_img_orig.shape) == 2:
132             np_img_orig = np.stack([np_img_orig] * 3, axis=2)
133 
134         h_orig, w_orig, c = np_img_orig.shape
135         if c == 1:
136             np_img_orig = np.concatenate([np_img_orig] * 3, axis=2)
137             c = 3
138 
139         # automatically resize the image to the next smaller possible
    size
140         w_scaled, h_scaled = self.img_size
141 
142         # w_scaled = w_orig - (w_orig % 8)
143         np_img_resized = cv2.resize(np_img_orig, (w_scaled, h_scaled))
144         # conversion to torch tensor (copied from original yolov7 repo)
145         img = np_img_resized.transpose((2, 0, 1))[::-1]  # HWC to CHW,
    BGR to RGB
146         img = torch.from_numpy(np.ascontiguousarray(img))
147         img = img.float()  # uint8 to fp16/32
148         img /= 255  # 0 - 255 to 0.0 - 1.
149         img = img.to(self.device)
150 
151         # inference & rescaling the output to original img size
152         detections = self.model.inference(img)
153         #pred_results = self.model(img)[0]
154         detections[:, :4] = rescale(
155             [h_scaled, w_scaled], detections[:, :4], [h_orig, w_orig])
156         detections[:, :4] = detections[:, :4].round()
157 
158 
159         # publishing
160         detection_msg = create_detection_msg(img_msg, detections)
161         self.detection_publisher.publish(detection_msg)
162 
163         # visualizing if required
164         if self.visualization_publisher:
165             bboxes = [[int(x1), int(y1), int(x2), int(y2)]
166                       for x1, y1, x2, y2 in detections[:, :4].tolist()]
167             classes = [int(c) for c in detections[:, 5].tolist()]
168             class_prob = [float(p) for p in detections[:,4].tolist()]
169             vis_img = draw_detections(np_img_orig, bboxes, classes,
    class_prob,
170                                       self.class_labels)
171 
172 
173             #vis_msg = self.bridge.cv2_to_imgmsg(vis_img)
174             #self.visualization_publisher.publish(vis_msg)
175         cv2.namedWindow("Detected", cv2.WND_PROP_FULLSCREEN)
```

```
176         cv2.setWindowProperty("Detected",cv2.WND_PROP_FULLSCREEN,cv2.
    WINDOW_FULLSCREEN)
177         cv2.imshow("Detected", vis_img)
178         cv2.waitKey(1)
179
180 if __name__ == "__main__":
181     rospy.init_node("yolov7_node")
182
183     ns = rospy.get_name() + "/"
184
185     weights_path = rospy.get_param(ns + "weights_path")
186     classes_path = rospy.get_param(ns + "classes_path")
187     img_topic = rospy.get_param(ns + "img_topic")
188     out_topic = rospy.get_param(ns + "out_topic")
189     conf_thresh = rospy.get_param(ns + "conf_thresh")
190     iou_thresh = rospy.get_param(ns + "iou_thresh")
191     queue_size = rospy.get_param(ns + "queue_size")
192     img_size = rospy.get_param(ns + "img_size")
193     visualize = rospy.get_param(ns + "visualize")
194     device = rospy.get_param(ns + "device")
195
196
197     # some sanity checks
198     if not os.path.isfile(weights_path):
199         raise FileExistsError(f"Weights not found ({weights_path}).")
200
201     if classes_path:
202         if not os.path.isfile(classes_path):
203             raise FileExistsError(f"Classes file not found ({
    classes_path}).")
204         classes = parse_classes_file(classes_path)
205     else:
206         rospy.loginfo("No class file provided. Class labels will not be
     visualized.")
207         classes = None
208
209     if not ("cuda" in device or "cpu" in device):
210         raise ValueError("Check your device.")
211
212
213     publisher = Yolov7Publisher(
214         img_topic=img_topic,
215         pub_topic=out_topic,
216         weights=weights_path,
217         device=device,
218         visualize=visualize,
219         conf_thresh=conf_thresh,
220         iou_thresh=iou_thresh,
221         img_size=(img_size, img_size),
222         queue_size=queue_size,
223         class_labels=classes
224     )
225
226     rospy.spin()
```

B.2 get_image_stream.py

```
#!/usr/bin/python3

# Copyright (c) 2022 Boston Dynamics, Inc.  All rights reserved.
#
# Downloading, reproducing, distributing or otherwise using the SDK Software
# is subject to the terms and conditions of the Boston Dynamics Software
# Development Kit License (20191101-BDSDK-SL).


import argparse
import sys

import cv2
import numpy as np
from scipy import ndimage
import rospy
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image as Image

import bosdyn.client
import bosdyn.client.util
from bosdyn.api import image_pb2
from bosdyn.client.image import ImageClient, build_image_request


class CameraNode:

    arg_defaults = {
        'username': 'null',
        'password': 'null',
        'hostname': 'null',
        'camera': 'null' ,
        'pixel-format': 'null'}

    def __init__(self) -> None:
        pass


    def setArgs(self,args):
        '''
        Sets arguments for authentication with spot-sdk
        @param args: dictionary of string values 'username' 'password' 'hostname'
        '''
        self.arg_defaults.update(args)

    def getArgs(self):
        '''
        @return: Dictionary of arguments
        '''
        return self.arg_defaults
```

```
51
52      def extractData(self, image_responses):
53          cv_visual = cv2.imdecode(np.frombuffer(image_responses[0].show.
    image.data, dtype=np.uint8), -1)
54          return cv_visual
55
56      def pixel_format_type_strings(self):
57          names = image_pb2.Image.PixelFormat.keys()
58          return names[1:]
59
60      def pixel_format_string_to_enum(self, enum_string):
61          return dict(image_pb2.Image.PixelFormat.items()).get(
    enum_string)
62
63
64      def authenticate(self):
65
66          argv = ['--camera', self.arg_defaults['camera'],
67                  '--pixel-format', self.arg_defaults['pixel-format'],
68                  '--username',self.arg_defaults['username'],
69                  '--password',self.arg_defaults['password'],
70                  self.arg_defaults['hostname']]
71
72          parser = argparse.ArgumentParser()
73          bosdyn.client.util.add_common_arguments(parser)
74          parser.add_argument('--camera', help='Camera to cature images
    from',
75                              default='hand_color_image')
76          parser.add_argument(
77          '--pixel-format', choices=self.pixel_format_type_strings(),
78          help='Requested pixel format of image. If supplied, will be
    used for all sources.')
79
80          options = parser.parse_args(argv)
81
82          sdk = bosdyn.client.create_standard_sdk('hand_image')
83          robot = sdk.create_robot(options.hostname)
84          robot.authenticate(options.username, options.password)
85          robot.sync_with_directory()
86          robot.time_sync.wait_for_sync()
87          image_client = robot.ensure_client(ImageClient.
    default_service_name)
88
89          return image_client, options
90
91      def initializeNode(self):
92
93          rospy.init_node("hand_camera", anonymous = True)
94          hand_pub = rospy.Publisher("/spot_hand_rgb", Image, queue_size
    =1)
95          return hand_pub
96
97
98      def run(self):
99          # Get parameters from launch file
100
```

```
        arg_defaults = {
            'hostname': rospy.get_param('hand_image/hostname'),
            'camera': rospy.get_param('hand_image/camera'),
            'pixel-format': rospy.get_param('hand_image/pixel-format'),
            'username': rospy.get_param('hand_image/username'),
            'password': rospy.get_param('hand_image/password')}


        cam.setArgs(arg_defaults)
        image_client, options = cam.authenticate()
        sources = [options.camera]

        hand_pub = cam.initializeNode()
        rate = rospy.Rate(30) # Set recording framerate

        bridge = CvBridge()

        while not rospy.is_shutdown():
            image_responses = image_client.get_image_from_sources(
    sources)

            for image in image_responses:
                num_bytes = 1  # Assume a default of 1 byte encodings.
                if image.shot.image.pixel_format == image_pb2.Image.
    PIXEL_FORMAT_DEPTH_U16:
                    dtype = np.uint16
                    extension = ".png"
                else:
                    if image.shot.image.pixel_format == image_pb2.Image
    .PIXEL_FORMAT_RGB_U8:
                        num_bytes = 3
                    elif image.shot.image.pixel_format == image_pb2.
    Image.PIXEL_FORMAT_RGBA_U8:
                        num_bytes = 4
                    elif image.shot.image.pixel_format == image_pb2.
    Image.PIXEL_FORMAT_GREYSCALE_U8:
                        num_bytes = 1
                    elif image.shot.image.pixel_format == image_pb2.
    Image.PIXEL_FORMAT_GREYSCALE_U16:
                        num_bytes = 2
                    dtype = np.uint8
                    extension = ".jpg"

                img = np.frombuffer(image.shot.image.data, dtype=dtype)
                if image.shot.image.format == image_pb2.Image.
    FORMAT_RAW:
                    try:
                        # Attempt to reshape array into a RGB rows X
    cols shape.
                        img = img.reshape((image.shot.image.rows, image
    .shot.image.cols, num_bytes))
                    except ValueError:
                        # Unable to reshape the image data, trying a
    regular decode.
                        img = cv2.imdecode(img, -1)
                else:
```

```
147                     img = cv2.imdecode(img, -1)
148
149                 imgmsg = bridge.cv2_to_imgmsg(img, encoding="
    passthrough")
150                 hand_pub.publish(imgmsg)
151                 rate.sleep()
152
153
154 if __name__ == "__main__":
155     cam = CameraNode()
156     cam.run()
```

Appendix C

Training results of networks

C.1 SODA

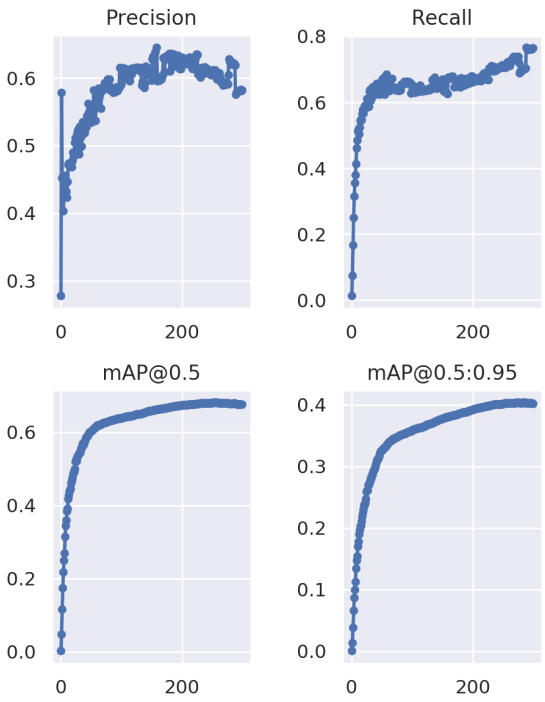


Figure C.1: Performance measurements as precision, recall, and mAP:s under training. The x-axis shows the progress after every epoch, it is 300 epochs in total.

C.2 Vipan dataset

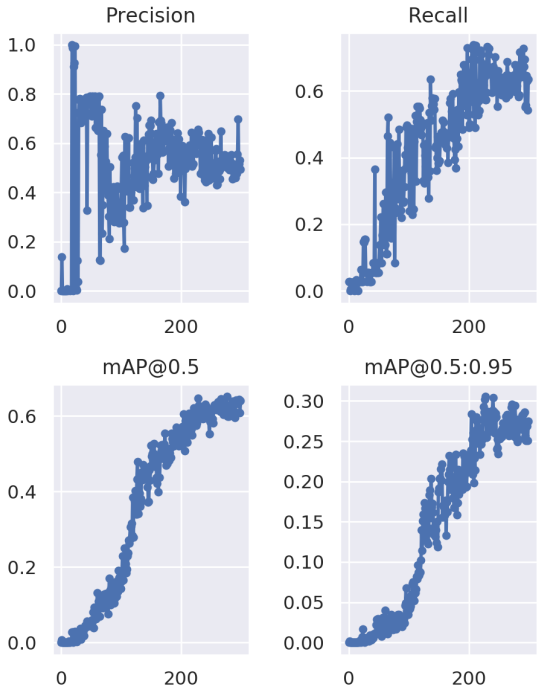


Figure C.2: Performance measurements as precision, recall, and mAP:s under training. The x-axis shows the progress after every epoch, it is 300 epochs in total.

Appendix D
Software deployment

D.1 Commands for deploying software

```
$ docker build --network=host --rm -t <image_name:tag> .
```

The docker image is saved to a tar archive with the command below. Before moving the tar file it needs to be executable.

```
$ docker save <image_name:tag> > <image_name>.tar
```

The image is loaded on Spot CORE with the command below.

```
$ docker load --input <image_name>.tar
```

To run the Docker container for the first time the following command is used:

```
$ docker run -p 192.168.50.5:x:y
-v /home/spot/Documents/johanamanda/Rosbags:/saved_bags
-it --name <container_name> <image_name:tag>
```

The option -p connects port x on the Spot CORE with port y in the Docker container, full configuration of ports is shown in Table D.1. A Docker volume is defined by -v, this makes it possible to save rosbags directly to Spot COREs file system. See Appendix A.2 for explanation of all flags. When the container has been started once, it can easily be restarted through the Cockpit GUI.

Table D.1: Network port forwarding for the two Docker containers

Package	Spot IP	CORE IP	Container port
Data Collection	192.168.80.3:20443	192.168.50.5:443	5000
Online-YOLO	192.168.80.3:20080	192.168.50.5:80	5001

Appendix E
Commands for YOLO

E.1 Training and testing YOLO

Training SODA dataset

```
$ python train.py --epochs 300 --workers 8 --device 0
  --batch-size 32 --data data/soda.yaml --img 640 640
  --cfg cfg/training/yolov7-tiny.yaml --weights ‘’
  --name yolov7_tiny --hyp data/hyp.scratch.tiny.yaml
```

Training Custom dataset

```
$ python train.py --epochs 300 --workers 8 --device 0
  --batch-size 32 --data data/vipan.yaml --img 640 640
  --cfg cfg/training/yolov7-tiny.yaml --weights ‘’
  --name yolov7_vipan --hyp data/hyp.scratch.tiny.yaml
```

Testing SODA dataset

```
$ python test.py --data data/<name_of_dataset>.yaml --img 640
--batch <batch_size> --conf 0.001 --iou 0.65
--device 0 --task=test --weights <name_of_weights>.pt
--name soda_test_out_data
```

Appendix F
Lessons learned

F.1 Experiences and thoughs

The outcomes of this master's thesis have provided a valuable understanding of the subject matter. These discoveries may be useful for the development of future research.

- When collecting the necessary software packages for the deep learning GPU environment it can be a struggle to install the right versions due to strict dependencies. Example of these kind of packages are Keras, Tensorflow and Pytorch. This problem can easily be solved with lambda stack which always keeps the necessary AI Software updated [18]. The installation is easily done in the terminal with one command line.

- When collecting data that is intended to be used in training object detection networks some important points have been noted. Take clear photographs, stop and let the camera auto focus if needed, from different angles and change the lighting if possible. If multiple cameras will be used with the completed network, make sure to include images from the whole range of perspectives. Using multiple types of cameras will also make the network more robust in the sense of camera parameters.

  Thanks to its arm, Spot is capable of taking photos from many more perspectives than robots with cameras at a fixed height.

- Having multiple people check the data will be needed when producing a larger dataset, as the method of annotating images can impact the quality a lot.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-01-16

EXAMENSARBETE Evaluation of Spot as an instrument to provide autonomous data collection
Performing object detection using YOLO and ROS at construction sites
STUDENTER Amanda Zarkout, Johan Lindberg
HANDLEDARE Mathias Haage (LTH), Volker Kruger (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Utvärdering av Spot som ett instrument för autonom datainsamling

POPULÄRVETENSKAPLIG SAMMANFATTNING **Amanda Zarkout, Johan Lindberg**

I detta arbete genomförs en förstudie med hjälp av roboten Spot från Boston Dynamics. Uppgiften är autonom insamling av bildmaterial från en byggarbetsplats och att automatiskt tolka insamlat material med hjälp av maskininlärning. Genom att nyttja robotik för insamling kan kontroller genomföras ofta och kan leda till lägre kostnader för korrigering av fel genom snabbare upptäckt.

Byggindustrin är en av de minst digitaliserade i dagens moderna samhälle där misstag bidrar till höga kostnader. År 2018 uppskattade Svenska myndigheten Boverket att dessa ligger mellan 80 och 110 miljarder kronor per år. I en bransch med så små marginaler finns det gott om möjligheter för ny teknik att mitigera fel och därmed bidra till minskade kostnader.

Ett stort område, och det som fokuserats på i denna rapport, är det kontinuerliga inspektionsarbetet. Detta är viktigt för att under arbetets gång kunna upptäcka fel så tidigt som möjligt medan de är lätta och billiga att rätta till. Olika tekniska hjälpmedel för att automatisera detta finns i form av till exempel drönare och hjälmar med monterade kameror på. Syftet med denna rapport är att utvärdera hur maskininlärning kan användas för att analysera data insamlat av den fyrbenta robothunden Spot. Tack vare sina ben samt den befintliga navigeringsmjukvaran kan Spot ta sig fram på byggarbetsplatser där små hinder ofta förekommer. Spots datainsamling är dock ett problem i dagsläget och ny mjukvara har därför utvecklats i detta arbete.

Alla applikationer har skrivits med hjälp av robotoperativsystemet ROS, detta hjälper till med mycket av kommunikationen samt lagring av data. Testdata i form av videor har, under hösten 2022, samlats in på en byggarbetsplats i Lund. För att tolka den data som samlats har artificiell intelligens (AI) använts, i form av det neurala nätverket You Only Look Once (YOLO). YOLO kan användas för att behandla datan i efterhand genom automatisk tolkning som annars genomförs av människor. Men det går även att göra identifieringen direkt på Spots inbyggda dator, detta låter Spot hantera ännu svårare situationer genom att kunna skilja på olika typer av objekt som står i dess väg.

Likt mycket annan AI så krävs det en stor mängd data för att träna upp det neurala nätverket. Datan måste innehålla de objekten man är intresserad av att identifiera. Det visade sig finnas relativt få relevanta publika dataset, men ett kinesiskt dataset vid namn SODA hittades och undersöktes. Det fungerade inte tillräckligt bra, då bilderna skiljde sig för mycket från den svenska miljön. Därför undersöktes även arbetsbördan samt prestandan på ett neuralt nätverk vilket tränades på den insamlade datan från Lund.

Figur 1: Identifierade objekt i en bild tagen av Spot

Dessa två neurala nätverk testades på samma data och resultatet anger hur bra de är på att identifiera objekt, främst byggmaterial och människor. Bägge testerna resulterade i strax under 30% träffsäkerhet, det neurala nätverket som tränades på SODA gav bättre på människor medans det egna nätverket presterade bättre på material. Detta tyder på att en större mängd träningsdata är viktig. Ett större dataset bör byggas upp innan metoden provad i detta arbeta kan nyttjas. Spot har dock visat sig vara ett lovande tillskott till verktygslådan för autonom datainsamling.