Kevin Karlsholm Kaldvansvik

Spot, a mobile four-legged visual asset tracking robot

Master's thesis in Cybernetics and Robotics
Supervisor: Professor Annette Stahl
Co-supervisor: Johan Hatleskog, Trygve Utstumo
December 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Kevin Karlsholm Kaldvansvik

Spot, a mobile four-legged visual asset tracking robot

Master's thesis in Cybernetics and Robotics
Supervisor: Professor Annette Stahl
Co-supervisor: Johan Hatleskog, Trygve Utstumo
December 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This master thesis concludes five years of studies in the Master of Science in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The work in this thesis was accomplished during the fall semester of 2021 and utilized a wide variety of methods that were digested through over five years of dedicated study. Thank you, NTNU, for the vibrant environment that has nurtured immense personal growth.

I want to thank my supervisor, professor Annette Stahl for providing excellent guidance and feedback along the way. I also want to thank my co-supervisors Johan Hatleskog and Trygve Utstumo at Cognite AS, which provided the asset tracking case for this thesis and let me utilize the Boston Dynamics Spot robot.

A very special thanks to my phenomenal significant other, Tine Mathilde Benjaminsen, who has provided invaluable love and support throughout these demanding years of studies.

Abstract

The motivation behind this work is to propose a novel visual asset tracking pipeline with the objective of localizing, identifying, and counting the number of containers in an outdoor industrial environment. This is achieved by using the four-legged robot Spot, capturing 360-images during inspections which are processed in the pipeline utilizing methods from the field of computer vision. Traditional asset tracking solutions rely on using RFID, GPS, and BLE technology, which are recognized as expensive to scale and unfeasible for certain assets. This work aims at extending the research on asset tracking by taking a visual approach to the problem. A custom data set consisting of both images and videos of containers at industrial sites in different weather conditions was created to develop and evaluate the pipeline. The proposed method first undistorts and detects containers in images utilizing a trained EfficientDet detector. Tracks are then generated using the prominent tracking-by-detection method SORT, which results in containers being mostly- or partially tracked. The tracks are triangulated to yield container point clouds that are fit to spherical container models using RANSAC, where a greedy merging scheme is proposed to merge overlapping models. The resulting pipeline localized 80% of the containers in sunny and overcast weather, where 75% are localized correctly to ground truths. The identification is achieved by the detection of fiducial markers, where 50% of the containers were identified. A consequence of triangulating each container track is the possibility of duplicate models, where one such model was triangulated for both weather conditions. Future work should improve the underlying data set and the proposed asset tracking metrics, in addition to components of the proposed pipeline. Hopefully, these initial results will pave the way for improved- and new visual asset tracking solutions.

Keywords: Visual asset tracking, object detection, multiple object tracking, 3D reconstruction

Sammendrag

Motivasjonen bak dette arbeidet er å foreslå et nytt visuelt sporingssystem av ressurser med målet om å lokalisere, identifisere og telle antallet konteinere i et utendørs industrielt miljø. Dette oppnås ved å bruke den firbeinte roboten Spot, som tar 360-bilder under inspeksjoner for å prosesseres i systemet ved hjelp av teknikker i datasyn. Tradisjonelle sporingssystemer er avhengig av å bruke RFID, GPS og BLE teknologi som er kjent for å være dyrt å skalere og ikke gjennomførbart for enkelte ressurser. Dette arbeidet har som mål å utvide forskningen på sporingssystem av ressurser ved å ta en visuell tilnærming til problemet. Et skreddersydd datasett bestående av bilder og videoer av konteinere på industrielle områder i forskjellig værforhold var konstruert for å evaluere systemet. Den foreslåtte metoden starter med å fjerne bildeforvrengninger og detekterer konteinere i bildene ved å utnytte en trent EfficientDet detektor. Spor blir deretter skapt ved å bruke sporing-ved-deteksjon metoden SORT, som resulterer i at konteinere blir for det meste- og delvis sporet. Sporene blir triangulert for å skape konteiner punktskyer som er tilpasset sfæriske konteiner-modeller ved RANSAC, hvor en grådig sammenslåing er foreslått for å slå sammen overlappende modeller. Det resulterende systemet lokaliserte 80% av konteinerne i solfullt og overskyet vær, hvor 75% er lokalisert riktig i forhold til fasiten. Identifiseringen er oppnådd ved detekteringen av *fiducial*-markører, hvor 50% av konteinerne ble identifisert. En konsekvens av trianguleringen av hvert spor er muligheten for duplikate modeller, hvor èn slik modell ble triangulert for hver værtilstand. Videre arbeid burde forbedre det underliggende datasettet og de foreslåtte ressurs-sporing metrikkene, i tillegg til komponentene av det foreslåtte systemet. Disse initiale resultatene vil forhåpentligvis rydde banen for forbedrede- og nye visuelle sporingssystem av ressurser.

Keywords: Visuelt sporingssystem av ressurser, objekt-deteksjon, sporing av flere objekter, 3D rekonstruksjon

Contents

Nomenclature **7**

List of Tables **10**

List of Figures **11**

1 Introduction **13**
- 1.1 Background . . . 13
- 1.2 State of research . . . 16
- 1.3 Contributions . . . 21
- 1.4 Report structure . . . 22

2 Theory **23**
- 2.1 Computer vision . . . 23
  - 2.1.1 RGB representation of images . . . 23
- 2.2 Homogeneous transformations . . . 24
  - 2.2.1 Rotation . . . 25
  - 2.2.2 Translation . . . 25
  - 2.2.3 The homogeneous transformation matrix . . . 25
  - 2.2.4 Properties of the transformation matrix . . . 25
  - 2.2.5 Coordinate transformations . . . 26
- 2.3 Image formation . . . 27
  - 2.3.1 The projective camera . . . 27
  - 2.3.2 Camera distortion . . . 29
- 2.4 Feature detection, description, and matching . . . 31
  - 2.4.1 Feature detection . . . 31
  - 2.4.2 Feature description . . . 32
  - 2.4.3 Feature matching . . . 33
- 2.5 Image homographies . . . 34
  - 2.5.1 The Direct Linear Transformation algorithm . . . 34
- 2.6 Structure computation . . . 35
  - 2.6.1 Initial linear reconstruction . . . 36
  - 2.6.2 Optimal reconstruction . . . 37
- 2.7 Random sample consensus . . . 38

2.8 Epipolar geometry 39
2.8.1 The fundamental matrix 40
2.9 Deep learning in computer vision 41
2.9.1 Convolutional neural network 41
2.9.2 Training 42
2.9.3 Hyperparameters 43
2.9.4 The overfitting problem 45
2.9.5 Regularization 46
2.9.6 Transfer learning 47
2.10 Object detection 48
2.10.1 Problem formulation 48
2.10.2 Modern object detector anatomy 49
2.10.3 Evaluation metrics 51
2.10.4 EfficientDet detector 52
2.11 Multiple object tracking 55
2.11.1 Problem formulation 55
2.11.2 The two main schemes 55
2.11.3 Evaluation Metrics 56
2.11.4 Simple Online Realtime Tracking 57
2.12 Fiducial detection 61
2.12.1 The AprilTag detector 61

3 Methods 63
3.1 Asset description 63
3.2 The Spot platform 64
3.2.1 Spot CAM+ 64
3.2.2 Spot CORE 64
3.2.3 Spot missions 66
3.2.4 Spot frames 66
3.3 The data set 68
3.3.1 Data gathering 68
3.3.2 Data set annotation 72
3.3.3 Summary 73
3.4 The pipeline 75
3.4.1 Undistortion of images 76
3.4.2 Detection of containers 77
3.4.3 Tracking of containers 79
3.4.4 Identification of containers 80
3.4.5 Reconstruction of container models 82
3.4.6 Merging of container models 88
3.4.7 Final localization, identification, and counting 91

4 Results and Discussions 93
4.1 Container detection 93
4.2 Container tracking 94
4.3 Container model estimation 97
4.4 Final pipeline metrics 101
4.5 Accuracy of reconstruction 102
4.6 Parameters and thresholds 103
4.7 Sphere approximation and merging 103
4.8 Occlusions 104
4.9 Weather 105

5 Conclusion 106
5.1 Future work 106

A Complete container detector performance tables 109

Bibliography 110

Nomenclature

Abbreviations

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
BA	Bundle Adjustment
BiFPN	Bi-directional Feature Pyramid Network
BLE	Bluetooth Low Energy
BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
DBT	Detection Based Tracking
DCNN	Deep Convolutional Neural Network
DFT	Detection Free Tracking
DLT	Direct Linear Transformation
FPN	Feature Pyramid Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
GTT	Ground Truth Tracks
HOG	Histogram of Oriented Gradients
ID	Identification

IoU	Intersection over Union
KF	Kalman Filter
MOT	Multiple Object Tracking
NNDR	Nearest Neighbor Distance Ratio
ORB	Oriented FAST and Rotated BRIEF
PTZ	Pan-Tilt-Zoom
RCNN	Region with CNN features
RFCN	Region-based Fully Convolutional Networks
RFID	Radio-Frequency Identification
RGB	Red, Green, and Blue
SDK	Software Development Kit
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SOBA	Structure Only Bundle Adjustment
SORT	Simple Online Realtime Tracking
SOTA	State Of The Art
SURF	Speeded Up Robust Features
YOLO	You Only Look Once
Evaluation metrics	
AP	Average Precision
AR	Average Recall
DM	Duplicate Models
FN	False Negative
FP	False Positive
IDA	Identified Assets
IDs	Identity Switches
LA	Localized Assets

LAGT	Localized Assets correct with respect to Ground Truth
mAP	mean Average Precision
ML	Mostly Lost
MOTA	Multiple Object Tracking Accuracy
MOTP	Multiple Object Tracking Precision
MT	Mostly Tracked
PT	Partially Tracked
TP	True Positive

List of Tables

2.1 EfficientDet models 54

4.1 Container detection metrics 93
4.2 Container tracking metrics 94
4.3 Final pipeline metrics 101

A.1 Container detection precision metrics 109
A.2 Container detection recall metrics 109

List of Figures

1.1 The container asset 15
1.2 The asset tracking problem 15

2.1 An RGB image 24
2.2 The homogeneous transformation 24
2.3 The pinhole model geometry 27
2.4 Radial distortion 29
2.5 Tangential distortion 30
2.6 Image patches 31
2.7 SIFT descriptor 33
2.8 Non-intersecting rays 36
2.9 Three view reconstruction 37
2.10 RANSAC 39
2.11 Epipolar geometry 40
2.12 Typical convolutional neural network 41
2.13 The overfitting problem 45
2.14 Early stopping regularization 46
2.15 The object detection problem 48
2.16 Feature pyramid networks 50
2.17 EfficientDet benchmarking 52
2.18 EfficientDet network architecture 53
2.19 The two MOT schemes 55
2.20 MOT online- and offline tracking 56
2.21 The general Kalman filtering scheme 58
2.22 Common fiducial markers 61

3.1 Containers at Kvaerner Stord 64
3.2 The Spot robotic platform 65
3.3 The SpotCAM+ payload 65
3.4 The SpotCORE payload 66
3.5 The Spot body frame 67
3.6 Configuration of ring camera frames 67
3.7 Robot trajectory in sunny conditions 69

3.8 Frames in sunny conditions 70
3.9 Frames in overcast conditions 71
3.10 The novel visual asset tracking pipeline 75
3.11 The view of the five ring fisheye cameras 76
3.12 The view of the five ring cameras after undistortion 76
3.13 Container detector training graphs 78
3.14 Container detections in ring cameras 78
3.15 Container tracks in ring cameras 80
3.16 AprilTag detections in ring cameras 81
3.17 SIFT features inside bounding box 82
3.18 Brute force matching of features 83
3.19 Epipolar constraint filtering 84
3.20 Homography filtering 85
3.21 Initial triangulated container point clouds 86
3.22 Angle between rays 87
3.23 Container model inliers 88
3.24 Initial and desired sphere configuration 89
3.25 Intersecting spheres geometry 90
3.26 Resulting merged model inliers 91

4.1 Lost tracks due to angular acceleration 96
4.2 Angular acceleration in sunny conditions 96
4.3 Angular acceleration in overcast conditions 97
4.4 Final inliers in sunny conditions using ground truth tracks . . 98
4.5 Final inliers in sunny conditions using generated tracks 98
4.6 Final inliers in overcast conditions using ground truth tracks . 99
4.7 Final inliers in overcast conditions using generated tracks . . 100
4.8 Occluding car, sun glare- and flare 104

Chapter 1

Introduction

1.1 Background

The problem of tracking the position of items that a business owns and uses repeatedly is known as asset tracking. For Kvaerner Stord shipping yard, this has become an increasingly important problem due to valuable items being lost or misplaced on a vast amount of area. These assets are typically manually relocated, which can be laborious and stressful for the workers and expensive for Kvaerner. Examples of such assets are shipping containers, forklifts, concrete supports, and cars, where shipping containers will be considered in this thesis due to the continuation of the asset tracking case in collaboration with Cognite AS in the preliminary work. Figure 1.1 shows two such containers, where fig. 1.2 shows an example of how five containers are spread on the shipping yard. A visual summary of this asset tracking case is found in the video here[1].

Traditional asset tracking solutions typically rely on mounting some external device to the asset which can be tracked. Common approaches are based on Radio-Frequency Identification (RFID), WI-FI, Global Positioning System (GPS), or Bluetooth Low Energy (BLE) [52]. In [66] the authors utilize RFID to track medical equipment in a healthcare facility, where they propose an optimal placement of RFID receivers to reduce the number of devices. Asset tracking has also received attention in space, where [10] utilizes RFID to track consumable items onboard the International Space Station. A hybrid approach is found in [43], where WI-FI is utilized to determine a course position of an asset. The exact position is determined by BLE as a mobile device approaches the asset. For outdoor environments, the GPS is one of the most successful approaches [52], where Differential GPS provides better precision. The evident drawback of these approaches based on external devices is that they do not scale well. Additional devices will likely introduce higher costs, increased maintenance, and new logistical problems.

[1]Video: `https://www.youtube.com/watch?v=fkZYSjVNHFc&ab_channel=Cognite`

Due to the drawbacks and limitations of the traditional approaches, utilizing mobile robots for asset tracking has gained some research attention in the last decades. The first identified asset tracking robot was introduced in 2005 [1], which utilized a wheeled BlueBot robot. The robot was equipped with an RFID reader and periodically surveyed a pre-defined space and located assets by utilizing a WI-FI positioning system. A similar approach is found in [62], where assets inside a data center are tracked. In [63] the authors equip a Roomba with cameras, to localize and identify assets with pre-programmed light-emitting diodes.

In the last few years, mobile robotic platforms such as Spot[2], Taurob[3] and ANYmal[4] [36] have become commercially available. These mobile robots provide researchers and developers with a fully functional and robust platform out of the box, enabling the users to focus on applications rather than the platform's development. Additionally, providers have seen the importance of developing payloads to extend the platform for commercial use. The Spot platform currently features an additional Central Processing Unit (CPU) and Graphics Processing Unit (GPU), light detection and ranging sensor, Pan-Tilt-Zoom (PTZ), thermal- and 360 camera, and an arm for interaction. In [37] the authors present an approach of using the ANYmal robot in the Autonomous Robot for Gas and Oil Sites challenge. The robot was able to navigate an outdoor industrial environment using light detection and ranging, where analog gauges were read utilizing a PTZ camera and methods from the field of computer vision. In [8] the authors utilize the Spot robot for autonomous exploration of extreme environments, where the proposed solution won the 1st-place in the 2020 DARPA Subterranean challenge.

Even though there have been proposed several solutions to the asset tracking problem, all of them except [63] utilize either one or a combination of traditional asset tracking approaches and robotic platforms, which are not suitable for outdoor environments. The introduction of robust robotic platforms calls for research on new asset tracking solutions utilizing these recent robotic advancements.

The scope of this thesis is thus to propose a novel visual asset tracking pipeline, with the primary objectives of localizing, counting, and identifying containers in an outdoor industrial environment. This will be achieved by utilizing the Spot robotic platform equipped with a 360 camera payload and methods from the field of computer vision. Due to the time limitation of this work, well-proven and simplistic methods are favored over State Of The Art (SOTA) and intricate methods. However, SOTA and other promising extensions will be considered in the discussions.

[2]Spot: https://www.bostondynamics.com/spot
[3]Taurob: https://www.taurob.com/taurob-inspector
[4]ANYmal: https://www.anybotics.com/anymal-legged-robot

Figure 1.1: Two shipping containers at Kvaerner Stord.

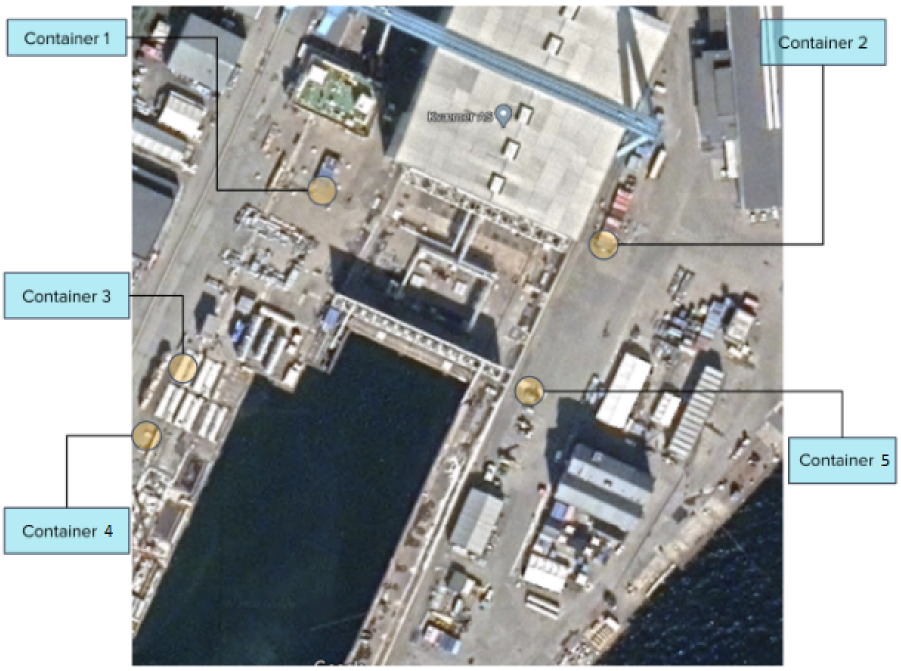


Figure 1.2: An example of five container assets on the Kvaerner Stord shipping yard which can be tracked utilizing asset tracking.

1.2 State of research

This section briefly introduces the reader to the research fields of feature detection, object detection, and multiple object tracking, as these are essential components to the novel visual asset tracking pipeline. The research on adverse weather conditions and object occlusions are also reviewed as these are known to degrade the performance of vision algorithms.

The section below on object detection, adverse weather conditions, and object occlusions are copied with a few modifications from the preliminary work in the project thesis *"Spot, a mobile four-legged asset tracking robot in adverse weather conditions"* [42]. These sections are included due to the high relevance to this work and to gather all the details of the pipeline in one single document, with the intent of improving the reading flow.

Feature detection

Global features are a popular choice in research fields such as image retrieval, where the entire image is described using techniques such as color histograms. However, such features cannot separate foreground and background and are not suitable for cluttered or occluded images. By contrast, local features are image patterns that differ from their immediate neighborhood and are prevalent in tasks such as Three-Dimensional (3D) reconstruction and camera calibration. This is due to the tailoring of the detectors to accurately locate the same set of features over time in a stable manner [83].

The literature on local feature detection goes as far back as 1954 when the paper *Some informational aspect of visual perception* [3] was published. This paper proposed that information on shape can be described by dominant points with high curvature. In 1988 the well-known Harris corner detector [26] was introduced, which detected corners in images based on intensity variations.

The drawback of many methods is that they require the calculation of derivatives or more complex measures such as the auto-correlation matrix for the Harris detector. Consequently, in the last few decades, the research has centered on developing detectors that achieve computational efficiency. The Scale-Invariant Feature Transform (SIFT) [56] was published in 1999 and utilized the Difference of Gaussians to approximate the Laplacian, also known as the trace of the intensity hessian matrix. Following the inherent success of SIFT, other efficient detectors with different properties such as Histogram of Oriented Gradients (HOG)(2005) [15], Speeded Up Robust Features (SURF)(2007) [5], Binary Robust Independent Elementary Features (BRIEF)(2010) [11] and Oriented FAST and Rotated BRIEF (ORB)(2011) [72] emerged.

In [29] the authors evaluate the performance of BRIEF, ORB, SIFT, and SURF in the task of Visual Simultaneous Localization and Mapping (SLAM). Their analysis revealed that SIFT achieved a lower localization error than the other methods at the cost of a higher computational load. In [41] the authors evaluate the matching performance using SIFT, Principal Component Analysis-SIFT, and SURF. This analysis suggested that SIFT is preferred in situations of changing scale, rotation, and motion blur.

Object detection

The task of detecting objects in images has been around for decades. In the paper *The Representation and Matching of Pictorial Structures* [18] published in 1973, the authors solve the problem of finding specific objects in a photograph by utilizing a template-based technique.

The publication of SIFT in 1999 marked the start of a new era for object detection. The proposed way of extracting local features in an image for classification was much more robust and efficient than previous methods. Other descriptors such as HOG, SURF, BRIEF, and ORB gave rise to more robust and efficient feature extraction in the following years.

Using these descriptive schemes and handcrafted features for object detection has been the state of the art until a transition took place in 2012, with the introduction of the Deep Convolutional Neural Network (DCNN) AlexNet [45]. This DCNN achieved record-breaking results in the Large Scale Visual Recognition Challenge. The authors utilized GPUs for training, making the computationally expensive model feasible.

Since then, the research in object detection has generally been divided into one-stage detection frameworks and two-stage detection frameworks [50]. Two-stage detection frameworks first locate category-independent regions of interest. Features are then extracted from these regions and sent to a category-dependent classifier to classify the proposed regions. Examples of such well known object detectors are Region with Convolutional Neural Network features (RCNN) (2013) [22], Fast RCNN (2015) [21], Faster RCNN (2015) [71], Region-based Fully Convolutional Networks (RFCN) (2016) [14] and Mask RCNN (2017) [31]. Even though successive iterations of this family of object detectors improved accuracy and efficiency, they could not compete with the efficiency of one-stage detection frameworks. These one-stage detectors consist of architectures that predict bounding boxes and corresponding class labels in a single feed-forward DCNN. Such methods jointly learn the important features for predicting bounding boxes and corresponding class labels, which removes the need for separate processing stages. Examples of such well known one-stage detectors are OverFeat (2014) [74], Single Shot Multibox Detector (2015) [54], You Only Look Once (YOLO) (2016) [70], YOLOv2 (2016) [68], YOLOv3 (2018) [69] and YOLOv4 (2020) [7].

In 2020 researchers from Google Brain published the revised paper *Effi-*

cientDet: Scalable and Efficient Object Detection [81], where they developed a new family of one-stage detectors named EfficientDet D0-D7x. This work's main contributions were the introduction of a novel multi-scale feature-fusion method and a scaling approach that scaled the entire network using a single parameter. The resulting EfficientDet D7x achieved a SOTA score on the Common Objects in Context (COCO) dataset with an average precision (AP) of 55.1% using a Tesla V100 GPU. The competing YOLOv4 model on the other hand, achieved a maximum AP of 43.5% at 62 FPS on the same GPU. Comparable speed is achieved using the minimal EfficientDet-D0 model, running at 62.5 FPS with an AP of 33.8%. This indicates that the EfficientDet family provides accurate and flexible detectors, while YOLOv4 sacrifices accuracy for speed.

Since the publication of EfficientDet in 2020 and the preliminary project thesis, a new SOTA method based on transformers has emerged. This SwinV2-G [89] method developed by Microsoft achieves an AP score of 63.1% on the COCO data set. The recent success of transformers for object detection challenges the traditional CNN architecture and suggests that future research in object detection might revolve around transformers.

Multiple Object Tracking

Multiple Object Tracking (MOT) has gained increased attention in recent years due to its academic and commercial potential. This task of detecting objects, maintaining their identities, and yielding a trajectory given a video sequence, includes the four key challenges of object occlusions, initialization and termination of tracks, similar object appearance, and interaction among objects [58].

These challenges have traditionally been addressed by utilizing the following MOT components: motion model, appearance model, interaction model, exclusion model, occlusion handling, and inference. In [90] and [9], a linear motion model is assumed in a Data-Driven Markov Chain Monte Carlo and particle filter framework, respectively. A constant linear velocity model is utilized in the Simple Online Realtime Tracking (SORT) algorithm [6], where a Kalman Filter (KF) is used for inference. An appearance model is an essential component for comparing the appearance of tracked objects, and in [79] the authors propose a clustering approach that groups trajectories of image features that belong to the same object. The DeepSORT [88] algorithm is an extension of the SORT algorithm, which includes a trained deep association metric as an appearance model. A social force model is proposed in [33] to model the movements of pedestrians, whereas an exclusion model is proposed in [60] to avoid physical collisions of trajectories.

Due to recent developments of multi-task learning in deep learning, the research on joint detection and tracking using a single network has attracted more attention in the last few years [91]. Track-RCNN [77] adds a re-identification head on top of Mask-RCNN to regress bounding boxes and re-identification features, while Joint Detection and Embedding [86] similarly builds on top of YOLOv3. In 2021 the TransCenter [89] method based on transformers was published, which achieved SOTA results on the MOT20 benchmarking data set. Similar to the field of object detection, this might mark the start of a new era in MOT, where transformers outcompete the CNN methods.

An evident drawback of these learning-based methods is the need for large-scale data sets. Due to approximately 70% of research in MOT targeting the tracking of pedestrians [58], the data sets publicly available reflect this. The commonly used MOT20 pedestrian data set contains 3833 individual tracks composed of over two million bounding boxes[5] and is a result of over five years of research and development. This is different from object detection, where the data sets cover a wide range of categories that can be utilized for transfer learning.

Adverse weather conditions

Robust vision algorithms are required to function well in adverse weather conditions such as rain, snow, ice, shadows, sun glare, and different lighting to be used in an outdoor setting. These conditions are known to degrade the performance of vision algorithms [30][17], which are generally benchmarked under clear weather conditions with favorable lighting. Prior research has focused on switching strategies or methods to transform adverse weather images into clear weather images.

In [67] the authors propose moving object detection of humans, which switches between using an infrared camera and a Red, Green, and Blue (RGB) camera depending on the weather conditions. The switching strategy is based on previous work, where they developed a model to predict the current weather conditions using supervised learning. In [46], the researchers use the YOLO object detector trained on an infrared data set of 3000 labeled images of humans. They achieve an AP_{50} score of 97.93% for all weather conditions. Even though [67] and [46] achieve impressive results in the specific case of detecting humans, more research is needed in detecting other objects with different emissive properties in adverse weather conditions.

Most research has focused on methods that transform adverse weather images into clear weather images. De-fogging of images is found in [12] and de-raining of images in [40]. Even though the papers showcase promising results, [35] shows how de-raining approaches have a negative effect on the detection precision score. The key finding in their work to achieve accurate detections is the lack of labeled data capturing the diversity of rainy conditions.

[5] https://motchallenge.net/data/MOT20/

In [30], the authors propose that adverse weather suffers from poor contrast due to scattering and absorption of light reaching the camera. An attempt to address this is found in the highly cited paper [61], where the authors propose a weather removal algorithm that restores the contrast of images based on a physics model. Their experiments show how the algorithm can restore contrast in foggy and rainy images. Another interesting approach is found in [40], where a contrast enhancement step is performed after removing visible droplets, fog, and snowflakes from the image.

Object occlusions

Once an object overlaps with another object in an image, it suffers from occlusion. Object occlusions are common in real-life situations, particularly in cluttered environments, where many objects are present in the scene.

An attempt to address this problem is found in [34], where they propose a method to recover unoccluded portions in images. Their proposed method uses a CNN to estimate the optical flow between two successive frames, followed by matching occluded and unoccluded portions of the image to restore the latter. However, the proposed method did not lead to satisfactory results.

In [20], the authors argue that occlusions as image augmentation are required for neural networks to achieve a holistic understanding of objects. This is because neural networks are known to learn easy to recognize portions of objects. They compare two such augmentations, Cutout [16], and Hide-and-Seek [78] to different variations of Dropout regularization. Cutout augmentation selects N patches randomly in the image of a predefined size and zeros these pixels out. Hide-and-Seek divides the image into a grid and randomly zeros those patches out with a predefined probability. Based on experiments on the ImageNet [73] data set, Cutout augmentation achieves the highest precision score and outperforms the other Dropout regularization methods.

1.3 Contributions

The five main contributions from this thesis are given below.

Implementation of a novel visual asset tracking pipeline

A novel visual asset tracking pipeline has been implemented to challenge the traditional asset tracking solutions. The proposed pipeline locates, counts, and identifies containers in an outdoor industrial environment by utilizing methods from the field of computer vision, where visual information is acquired using the Spot robotic platform equipped with a 360 camera. Hopefully, this novel approach will pave the way for improved- and new visual asset tracking solutions.

Creation of a custom data set

A custom data set of containers were created to evaluate the performance of the pipeline. The data set is composed of two parts where the first contains labeled images of containers used in the development of a container detector. This set of 584 images was extended from the preliminary work to improve detection accuracy on the images from the 360 camera. The second part includes two videos of containers in sunny- and overcast weather, synchronized robot poses, and camera calibration parameters. Both videos are labeled in widespread object detection- and tracking formats.

Developed an accurate container detector

The developed EfficientDet object detector from the preliminary work was retrained using the extended image data set. The resulting container detector achieves high detection accuracy in sunny- and overcast weather conditions.

Proposal of visual asset tracking metrics

The visual asset tracking evaluation metrics LA, LAGT, IDA, and DM, were proposed due to the absence of existing metrics communicating the performance of the localization, counting, and identification of assets. The proposed metrics were constructed as simple as possible to serve as an initial first step towards future versions of improved visual asset tracking metrics.

Conducted a literature overview

A literature overview in feature detection and multiple object tracking was conducted to showcase some of the most notable developments in the fields and to explore the different methods available for this work.

1.4 Report structure

The remainder of this thesis is structured as follows. Chapter 2 introduces the theory of the various methods from the field of computer vision which are utilized in this work. This chapter is lengthy, and readers which are familiar with computer vision concepts such as image formation, features, triangulation, homographies, Random Sample Consensus (RANSAC), epipolar geometry, and CNNs are welcome to start reading at section 2.10, where the following sections will introduce the fields of object detection, multiple object tracking, and fiducial detection.

In chapter 3 the methodical approach is presented. First, the container asset is described in section 3.1, followed by a brief presentation of the Spot robotic platform in section 3.2. Then the gathered custom data set is presented in section 3.3, which is composed of both images and videos for the development and evaluation of the pipeline. Finally, the proposed pipeline is presented in section 3.4.

The results are presented and discussed in chapter 4, based on the custom data set. Some of the essential components in the pipeline are evaluated utilizing well-proven metrics from the respective fields, followed by a presentation of the final pipeline performance metrics, which were constructed in this work due to the absence of existing suitable metrics. These metrics represent the success of localizing, counting, and identifying the containers in a scene.

Chapter 5 concludes the work in this thesis, where exciting findings and the final pipeline performance are summarized. The thesis is outlined with further work, which contains elements that are anticipated to improve the performance of the pipeline.

Chapter 2

Theory

This chapter introduces the relevant theory and related methods utilized in the proposed pipeline, starting with a brief section on the field of computer vision and the general representation of RGB images. The following sections introduce the concepts of image formation, features, triangulation, homographies, RANSAC, epipolar geometry, and CNNs, which are essential elements in the reconstruction of container models. The final three sections 2.10 through 2.12 presents the respective fields of object detection, multiple object tracking, and fiducial detection.

2.1 Computer vision

The research field of computer vision studies how a high-level understanding of a scene in the real world can be extracted from images. This field has seen rapid advancements in the last decade due to the efficient processing of visual information using GPUs. A commonly used sensor to gather visual information is the RGB camera. This popular genre of cameras is described shortly in section 2.1.1.

2.1.1 RGB representation of images

An RGB camera captures images by measuring the intensity of red, green, and blue color, stored as pixels in a grid-like structure of width W, height H, and depth of three. The intensity values take values in the range $[0, 255]$, depending on the intensity of the respective color. The resulting RGB image and its composition is shown in fig. 2.1.

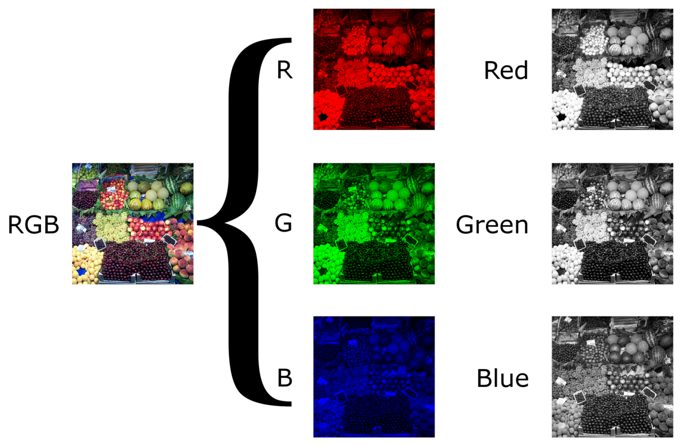


Figure 2.1: An RGB image is a brightness array of size W× H × 3, Source: https://en.wikipedia.org/wiki/Grayscale#/media/File:Beyoglu_4671_tricolor.png (retrieved 2020)

2.2 Homogeneous transformations

In this section, the aim is to mathematically formulate the relationship between coordinate systems defined in Euclidean space by using a homogeneous transformation matrix $\mathbf{T}_{AB}$. This transformation matrix defines the relative orientation and position of the B frame relative to the A frame. For simplicity, we will use the term frame instead of coordinate systems in the remainder of this thesis.

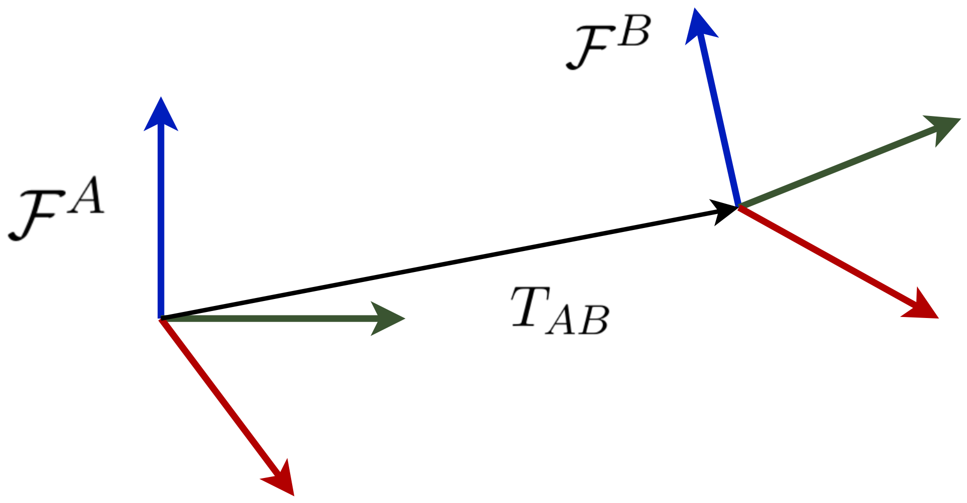


Figure 2.2: The homogeneous transformation $\mathbf{T}_{AB}$ defines the relative orientation and position of frame B relative to frame A.

2.2.1 Rotation

The first part of the homogeneous rigid body transformation matrix is the rotational part $\mathbf{R}$. This rotation matrix, as defined in eq. (2.1) defines the relative orientation of a frame B relative to a frame A. The matrix columns define the orientation of the x, y, and z axes of the B frame relative to the A frame.

$$\mathbf{R}_{AB} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.1}$$

A unique property of this rotation matrix is that it is part of the special orthogonal group, which is defined as shown in eq. (2.2).

$$\text{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}^T\mathbf{R} = \mathbf{I}, det(\mathbf{R}) = 1\} \tag{2.2}$$

2.2.2 Translation

The second part of the transformation matrix is the translational part $\mathbf{t}$. This vector, as shown in eq. (2.3) defines the relative vector between the B frame relative to the A frame.

$$\mathbf{t}_{AB} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \in \mathbb{R}^3 \tag{2.3}$$

2.2.3 The homogeneous transformation matrix

The homogeneous transformation matrix as shown in eq. (2.4). This joint transformation matrix defines the relative translation and rotation between frames A and B.

$$\mathbf{T}_{AB} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

A unique property of this homogeneous transformation matrix is that it is part of the special Euclidean group, which is defined in eq. (2.5).

$$\text{SE}(3) = \{\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} | \mathbf{R} \in \text{SO}(3), \mathbf{t} \in \mathbb{R}^3\} \tag{2.5}$$

2.2.4 Properties of the transformation matrix

The homogeneous transformation matrix has some essential properties, which allows us to easily determine cumulative and inverse transformations.

Inverse

The inverse of a transformation matrix $\mathbf{T}_{AB}$ is given by eq. (2.6) and allows us to determine the orientation and position of frame A relative to frame B.

$$\mathbf{T}_{BA} = (\mathbf{T}_{AB})^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.6}$$

Closure

Matrix multiplication of two or more transformation matrices $\mathbf{T}$ will remain within SO(3) as shown in eq. (2.7).

$$\mathbf{T}_1\mathbf{T}_2 \in SO(3) \tag{2.7}$$

Associative

The order of matrix multiplication is arbitrary and will yield the same final transformation matrix, as seen in eq. (2.8).

$$(\mathbf{T}_1\mathbf{T}_2)\mathbf{T}_3 = \mathbf{T}_1(\mathbf{T}_2\mathbf{T}_3) \tag{2.8}$$

Not commutative

Different order of transformation matrices during matrix multiplications yields different results as shown in eq. (2.9).

$$\mathbf{T}_1\mathbf{T}_2 \neq \mathbf{T}_2\mathbf{T}_1 \tag{2.9}$$

2.2.5 Coordinate transformations

Using the homogeneous transformation matrix $\mathbf{T}_{AB}$, we can transform a point $\mathbf{p}$ in frame A described by the homogeneous vector $\mathbf{X_A} = [x, y, z, 1]$ to frame B by using eq. (2.10).

$$\mathbf{X}_B = (\mathbf{T}_{AB})^{-1}\mathbf{X}_A = \mathbf{T}_{BA}\mathbf{X}_A \tag{2.10}$$

If we also have the transformation matrix $\mathbf{T}_{BC}$, we can combine the transformations as shown in eq. (2.11) to determine the position of $\mathbf{p}$ in the C frame.

$$\mathbf{X}_C = (\mathbf{T}_{BC})^{-1}(\mathbf{T}_{AB})^{-1}\mathbf{X}_A = \mathbf{T}_{CB}\mathbf{T}_{BA}\mathbf{X}_A \tag{2.11}$$

2.3 Image formation

Image formation studies how objects in the real world give rise to images [59]. This is an essential step to develop an accurate model, relating 3D points in the real world to Two-Dimensional (2D) coordinates in an image. In the following section, the projective camera model is presented by a series of generalizations of the simple pinhole camera model. In the end, radial- and tangential distortion models are introduced to model the non-linearities of the camera lens.

2.3.1 The projective camera

A projective camera model **P**, maps homogeneous 3D world points $\mathbf{X} = [\mathrm{X}, \mathrm{Y}, \mathrm{Z}, 1]$ to homogeneous 2D pixels $\mathbf{x} = [x, y, w]$ according to eq. (2.12).

$$\mathbf{x} = \mathbf{P}\mathbf{X} \tag{2.12}$$

In order to determine this camera model **P**, we start with the simplest pinhole camera model, and through a series of generalizations, we end up with the projective camera.

The geometry of the pinhole camera model is shown in fig. 2.3 where the camera centre **C** is located a focal length f behind the image plane with centre **p**. By similar triangles, the point $(\mathrm{X}, \mathrm{Y}, \mathrm{Z})^T$ is mapped to the image plane with coordinates $(f\mathrm{X}/\mathrm{Z}, f\mathrm{Y}/\mathrm{Z})^T$. This can be written as seen in eq. (2.13), where the homogeneous coordinates $x = f\mathrm{X}$, $y = f\mathrm{Y}$, and $w = Z$ are utilized. The left hand side matrix **K** is known as the camera intrinsic parameter matrix, which describes the internal parameters of the camera.

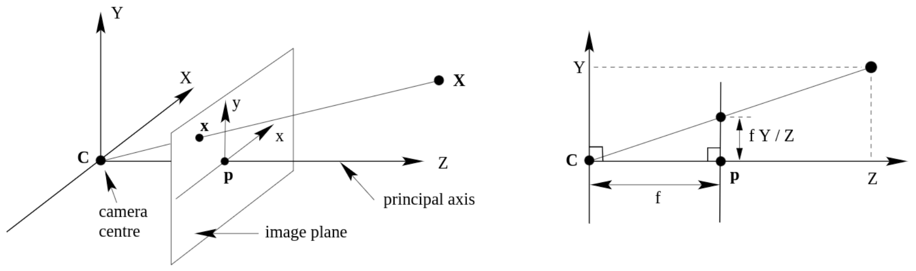


Figure 2.3: The pinhole model geometry. The camera and image centre is denoted by **C** and **p** respectively. Source: [28] (retrieved 2021)

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathrm{K}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{bmatrix} \tag{2.13}$$

The first customary enhancement of this model is to move the principal point center **p** in the image by $(p_x,\ p_y)$. This is achieved by modifying the camera intrinsic matrix **K** as shown in eq. (2.14).

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

The second generalization is to add the possibility of having non-square pixels. By adding the scale factors s_x and s_y to the focal lengths in the x and y direction, we end up with the following intrinsic parameters as shown in eq. (2.15).

$$\mathbf{K} = \begin{bmatrix} fs_x & 0 & p_x \\ 0 & fs_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

The third generalization is to add a skew parameter s to the camera intrinsic parameters. This parameter is often zero or very close to zero for most cameras. The resulting camera matrix is shown in eq. (2.16).

$$\mathbf{K} = \begin{bmatrix} fs_x & s & p_x \\ 0 & fs_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

Finally, the last generalization is to add a Euclidean transformation between the world frame $\mathcal{F}^W$ and the camera frame $\mathcal{F}^C$. This transformation is a homogeneous transformation which was described more in detail in section 2.2. The transformation is also known as the camera extrinsic parameters and is often referred to as the camera pose, which describes the relative orientation and position of the camera relative to the world. The resulting projective camera model is described by eq. (2.17), where we have set $f_x = fs_x$ and $f_y = fs_y$. If the camera intrinsic parameters are known, the model can be used for Euclidean reconstruction, which will preserve the correct Euclidean shape of the reconstructed scene or object [28].

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{bmatrix} \tag{2.17}$$

2.3.2 Camera distortion

The projective model developed in the previous subsection has assumed linear relation between world- and image points. This will not hold for real cameras due to the underlying pinhole model being too simplistic to capture the non-linearities of different camera lenses. Fortunately, these lense non-linearities can be modeled as radial- and tangential distortions to the pixel coordinates.

Radial distortion

The standard model for radial distortion which transforms the normalized distorted pixel coordinates (x_d, y_d) to ideal- or undistorted pixel coordinates (x, y) is shown in eq. (2.18) [92]. In this equation, $L(r)$ is commonly chosen as the even Taylor expansion $L(r) = 1+k_1r^2+k_2r^4+k_2r^6$, were $r^2 = x^2+y^2$, and k_1, k_2, k_3 is the radial distortion parameters.

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = L(r) \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.18}$$

In fig. 2.4, an un-distorted image with ideal coordinates is shown in (a), whereas in (b) and (c), positive and negative radial distortion is displayed.

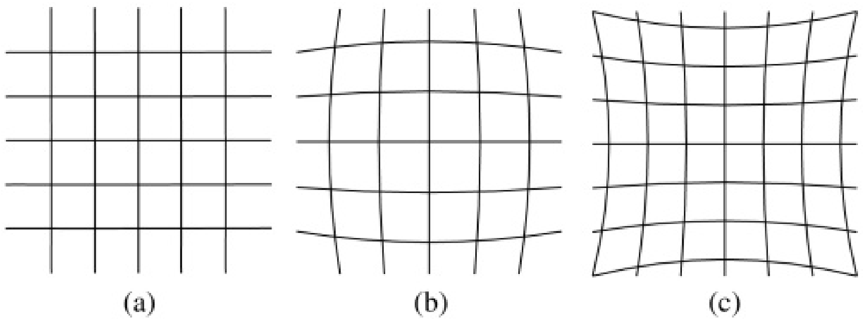


Figure 2.4: (a) No radial distortion, (b) Positive radial distortion, (c) Negative radial distortion Source: [24] (retrieved 2021)

Tangential distortion

Tangential distortion is another non-linearity that is essential to model to achieve an improved linear relation between pixels and real-world coordinates. This distortion is due to a misalignment between the lens and the charge-coupled device image sensor, as shown in fig. 2.5. The tangential distortion model is shown in eq. (2.19) [32], where p_1 and p_2 is the tangential distortion parameters.

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} x + 2p_1xy + p_2(r^2 + 2x^2) \\ y + p_1(r^2 + 2y^2) + 2p_2xy \end{bmatrix} \tag{2.19}$$

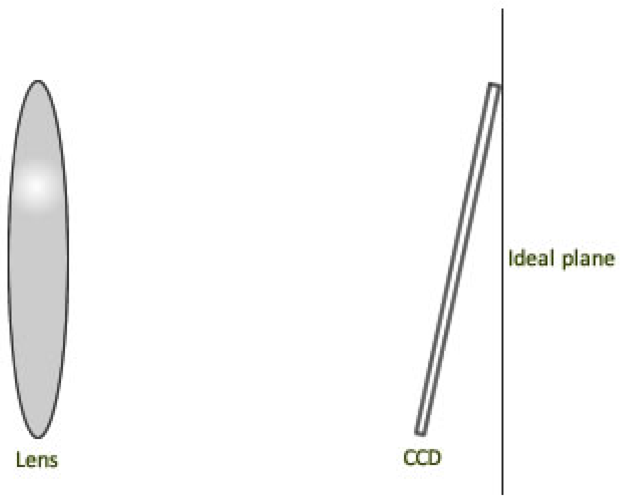


Figure 2.5: Tangential distortion is due to a misalignment of the camera lens and the charge-coupled device camera sensor. Ideally, the sensor should be parallell to the camera lense. Source: `https://aishack.in/tutorials/major-physical-defects-cameras/` (retrieved 2021)

2.4 Feature detection, description, and matching

The detection, description, and matching of features in images are essential tasks in computer vision. This section introduces the notion of detecting features, also referred to as interest points. The auto-correlation matrix is introduced to give the reader an intuition on how intensity gradients of image patches can be used to locate edges and corners. This is followed by the description of features, which is a requirement for matching. The popular SIFT detector and descriptor is briefly introduced followed by the process of matching features.

2.4.1 Feature detection

The process of extracting features in images is known as feature detection. A feature is an interesting point in an image, such as an edge or corner that are likely to be found and matched well in other images. Two of the essential properties of a feature detector are compactness and efficiency. This can be achieved by sliding a window over the image to find interesting patches. In fig. 2.6, two images with three extracted patches are shown. From these patches, notice how the second and third patch can be localized and matched more accurately as it carries more texture than the first patch.

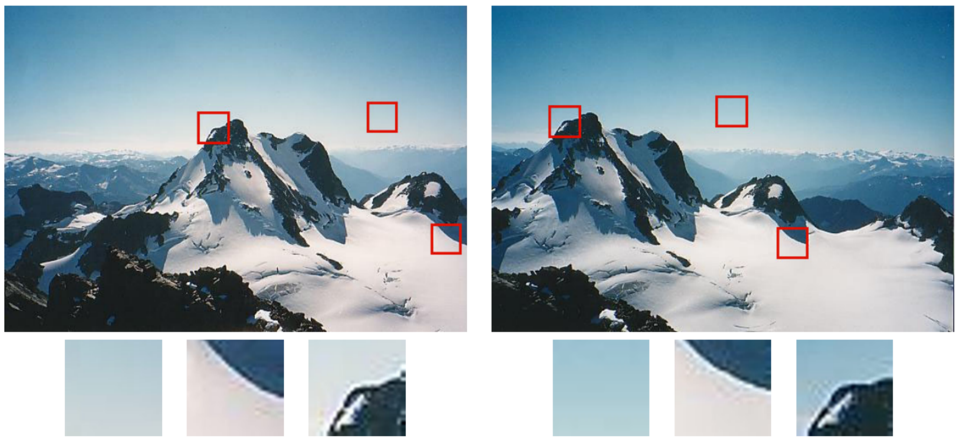


Figure 2.6: Two images with three extracted image patches. Notice how the second and third patch can be localized and matched more accurately. Source: [80], (retrieved 2021)

The traditional auto-correlation function E_{AC} as given in eq. (2.20) can be used to detect features based on the intensity values I_0 of the pixels. This function compares image patches against itself by applying a small variation in the pixel position $\Delta\mathbf{u}$, where $w_i(\mathbf{x}_i)$ is a weighting function and the summation i is over all the pixels in the patch.

$$E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i)[I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x_i})]^2 \tag{2.20}$$

By utilizing the Taylor expansion $I_0(\mathbf{x}_i + \Delta\mathbf{u}) \approx I_0(\mathbf{x}_i) + \Delta I_0(\mathbf{x}_i)$, the function can be written as seen in eq. (2.21), where $\mathbf{A}$ is known as the auto-correlation matrix. The image gradients can be computed by using a variety of techniques. The classic *Harris* detector uses a [-2, -1, 0, 1, 2] filter, but more sophisticated approaches are also common.

$$E_{AC}(\Delta\mathbf{u}) = \Delta\mathbf{u}^T \underbrace{w * \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}}_{\mathbf{A}} \Delta\mathbf{u} \tag{2.21}$$

The eigenvalues (λ_0, λ_1) of the auto-correlation matrix identify the directions of the largest and smallest change of the auto-correlation function. The patch is typically an edge when either eigenvalue is much larger than the other. Flat regions with little texture have small eigenvalues, while corners have large values [75]. Thus the detection of features can be achieved by choosing a criterion for the eigenvalues.

2.4.2 Feature description

After detecting the features, they have to be described to match them across images. In most cases, the features are described using the local appearance of the interesting point. This local appearance will often change in orientation, scale, or undergo an affine deformation. Modern feature descriptors address these issues by extracting a local scale, orientation, or affine frame estimate from the feature image patch. A popular method utilized for both feature detection and matching is SIFT, which will be very briefly introduced below.

SIFT

The SIFT is invariant to scale, orientation, and affine deformations, as well as robust to changes in illumination and noise [56]. This method detects features by extracting the maxima and minima of the Difference of Gaussians function in scale space. This function is applied to an image pyramid consisting of re-sampled and smoothed layers at different scales. For each detection, a 16×16 window at the corresponding scale in the Gaussian pyramid is used to calculate the pixel gradients around the detection. The magnitudes of the gradients are then down-weighted using a Gaussian fall-off function to reduce the influence of gradients far from the center. The patch is then divided into eight quadrants, where gradients are added to one of eight histogram bins in the respective quadrant. These two steps are showcased in fig. 2.7, where the illustration utilizes 8×8 patches and four quadrants. Finally, the gradients are then added to $2 \times 2 \times 2$ histogram bins using trilinear interpolation to reduce the effect of location, and dominant orientation misestimation [80]. This results in a vector of 128 non-negative values used to describe the detected feature.

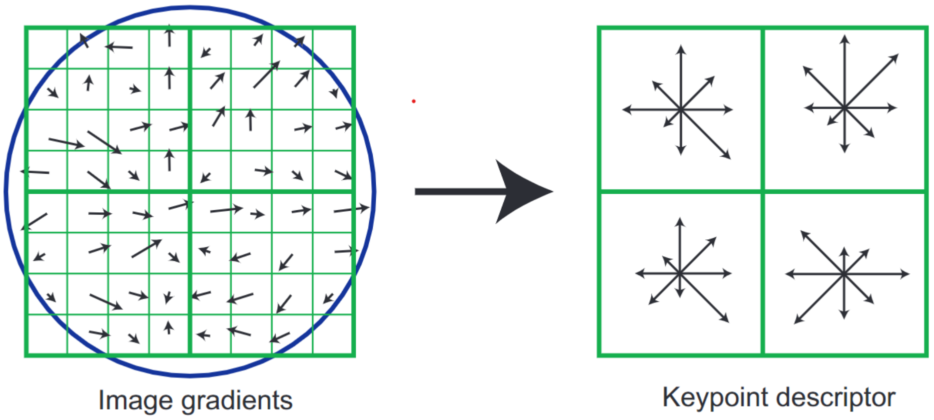


Figure 2.7: The left image shows the SIFT gradient orientations and magnitudes at each pixel, where the blue circle shows the Gaussian fall-off function. The right image shows the weighted gradient orientation histogram computed at each subregion. Note that this illustration utilizes a patch size of 8×8 and four quadrants. Source: [57] (retrieved 2021)

2.4.3 Feature matching

After extracting and describing features in a series of images, the next step is to match the features. We will assume that the descriptors have been designed such that the Euclidean distance between descriptors can be calculated. By calculating the distance between all descriptor pairs in two images, the most straightforward matching strategy is to return the match with the lowest distance above some pre-defined maximum distance threshold. However, setting this maximum distance threshold is difficult and will vary as we move to different parts of the feature space [80]. A better strategy is to utilize the Nearest Neighbor Distance Ratio (NNDR) given by $\frac{d1}{d2}$, where d_1 and d_2 is the nearest and second nearest neighbor distance, respectively. The closest match can be kept or discarded based on the calculated NNDR value, where a lower value indicates a good match. As in the previous method, this requires a maximum threshold value to be set. The creator of SIFT proposed in [57] to reject all matches with an NNDR above 0.80. This resulted in an elimination of 90 % of the false matches while discarding less than five percent of the correct matches.

2.5 Image homographies

Two images of the same planar surface are related by a 2D image homography $\mathbf{H} \in \mathbb{R}^{3\times3}$. This homography allows us to transform points $\mathbf{x}_i$ on the plane in the first image to the corresponding point $\mathbf{x}'_i$ in the second image as seen in eq. (2.22). This matrix has in total eight independent elements which require to be determined [28]. Each point correspondence frees up two degrees of freedom, which results in a minimum number of four points required to estimate the homography. There have been proposed several different algorithms for this estimation. However, the Direct Linear Transformation (DLT) algorithm is arguably the most prominent and commonly used algorithm for this purpose.

$$\mathbf{H}\mathbf{x}_i = \mathbf{x}'_i \tag{2.22}$$

2.5.1 The Direct Linear Transformation algorithm

The DLT is a simple linear algorithm that estimates the homography $\mathbf{H}$ given four or more point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ of a planar surface. These point correspondences can be determined by matching of features across views as introduced in section 2.4.3. The linear solution can be formed by the cross product $\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i$=0, resulting in three equations for each correspondence as seen in eq. (2.23), where $\mathbf{h}_1^T, \mathbf{h}_1^T, \mathbf{h}_1^T$ are the rows of $\mathbf{H}$.

$$\begin{bmatrix} \mathbf{0}^T & -w'_i\mathbf{x}_i^T & y'_i\mathbf{x}_i^T \\ w'_i\mathbf{x}_i^T & \mathbf{0}^T & -x'_i\mathbf{x}_i \\ -y'_i\mathbf{x}_i^T & x'_i\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{h_1} \\ \mathbf{h_2} \\ \mathbf{h_3} \end{bmatrix} = \mathbf{0} \tag{2.23}$$

The third row in this linear system can be omitted as it is linearly dependent on the first and second rows. The resulting system takes the form $\mathbf{A}_i\mathbf{h} = 0$, where $\mathbf{A}_i$ is a 2×9 matrix, and $\mathbf{h}$ a 9×1 vector containing the elements of $\mathbf{H}$.

By assembling all $\mathbf{A}_i$ for the $n \geq 4$ correspondences, a $2n \times 9$ matrix is constructed. Due to noise in the measurement of correspondences, the linear system will not have an exact solution other than the trivial zero solution. Additionally, as $n > 4$, the system becomes over-determined. To address these issues, the DLT algorithm utilizes the singular value decomposition. This decomposition minimizes the norm $||\mathbf{Ah}||$ subject to the constraint $||\mathbf{h}|| = 1$, where the solution is the unit singular vector corresponding to the smallest singular value of the decomposition. The resulting DLT algorithm is given as:

1. For each image correspondence $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$, construct $\mathbf{A}_i$.

2. Assemble $\mathbf{A}_1$, .., $\mathbf{A}_N$ in to a single matrix $\mathbf{A}$ of size $2n \times 9$.

3. Obtain the singular value decomposition of $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$.

4. The "best" linear estimate $\hat{\mathbf{h}}$ can then be extracted from the decomposition as the last column of $\mathbf{V}$. This is the unit singular vector corresponding to the smallest singular value for the solution $\mathbf{h}$.

Additionally, it is customary to normalize the point correspondences utilizing a transformation such that the centroid of the points is at the origin with an average distance to the origin of $\sqrt{2}$. This extension is known as the normalized-DLT and is considered as a mandatory extension in the estimation of the homography [28].

2.6 Structure computation

This section introduces the theory necessary to estimate the 3D position of a point $\mathbf{X}$ imaged in two and up to M views. For each view, we require the corresponding image point $\mathbf{x}_1, .., \mathbf{x}_M$ in order to utilize the projective camera model as introduced in eq. (2.17), relating a 2D coordinate in an image to a 3D coordinate in the world frame. Thus the problem becomes:

Given image correspondences $\mathbf{x}_1, .., \mathbf{x}_M$ for a 3D point $\mathbf{X}$ and the projection matrices $\mathbf{P}_1, .., \mathbf{P}_M$, estimate $\hat{\mathbf{X}}$.

In fig. 2.8 the problem is showcased for two cameras with image correspondences $\mathbf{x}$ and $\mathbf{x}'$. From the figure, it is evident that there is no exact solution for $\mathbf{X}$ due to the non-intersecting rays. This is always the case, explained by errors and noise in the camera model and the image correspondences. Thus the best one can do is to estimate $\hat{\mathbf{X}}$, where the accuracy of this estimate is governed by the method of triangulation used. A standard way of achieving a good estimate is first to do an initial reconstruction using a simple method such as linear triangulation, then use this estimate as a starting point in a non-linear optimization scheme such as Bundle Adjustment (BA).

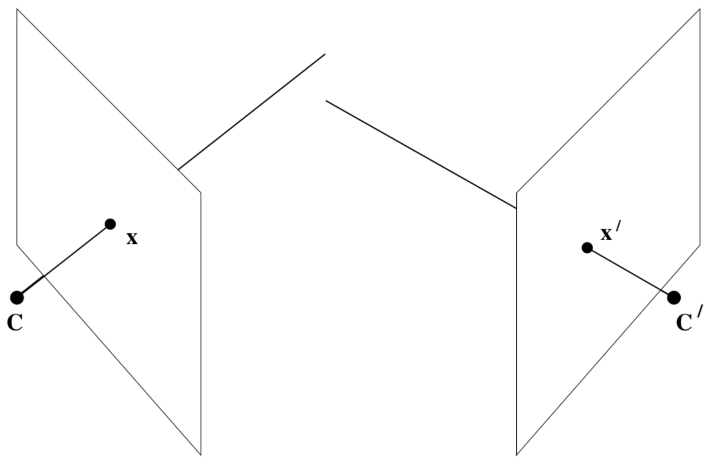


Figure 2.8: Two point correspondences $\mathbf{x}$ and $\mathbf{x}'$ forms two non-intersecting rays. Source: [28] (retrieved 2021)

2.6.1 Initial linear reconstruction

Linear triangulation is a popular method used to estimate an initial set of 3D points due to the simplicity of the method and generalizing easily to M-views. The goal of this method is to form a system $\mathbf{AX} = 0$ which can be solved for $\mathbf{X}$ by utilizing the DLT algorithm as introduced in section 2.5.1. This minimizes an algebraic error, which has no statistical or geometric interpretation but is shown to yield a good starting point for a non-linear optimization procedure.

To form this $\mathbf{A}$ matrix we first utilize the cross product $\mathbf{x} \times \mathbf{PX} = 0$ to give rise to three equations for each image point $\mathbf{x}_i$ as shown in eq. (2.24), where $\mathbf{P}_i^{1T..3T}$ are used to denote the columns of $\mathbf{P}_i$.

$$\begin{bmatrix} 0 & -1 & y_i \\ 1 & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_i^{1T} \\ \mathbf{P}_i^{2T} \\ \mathbf{P}_i^{3T} \end{bmatrix} \mathbf{X} = 0 \tag{2.24}$$

This gives rise to two linearly independent equations as shown in eq. (2.25) where the number of unknowns is higher than the number of equations, and the linear system is thus under-determined using only the equations for one view.

$$\begin{bmatrix} x_i\mathbf{P}^{3\mathrm{T}} - \mathbf{P}^{1\mathrm{T}} \\ y_i\mathbf{P}^{3\mathrm{T}} - \mathbf{P}^{2\mathrm{T}} \end{bmatrix} \mathbf{X} = 0 \tag{2.25}$$

The solution is to add the equations for the remaining M-1 views to make the system over-determined instead. For M views, the resulting $\mathbf{A}$ matrix takes the following form as shown in eq. (2.26), which can be solved for $\mathbf{X}$ using the DLT.

$$\begin{bmatrix} x_1\mathbf{P}_1^{3\mathrm{T}} - \mathbf{P}_1^{1\mathrm{T}} \\ y_1\mathbf{P}_1^{3\mathrm{T}} - \mathbf{P}_1^{2\mathrm{T}} \\ \vdots \\ x_M\mathbf{P}_M^{3\mathrm{T}} - \mathbf{P}_M^{1\mathrm{T}} \\ y_M\mathbf{P}_M^{3\mathrm{T}} - \mathbf{P}_M^{2\mathrm{T}} \end{bmatrix} \mathbf{X} = 0 \tag{2.26}$$

2.6.2 Optimal reconstruction

The initial reconstruction in the previous subsection minimized the algebraic error using the homogeneous DLT. Though this reconstruction often yields good results, a final polish using an iterative nonlinear optimization is often desirable. This is achieved by minimization of a geometric reprojection error as shown in eq. (2.27), which iteratively calculates corrections in order to align correspondences as shown in fig. 2.9, where $d(\,\cdot\,)$ is the euclidean distance. In general, such methods are known as BA, which solves a nonlinear least-squares minimization problem of the reprojection error by refining over the both the 3D points $\mathbf{X}_j$ and the projection matrices $\mathbf{P}_i$. In the case of refining only over the 3D points $\mathbf{X}_j$, the resulting method is known as Structure Only Bundle Adjustment (SOBA).

$$\min \sum_{ij} d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 \tag{2.27}$$

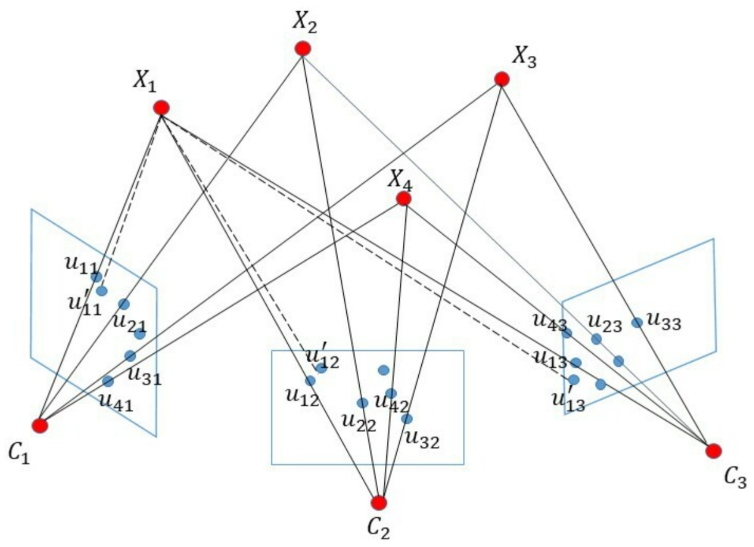


Figure 2.9: Three views C_j with correspondences $\mathbf{u}_{ij}$ and $\mathbf{X}_i$ the 3D points in the world frame. The solid lines represent the projection, while the dotted lines the reprojections. The reprojection error is thus this distance between the projections and the reprojections. Source: [13] (retrieved 2021)

Structure Only Bundle Adjustment

SOBA is a special case of BA where we only refine the 3D points $\mathbf{X}_j$ as shown in eq. (2.28). This is a nonlinear least-squares problem [13] which requires an iterative minimization method in order to find a satisfying solution. By satisfying, it is implied that the found solution might be a local optimum and thus not the global optimal solution [28].

$$\underset{\mathbf{X}_i}{min} \sum_{i=1}^{n} \sum_{j=1}^{m} (\mathbf{u}_{ij} - \mathbf{P}_j \mathbf{X}_i)^2 \tag{2.28}$$

It is customary to write the minimization problem above in a simpler form as seen in eq. (2.29) using residuals $\mathbf{r}_{ij} = \mathbf{u}_{ij} - \mathbf{P}_j \mathbf{X}_i$. The resulting vector $\mathbf{r}$ is known as a residual vector.

$$min\ \mathbf{r}^T \mathbf{r} \tag{2.29}$$

By performing a Taylor expansion of $\mathbf{r}$, followed by taking the derivative and setting this to zero, we obtain obtain the general Gauss-Newton equation as shown in eq. (2.30) for finding the search direction $\delta\mathbf{x}$, where the hessian approximation $\mathbf{H} \approx \mathbf{J}^T \mathbf{J}$ is utilized.

$$\mathbf{J}^T \mathbf{J} \delta\mathbf{x} = -\mathbf{J}^T \mathbf{r} \tag{2.30}$$

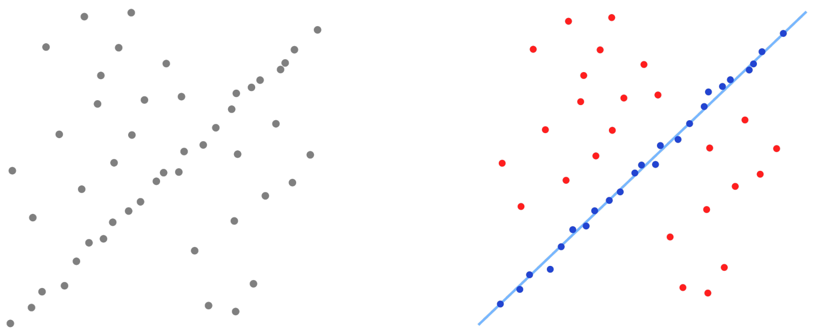
The Gauss-Newton method will always produce a descent direction as long as $\mathbf{J}^T \mathbf{J}$ is positive definite, but when this term is singular, this method becomes unstable. The Levenberg-Marquardt algorithm solves this problem by a modification of the Hessian approximation as seen in eq. (2.31), where the parameter λ is added to avoid rank deficiency. This is known as a trust-region approach, where the parameter λ is increased when a search direction reduces the objective function and decreased otherwise. This is the preferred method for solving the nonlinear least-squares problem in eq. (2.28) [2], where the Jacobian is commonly approximated by finite-differencing [4].

$$(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}) \delta\mathbf{x} = -\mathbf{J}^T \mathbf{r} \tag{2.31}$$

2.7 Random sample consensus

RANSAC is an iterative method used to fit experimental data to a mathematical model [19]. This is a popular method in robust estimation, where data outliers need to be separated from the inliers. The algorithm randomly draws the minimum number of data points required for each iteration, determines the number of inliers for this particular model, and updates the best model parameters if there are more inliers in the current model.

An example where RANSAC is used to fit data points to a line described by the equation $y = ax + b$ is shown in fig. 2.10. From the initial data points shown in fig. 2.10a, two points are randomly sampled for each iteration, and the number of inliers within a threshold of the model is counted. The final model with the highest amount of inliers is shown in fig. 2.10b.



(a) Data points (b) Line estimate

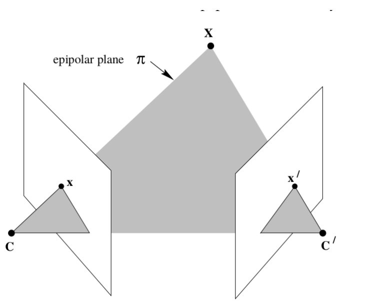
Figure 2.10: (a): The initial set of data points. (b): The fitted line using RANSAC. Source: https://en.wikipedia.org/wiki/Random_sample_consensus (retrieved 2021)

2.8 Epipolar geometry

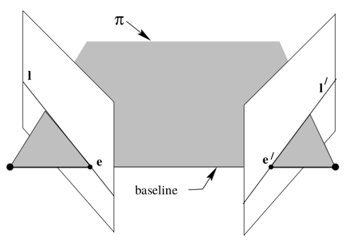
Epipolar geometry is the projective geometry between two views that only depends on the camera's intrinsic and extrinsic parameters. Given point correspondences $\mathbf{x}$ and $\mathbf{x}'$ for two images, the relationship between these two points satisfies what we call the epipolar constraint as seen in eq. (2.32). The matrix $\mathbf{F}$ is known as the fundamental matrix and entails the relative motion between the two frames as well as the internal parameters of the cameras.

$$\mathbf{x}'^{\mathrm{T}}\mathbf{F}\mathbf{x} = 0 \tag{2.32}$$

An important property for the epipolar constraint is that the correspondences and the 3D points $\mathbf{X}$ all lies on an epipolar plane π as shown in fig. 2.11(a). This epipolar plane intersects the epipoles $\mathbf{e}$ and $\mathbf{e}'$ on the baseline as shown in fig. 2.11(b).



(a) The correspondences and 3D point lies on the epipolar plane π.



(b) The epipolar lines l and l' and the epipoles e and e'.

Figure 2.11: The epipolar geometry between two views. Source: [28] (retrieved 2021)

Another important property is that a point $\mathbf{x}$ forms an epipolar line in the other view C' as shown in fig. 2.11(b). This means that given a point $\mathbf{x}$, we can determine a line in the other image where the correspondence should lie on. This is a useful property that is often utilized when matching point correspondences between views.

2.8.1 The fundamental matrix

The fundamental matrix $\mathbf{F}$ is defined according to eq. (2.33). Intrinsic parameters are given by $\mathbf{K}$ and $\mathbf{K}'$ and the relative motion between the two views as $\lfloor \mathbf{R}, \mathbf{T} \rfloor$. The term $\left[\mathbf{t}\right]_{\times} \mathbf{R}$ is also known as the essential matrix, encoding the relative motion between the two views.

$$\mathbf{F} = \mathbf{K}'^{\text{-T}} \left[\mathbf{t}\right]_{\times} \mathbf{R}\mathbf{K}^{-1} \tag{2.33}$$

The epipolar lines can be computed using the fundamental matrix using the equations shown in eq. (2.34).

$$\begin{aligned} \mathbf{l}' &= \mathbf{F}\mathbf{x} \\ \mathbf{l} &= \mathbf{F}^{\text{T}}\mathbf{x}' \end{aligned} \tag{2.34}$$

If the relative motion between the two views and the internal parameters are available, it is straightforward to calculate the fundamental matrix. When these parameters are not known, a popular approach is to estimate the matrix using the normalized 8-point algorithm [27].

2.9 Deep learning in computer vision

Deep learning has become an essential tool in computer vision due to the inherent success of convolutional neural networks inspired by the structure of the human visual system [84].

This particular section 2.9 is copied with a few modifications from the preliminary work in the project thesis *"Spot, a mobile four-legged asset tracking robot in adverse weather conditions"*[42]. These sections are included due to the high relevance to this work and to gather all the details of the pipeline in one single document, with the intent of improving the reading flow.

2.9.1 Convolutional neural network

A convolutional neural network mainly consists of three layers, convolutional layers, pooling layers, and fully connected layers. Figure 2.12 shows the typical structure of such a network. The input image is convolved using convolutional layers resulting in feature maps of the same dimension as the image. The feature maps are then down-sampled through pooling layers before the same process is repeated multiple times. The last feature maps are then connected to a fully connected layer.

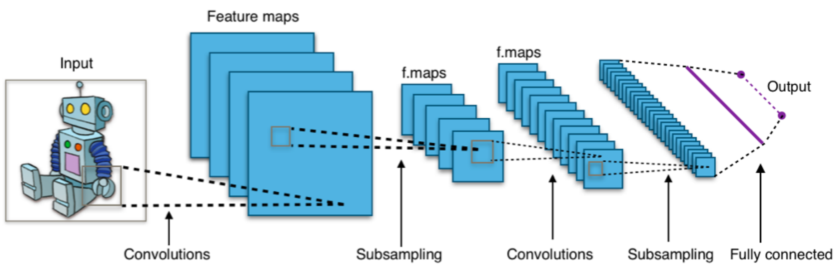


Figure 2.12: The structure of a typical convolutional neural network. Feature maps are generated using successive convolutional- and pooling layers. The output from the network is generated using a fully connected layer. Source: https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png (retrieved 2020)

Convolutional layer

A convolutional layer consists of N kernels or filters of which the input is convolved to produce the output feature maps. This results in a total of N feature maps with the same spatial dimension as the layer's input. Each layer generates various feature maps that store essential information in the prediction of new images.

Pooling layer

A pooling layer reduces the spatial dimension of the input feature maps. The depth or number of feature maps remains fixed in the process. The dimension reduction is achieved by using the concept of stride. Stride states the number of activations to skip from one convolution to the next using a square filter. Selecting a stride of one results in no dimension reduction, while selecting a stride of two halves the dimension. Two popular pooling techniques are max pooling and average pooling. Both techniques run a filter of size $M \times M$ on the input with a stride larger than one. Max pooling sets the output activation value equal to the maximum activation value within the filter. Average pooling sets the output activation value equal to the average of the activations within the filter.

Reducing the spatial dimension is also known as downsampling. Despite the loss of information in the process, pooling reduces the computational overhead and serves as a regularizer against overfitting, which will be considered in section 2.9.4.

Fully connected layer

A fully connected layer reduces the 2D feature maps in the previous layer to a One-Dimensional (1D) feature vector, which can be used directly for classification or further processing. This is achieved by connecting each neuron to all the activations in the previous layer, hence the name.

2.9.2 Training

Training a DCNN refers to the process of changing the weights within the network such that we get the desired output given the input. This process is also known as learning, as the network changes the weights within the network to get better at the task it is given in a supervised-learning theme. To achieve this, we need to introduce the concept of the loss function, Stochastic Gradient Descent (SGD), and backpropagation.

Loss function

A loss- or cost function $C(\mathbf{w})$ measures the difference between the predicted values $\hat{y}_i$ and the desired values y_i. Mean squared error is one of the simplest and most common loss functions given in eq. (2.35).

$$C(\mathbf{w})_{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.35}$$

Stochastic gradient descent

The SGD algorithm used to optimize the loss function is a standard iterative scheme. This algorithm uses an estimate of the loss function gradient to update the weights in the network. The resulting iterative update scheme is seen in eq. (2.36).

$$\mathbf{w}_i = \mathbf{w}_{i-1} - \frac{\eta}{n}\sum_{i=1}^{n} \nabla C_i(\mathbf{w}) \tag{2.36}$$

In this formula, the n states the number of samples passed through the network and η the learning rate. In order to calculate the gradient, a common concept of backpropagation is utilized. An extension to this method is to add a momentum term $\alpha\Delta\mathbf{w}$. In this term, the α is a decay factor between zero and one that determines the contribution of the $\Delta\mathbf{w}$ term to the new weights. This $\Delta\mathbf{w}$ term is determined through a linear combination of the previous update and the gradient.

Backpropagation

Backpropagation computes the gradient used to optimize the loss function with respect to the weights efficiently. This is achieved by utilizing the chain rule to calculate the gradient of the weights one layer at a time, starting from the last layer. By iterating backward through the layers, intermediate computations and redundant computations are avoided.

2.9.3 Hyperparameters

Hyperparameters refer to the parameters that must be determined outside the learning algorithm itself [23]. These parameters must be chosen carefully, as they control the behavior of the learning process. Finding the most optimal hyperparameters in deep learning is called hyperparameter optimization. Conventional algorithms for this purpose are grid search, random search, and Bayesian optimization. The simplest way of selecting these parameters is through manual search. Some important hyperparameters will be introduced below.

Hidden layers

A neural network's hidden layers refer to the layers between the input and output layers. Adding hidden layers to the network generally improves the network's accuracy but increases computational cost as the number of parameters in the network increases.

Learning rate

A very important hyperparameter in deep learning is the learning rate. This parameter $\eta_t \in \mathbb{R}$ in eq. (2.36) determines the step size for each update of the model parameters by the SGD algorithm. Choosing a large step size can cause instability in the optimization problem while choosing a small one will result in slower convergence towards a local optimum. We are likely to be far from the local optimum at the start of the learning process, and large step size is therefore desirable. After some iterations, we would like the optimizer to slow down to accurately navigate to the local optimum, avoiding any zig-zagging instabilities. Dynamically adjusting this learning rate is the status quo of addressing this problem. Restarting schemes such as the function scheme restarts the learning rate whenever an increase in the cost function is observed. In contrast, the gradient scheme utilizes a more sophisticated scheme based on the angle between the momentum term and the negative gradient to reset the learning rate. A popular choice for dynamically adjusting the learning rate is cosine decay [55], which is given by eq. (2.37).

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + cos(\frac{T_{cur}}{T_i}\pi)) \tag{2.37}$$

In this update scheme, T_i denotes the number of epochs to wait until an update, while T_{cur} states the number of epochs since the last update. The maximum- and the minimum learning rate is given by η^i_{max} and η^i_{min} respectively.

Batch size

Updating the network parameters after passing through a single input is an expensive operation and will considerably slow down the training process. Minimizing the number of updates necessary is therefore important to speed up the training process. One method to address this problem is to split the data set into batches of constant size. Each batch will then be fed through the network, and an update of the model parameters is executed once the entire batch is processed. Larger batch sizes speed up the training process but come at the cost of reducing the training's accuracy. When multiplying the batch size N with a factor k, the authors in [44] suggests using what they refer to as a *weird trick*. This trick states that the learning rate η should be multiplied by $\sqrt{k}$ as seen in eq. (2.38).

$$\eta = \eta_{prev}\sqrt{k} \tag{2.38}$$

Epoch

An epoch is when the entire data set is passed through the network once, including backpropagation to adjust the weights. As the learning process is iterative, there is no guarantee for the optimizer to locate a local optimum in a single epoch. Passing the same data set through the network N times is known as the epoch number, which has proven essential to maximize the learning from the data set.

2.9.4 The overfitting problem

A central goal in deep learning is to develop a model that performs well on new and unseen data, which is also known as generalization [23]. The labeled training set is usually split into a training and validation set when training a model. This allows us to compute an error measure on both data sets, called training- and validation errors. Monitoring the change of these two metrics is essential to obtain a good model. In an ideal training situation, both metrics keep decreasing at the same rate. If the evaluation error starts diverging from the training error, the model suffers from overfitting as illustrated in fig. 2.13. This is when the model becomes overly dependent on the training data and will likely fail on new unseen data. Methods to address this problem are known as regularization techniques.

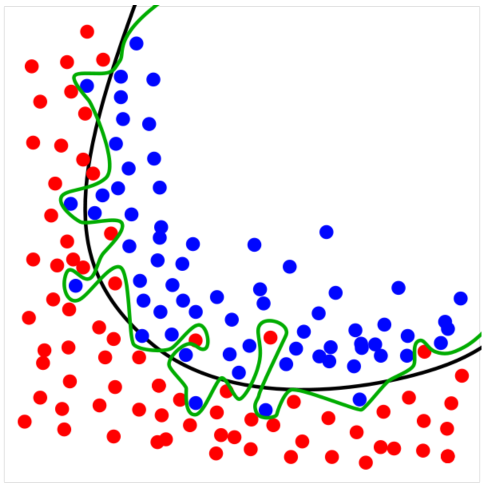


Figure 2.13: In two dimensions, the overfitted green model separates the two classes in red and blue but is too dependent on the training data, which will most likely fail on new unseen data. The regularized model is shown in black and will likely perform much better. Source: https://commons.wikimedia.org/wiki/File:Overfitting.svg (retrieved 2020)

2.9.5 Regularization

Regularization is defined as any modification we make to a learning algorithm to decrease the validation error, but not its training error. Some essential regularization techniques will be introduced below.

Early stopping

Early stopping is the most common regularization technique, where the training process is stopped once the validation error starts diverging from the training error. This technique is illustrated in fig. 2.14, where the blue arrow indicates when the training process should stop to prevent overfitting.

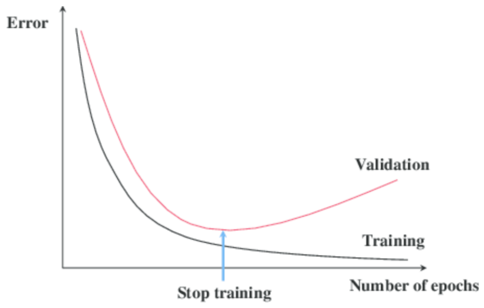


Figure 2.14: Early stopping regularization suggests stopping the training when the validation error starts diverging from the training error. Source: https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd (retrieved 2020)

L2 regularization

L2 regularization is a popular regularization technique known as weight decay. This can be implemented by including the term $\Omega(\mathbf{w})$ in eq. (2.39) to the loss function. The term α denotes the regularization rate, where a higher rate encourages smaller weights.

$$\Omega(\mathbf{w}) = \frac{\alpha}{2}||\mathbf{w}||_2^2 \tag{2.39}$$

Batch normalization

Batch normalization refers to the method of normalizing the activations in the network for each batch processed by the network [38]. This allows for much higher learning rates and reduces the importance of network initialization.

Image augmentation

It is generally an accepted notion that more extensive data sets improve the performance of deep learning models [25]. Image augmentation techniques exploit this by assuming that there is more information to gather from the original data set through augmentations [76]. This augmentation of the data set is achieved by either image warping- or oversampling. Image warping techniques perform augmentation directly on training instances while preserving the label. Common image warping techniques are random cropping, horizontal flipping, and color space augmentations. Oversampling techniques adds synthetic instances to the training set, which are independent of the original data set. This type of augmentation is typically used when very limited or no real data is available. This is verified in [64] where the researchers analyze the influence of increasing the amount of synthetic data in a fixed-size training set of 13K images. By increasing the amount of synthetic data and reducing the amount of real data, they observe a general trend of a dramatic sacrifice in model performance.

2.9.6 Transfer learning

Transfer learning is a widely used method that reuses a model developed for a specific task as the initial starting point for a new model on another task. The most common transfer learning approach is to use a pre-trained model, which is trained on an extensive data set such as COCO [51], ImageNet [73], or Modified National Institute of Standards and Technology [49]. This model is then fine-tuned to fit the task at hand by freezing all but the very last layers during training.

2.10 Object detection

In this section, the object detection problem will be introduced, then the general anatomy of a modern object detector will be investigated. Finally, the EfficientDet object detector and evaluation metrics used in this thesis are presented.

This particular section 2.10 is copied from the preliminary work in the project thesis *"Spot, a mobile four-legged asset tracking robot in adverse weather conditions"*[42]. These sections are included due to the high relevance to this work and to gather all the details of the pipeline in one single document, with the intent of improving the reading flow.

2.10.1 Problem formulation

The object detection problem concerns both object localization and classification in images and is formulated as follows [53].

Given an image, determine whether or not there are instances of objects from predefined categories and, if present, return the spatial location and extent of each instance.

This problem is illustrated in fig. 2.15, where bounding boxes are used to define each detected instance's spatial location and extent. Each bounding box is then labeled according to the predefined categories.

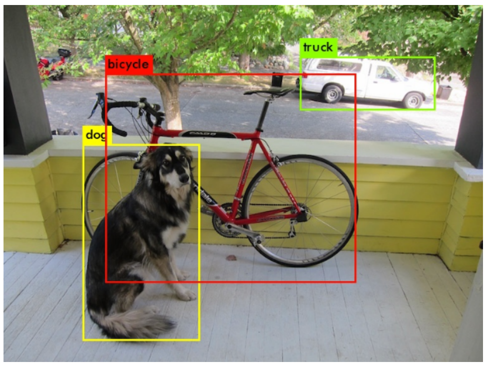


Figure 2.15: The general object detection problem. Pre-defined categories are localized and classified using bounding boxes. Source: https://pjreddie.com/darknet/yolo/ (retrieved 2020)

2.10.2 Modern object detector anatomy

A modern object detector based on deep learning includes three main parts, a backbone, neck, and head, as shown in the list below, retrieved from [7].

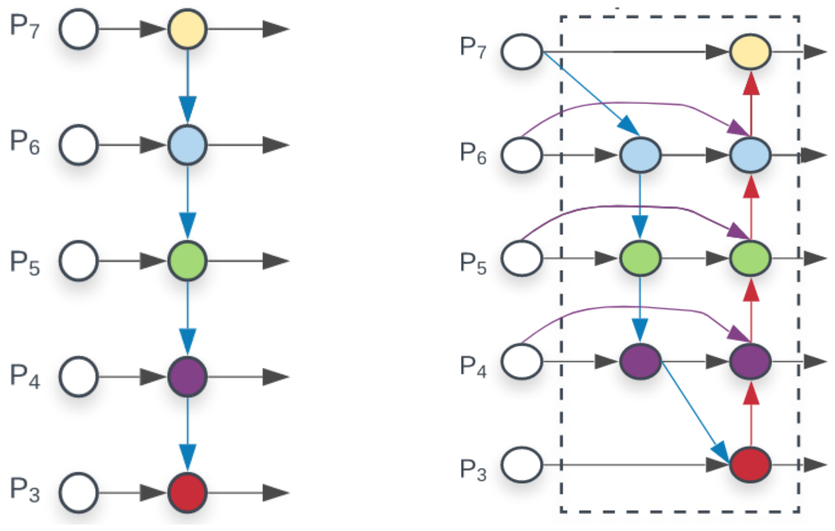
- **Input**: Image, patches, Image Pyramid
- **Backbones**: VGG16, ResNet-50, SpineNet, EfficientNet-B0/B7, CSPResNeXt50, CSPDarkNet53
- **Neck**:
  - **Additional blocks:** SPP, ASPP, RFB, SAM
  - **Path-aggregation blocks**: FPN, PAN, NAS-FPN, Fully-connected FPN, BiFPN, ASFF, SFAM
- **Heads**:
  - **Dense prediction (one-stage)**: RPN, SDD, YOLO, RetinaNet, CornetNet, CenterNet, MatrixNet, FCOS
  - **Spare prediction (two-stage)**: Faster R-CNN, R-FCN, Mask R-CNN, RepPoints

Backbone

The backbone is a pre-trained network that extracts a general feature map from the input image. These networks come in different sizes and differ in speed and accuracy. VGG16 is a common backbone consisting of 16 layers, where 13 are convolutional, and three are fully connected. This network has much fewer parameters than ResNet-50, which has 50 layers, resulting in higher speed at the cost of lower accuracy. EfficientNet is a family of backbones addressing this trade-off between speed and accuracy. The B0 model serves as a baseline model with the least number of parameters and is thus the fastest in the family. Succeeding models from B1 to B7 are scaled using a single parameter ϕ, resulting in models of increasing size.

Neck

The neck includes layers between the backbone and head to collect feature maps at different scales. This enables detections across scales and thus improves the performance in detecting objects of different scales. Two standard methods to achieve this is a Feature Pyramid Network (FPN) or a Bi-directional Feature Pyramid Network (BiFPN). FPN fuses features in a top-down manner, as shown in fig. 2.16a. This is achieved by resizing the lower resolution feature map and conducting a convolution between each level's two feature maps. BiFPN, as shown in fig. 2.16b is a top-down and bottom-up approach that removes nodes with one input edge. Additional edges are inserted between the input and output node of each level, and finally, a bottom-up edge is added at the output node. This BiFPN neck achieves approximately 2% higher AP score on the COCO test set [81] and uses 12% fewer parameters than the standard FPN method.



(a) FPN (b) BiFPN

Figure 2.16: FPN and BiFPN collect feature maps at different scales. P3 to P7 are feature maps down-sampled from the feature map provided by the backbone. The edges show the feature maps fused at the nodes. Source: [81] (retrieved 2020)

Head

The head is the final part of the object detector, which predicts bounding boxes and the corresponding classes. Two-stage detection frameworks first locate category-independent regions of interest. Features are then extracted from these regions and sent to a category-dependent classifier to classify the proposed regions. One-stage detectors consist of architectures that predict bounding boxes and corresponding class labels in a single feed-forward DCNN. These methods jointly learn the important features for predicting bounding boxes and corresponding class labels, which removes the need for separate processing stages.

2.10.3 Evaluation metrics

The most popular metric for measuring an object detection model's accuracy is the AP metric or mean average precision (mAP) for multi-class detection models. It is customary not to distinguish between these two in the literature and assume the difference is clear from the context. A precision-recall curve is a graphical alternative for visualizing the model's accuracy. To define these two concepts of AP and precision-recall curve, we start with defining true positives (TP), false positives (FP), and false negatives (FN). True positives are the number of correctly detected objects in an image. On the other hand, false positives are the number of wrongly detected objects in an image. False negatives are the number of objects in the image not detected. The precision and recall metrics are then defined as shown in eq. (2.40) and eq. (2.41) respectively.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.40}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.41}$$

To precisely define these metrics, the metric intersection over union (IoU), which measures the overlap between the predicted- and the ground truth bounding box, is defined as seen in eq. (2.42).

$$\text{IoU} = \frac{B_p \cap B_{gt}}{B_p \cup B_{gt}} = \frac{\text{area of intersection}}{\text{area of union}} \tag{2.42}$$

IoU can then be set as a threshold to determine when we want to consider a detection as a TP, where typical values for IoU ranges from 0.50 to 0.95. A precision-recall curve can then be created by calculating the precision-recall pairs for each image in the test set using the object detection model.

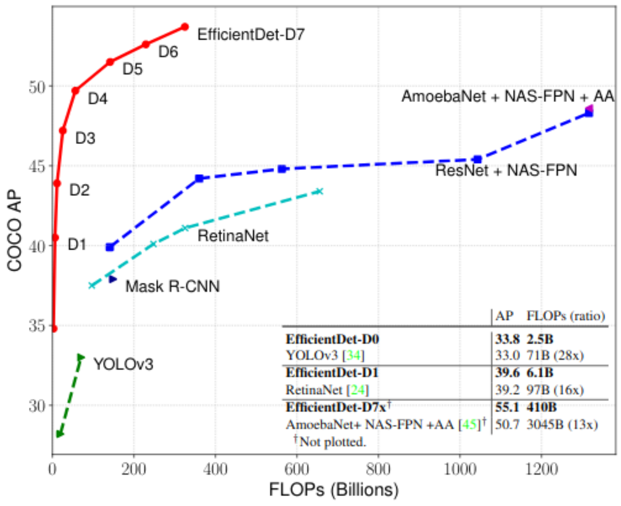
The area under the precision-recall curve is then defined as the average precision, taking values from 0 to 1. A simple method for calculating this area is using interpolation, where the precision values are sampled for N recall values. The AP can then be calculated by summing these interpolated values and taking the average. There exist several other methods for calculating this metric. However, the latest research provides their results in the COCO mAP metric only [7][81]. The COCO AP metric interpolates the precision-recall curve with 101 points for IoU values between 0.50 and 0.95 [51]. Other commonly used COCO metrics related to different IoU levels are AP_{50} and AP_{75} which uses IoU=0.5 and IoU=0.75 respectively.

To provide a metric for how the detector performs on objects of different scales, AP_{small}, $\text{AP}_{\text{medium}}$ and AP_{large} measures the COCO AP for objects of different pixel sizes in the image. AP_{small} is used for objects with size $< 32^2$ pixels and AP_{large} is used for objects of size larger than $> 96^2$ pixels. The objects which fall between these two sizes will be evaluated in $\text{AP}_{\text{medium}}$.

Average recall (AR) is computed using the same method as AP but varies the number of detections to consider in each image. There are three variants used in the COCO evaluation metrics, AR_1, AR_{10} and AR_{100} for 1, 10 and 100 detections respectively.

2.10.4 EfficientDet detector

The EfficientDet object detector introduced in section 1.2, is according to [81] the current SOTA object detector. A comparison to other object detectors is shown in fig. 2.17. The figure shows that this family of object detectors outperforms other object detectors in precision and speed. This family of object detectors scales the entire network using a compound scaling approach through a single parameter ϕ. This results in the EfficientDet models D0-D7x in increasing size. To fuse feature maps at different scales, a novel BiFPN is utilized in combination with weights to learn the importance of each scale during training.



	AP	FLOPs (ratio)
EfficientDet-D0	**33.8**	**2.5B**
YOLOv3 [34]	33.0	71B (28x)
EfficientDet-D1	**39.6**	**6.1B**
RetinaNet [24]	39.2	97B (16x)
EfficientDet-D7x†	**55.1**	**410B**
AmoebaNet+ NAS-FPN +AA [45]†	50.7	3045B (13x)

†Not plotted.

Figure 2.17: EfficientDet models D0-D7x benchmarked on the COCO test set compared to other detectors. Source: [81] (retrieved 2020)

Architecture

The architecture of the EfficientDet model is shown in fig. 2.18. EfficientNet is used as the backbone for feature extraction. The feature map is then down-scaled seven times by successive halving, resulting in feature maps from P1 to P7 in decreasing resolution. Feature maps P3, P4, P5, P6, and P7 are then passed to the neck consisting of multiple BiFPN layers before a single stage Single Shot multibox Detector head predicts bounding boxes and corresponding classes.

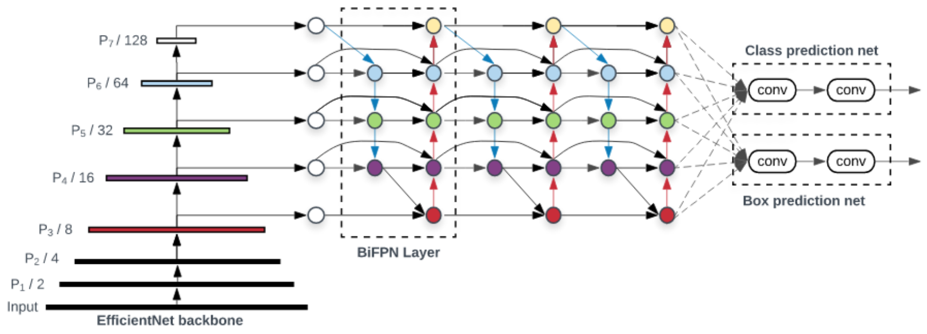


Figure 2.18: EfficientDet network architecture. Source: [81] (retrieved 2020)

Multi scale feature fusion

EfficientDet utilizes BiFPN in combination with fast normalized fusion to add weights to the scales to achieve an efficient multi-scale feature fusion in the neck. Adding weights to the fusion lets the network learn the importance of the different scales. The output from a node O given input edges I_i is then given as seen in section 2.10.4.

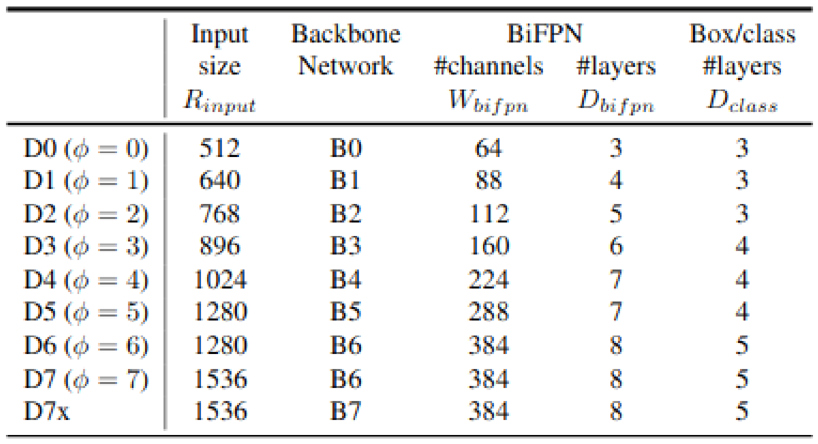
$$O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$$

The parameter $\epsilon = 0.0001$ is added to avoid numerical instability, while Relu is applied after each weight to ensure positive weights.

Compound scaling

A single parameter ϕ is proposed to uniformly scale the input image resolution, backbone, neck, and head. The resulting family of object detectors ranging from D0 to D7x is shown in table 2.1.

Table 2.1: Compound scaling of EfficientDet using a single parameter $\phi \in [0, 7]$, yielding the models in increasing size from D0 to D7x. Source: [81] (retrieved 2020)



	Input size	Backbone Network	BiFPN		Box/class
			#channels	#layers	#layers
	R_{input}		W_{bifpn}	D_{bifpn}	D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

2.11 Multiple object tracking

In this section, the Multiple Object Tracking (MOT) problem will be introduced, then the two main schemes and processing modes investigated. This is followed by an introduction of MOT evaluation metrics and the Simple Online Realtime Tracker (SORT).

2.11.1 Problem formulation

The MOT problem has been formulated differently in previous work in the field. In [58] the authors propose the general formulation as follows:

The objective of MOT is to find the "optimal" sequential states of all the objects, which can be generally modeled by performing Maximum a Posteriori estimation from the conditional distribution of the sequential states given the observations.

This formulation entails solving eq. (2.43) using different approaches, either from a probabilistic inference- or deterministic optimization perspective. In this equation, $\mathbf{S}_{1:t}$ denotes the states of all objects from time $t = 1$ to $t = t$, while $\mathbf{O}_{1:t}$ denotes the observations.

$$\hat{\mathbf{S}}_{1:t} = \underset{\mathbf{S}_{1:t}}{\arg\max}\, P(\mathbf{S}_{1:t}|\mathbf{O}_{1:t}) \tag{2.43}$$

2.11.2 The two main schemes

There are generally two main schemes in MOT, Detection Based Tracking (DBT) or Detection Free Tracking (DFT). These two schemes are illustrated in fig. 2.19, where the main difference between these two schemes is how tracks are initiated. The former scheme-tracks are initiated by detections from an object detector such as described in section 2.10.4 while the latter required manual initialization of a fixed number of tracks in the first frame.

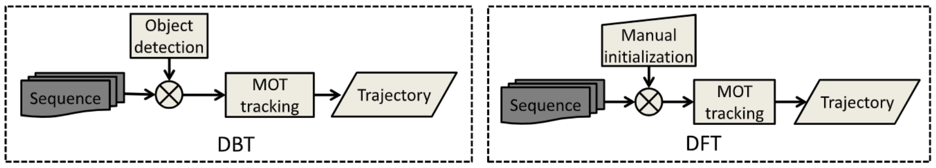


Figure 2.19: The two prominent MOT schemes with Detection Based Tracking to the left, and Detection Based Tracking to the right. Source: [58] (retrieved 2021)

Processing mode

Additionally, it is customary to separate MOT methods into online- or offline methods. The former is also known as sequential tracking, where tracks are updated and produced on the fly as new frames arrive. The latter generally has all frames available and will exploit this to process the frames jointly in batches to estimate the final output. Hence, online tracking methods only utilize past information up to the current frame, while offline methods employ observations in the past and the future.

In fig. 2.20, both tracking modes are illustrated with online and offline methods in the upper and bottom portion of the figure, respectively. The tracked objects a, b, and c are illustrated with blue circles, while the green arrows represent observations in the past. For offline tracking, the problem can be split in to batches due to computational limitations.

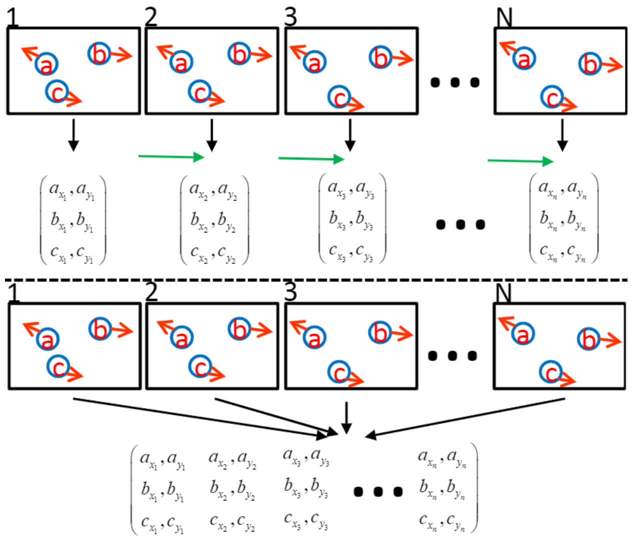


Figure 2.20: The two MOT processing modes, with online methods on the top, and offline methods on the bottom. Source: [58] (retrieved 2021)

2.11.3 Evaluation Metrics

As in object detection, there are many different metrics for measuring the performance of a tracker. The most widely accepted metric is the Multiple Object Tracking Accuracy (MOTA) metric which attempts to measure the tracker's accuracy. Two other vital metrics are Multiple Object Tracking Precision (MOTP) and Identity Switches (IDs). The former measures how precisely objects are tracked utilizing bounding box IoU, and the latter counts the number of times a track switches identity. The quality of tracks can be classified as Mostly Tracked (MT), Partially Tracked (PT), and Mostly Lost (ML).

MOTA

The MOTA metric can be calculated as seen in eq. (2.44), where t is the frame index. The terms FN_t, FP_t, IDs_t and GT_t are the false negatives, false positives, identity switches, and number of ground truths for frame t respectively.

$$\mathrm{MOTA} = 1 - \frac{\sum_t (\mathrm{FN}_t + \mathrm{FP}_t + \mathrm{IDs}_t)}{\sum_t \mathrm{GT}_t} \tag{2.44}$$

MOTP

The MOTP metric can be calculated as seen in eq. (2.45), where t is the frame index and i is the track number. The terms c_t, and $\mathrm{d}_{t,i}$ are the number of matches in frame t, and the IoU of target i for the assigned ground truth. This metric formally calculates the average dissimilarity between true positives and their corresponding ground truths. Simply put, it measures the localization accuracy of the tracker.

$$\mathrm{MOTP} = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \tag{2.45}$$

MT, PT, ML

A target is classified as MT if it is successfully tracked in over 80% of its lifespan, ML if less than 20%, and PT otherwise. These metrics do not account for identity switches but rather how much of a target's trajectory is recovered.

IDs

The IDs metric measures the number of times a reported identity of a ground truth track changes identity. The metric is incremented if a ground truth target i has formerly been associated to track j, while the most recent assignment to i is $k \neq j$.

2.11.4 Simple Online Realtime Tracking

SORT is a simple online DBT approach utilizing only the rudimental technique of Kalman Filtering for estimation and the Hungarian algorithm for assignment of detections to tracks. At the time of the publication of this paper in 2017, the SORT algorithm achieved SOTA results on the MOT2015 [48] benchmarking data set. Despite being outcompeted by newer algorithms utilizing techniques of deep learning, SORT is still ranked 31 on the MOT2020 challenge[1]. For the remainder of this subsection, the model estimation using

[1] https://motchallenge.net/results/MOT20/

Kalman Filter (KF), the data association using the Hungarian Algorithm and the creation and deletion of tracks is presented.

The Kalman Filter

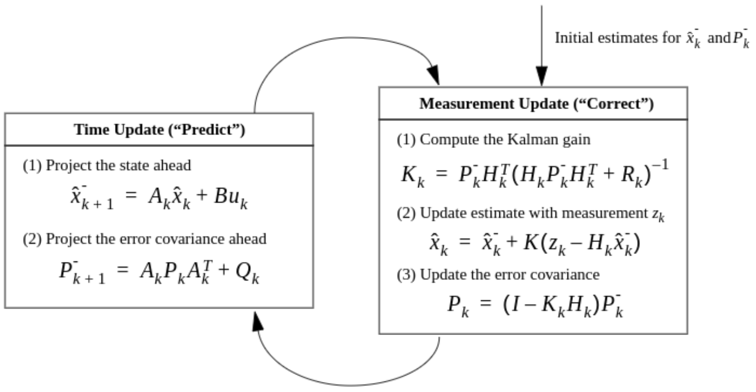
For model estimation, SORT utilizes a discrete-time KF which solves the MAP problem in eq. (2.43) in a probabilistic inference manner using a two-step iterative process, consisting of a prediction step, followed by an update step. The filtering problem can be modeled as seen in eq. (2.46) following the conventions of [87]. In this system, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{H}$ denotes the transition-, input and observation model, respectively. The first carries the dynamics of the state variable $\mathbf{x}_k$ while the second defines how the input $\mathbf{u}_k$ influences the states, and the third defines which variables are observed.

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{z}_k &= \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{2.46}$$

The random variables $\mathbf{w}_k$ and $\mathbf{v}_k$ represent the process- and measurement noise, respectively. These are assumed to be uncorrelated, zero-meaned, and normally distributed as seen in eq. (2.47).

$$\begin{aligned} \mathbf{w}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{Q}) \\ \mathbf{v}_k &\sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \end{aligned} \tag{2.47}$$

Using this developed model, we can apply the general two-step iterative KF correction and prediction scheme as seen in fig. 2.21 to estimate $\hat{\mathbf{X}}_k$ at the current time $t = k$.



Initial estimates for $\hat{x}_k^-$ and P_k^-
Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_{k+1}^- = A_k\hat{x}_k + Bu_k$
(2) Project the error covariance ahead
$P_{k+1}^- = A_kP_kA_k^T + Q_k$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^-H_k^T(H_kP_k^-H_k^T + R_k)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k\hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_kH_k)P_k^-$

Figure 2.21: The general KF scheme: 1. The Kalman filter gain $\mathbf{K}_k$ is computed and used to update the current estimate of $\mathbf{X}_k$ and the error covariance matrix $\mathbf{P}_k$. 2. The state estimated $\hat{\mathbf{X}}_k$ is projected ahead using the state dynamics. Finally, the error covariance is projected ahead. Source: [87] (retrieved 2021)

Bounding box prediction

The KF in SORT is used to predict bounding boxes from one frame to the next frame utilizing the general linear constant velocity model $x = x_0 + v\Delta t$ for the states of a bounding box as seen in eq. (2.48). The states u_k, v_k are the coordinates of the center, s_k the area, and r_k the aspect ratio of the bounding box.

$$\mathbf{x}_k = \begin{bmatrix} u_k & v_k & s_k & r_k & \dot{u}_k & \dot{v}_k & \dot{s}_k \end{bmatrix} \tag{2.48}$$

The linear constant velocity model yields the transition model **F** while the bounding box detections yield the observation model as both seen in eq. (2.49). The input $\mathbf{u}_k$ to the system is omitted.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.49}$$

The covariance matrices **Q** and **R** are chosen as the identity matrix, where higher uncertainty is put on the states $s_k, r_k, \dot{u}_k, \dot{v}_k$ and $\dot{s}_k$ by a factor of 10, 10, 10, 10 and 100 respectively. For the process error covariance matrix **P**, the matrix is initialized as identify, with an increased uncertainty by a factor of 1000 for the initial velocities $\dot{u}_k, \dot{v}_k$ and $\dot{s}_k$.

The Hungarian method

The Hungarian method as introduced in [47] tries to solve the following assignment problem:

Assuming that numerical scores are available for the performance of each n persons of n jobs, the "assignment problem" is the quest for an assignment of persons to jobs so that the sum of the n scores so obtained is as large as possible.

A naive approach to solve this problem is to compute the $n!$ scores, but this is very inefficient for large-scale problems of large n. The Hungarian method solves this problem in polynomial time. This is achieved by first creating a qualification matrix **Q** where each entry (i,j) in the matrix corresponds to the cost of assigning a person in row i to the job at column

j. Then through sequential steps working on this matrix, it can find the optimal assignment.

Assigning detections to tracks

To assign detections to existing tracks, a qualification matrix **Q** is computed as the IoU between all the detections and all predicted bounding boxes for each track. This is solved by utilizing the Hungarian Method where assignments with IoU less than IOU_{min} are rejected. Assigned detections are treated as updates through the observation model in the KF of the assigned tracks.

Creation and deletion of tracks

Detections with an IoU to existing tracks less than IOU_{min} are assigned to new tracks where the initial velocity of the track is set to zero. If the new track has not been associated with a new detection within T_{min} frames, the track is terminated to prevent tracking of false positives. Additionally, tracks are terminated if no detection is associated with the track within T_{lost} frames.

2.12 Fiducial detection

Fiducials are artificially created landmarks designed to be easily recognizable and distinguishable from another. By encoding information in a bit-like fashion as seen in fig. 2.22, vision algorithms can extract useful information such as the identity and the pose of the tag relative to the camera. A popular algorithm for the detection of AprilTag markers is the AprilTag detector.

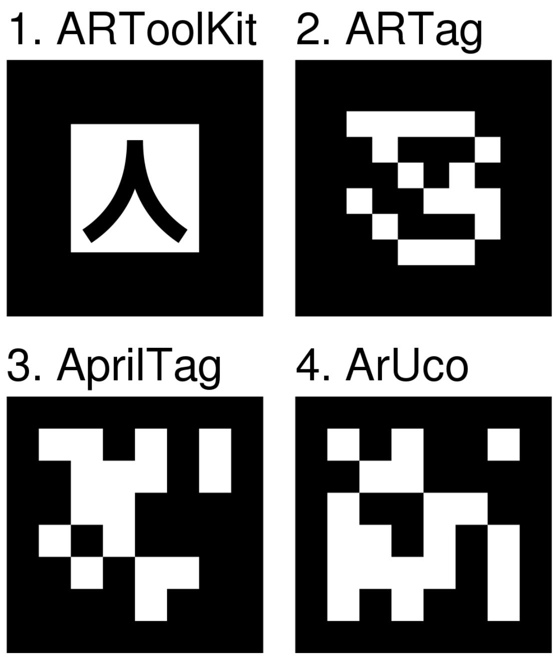


Figure 2.22: Four common fiducial markers. The AprilTag marker is used in the AprilTag detection system. Source: https://en.wikipedia.org/wiki/Fiducial_marker (retrieved 2021)

2.12.1 The AprilTag detector

The AprilTag detector is composed of several phases to localize and identify AprilTags in an image. The first phase is to construct line segments based on the gradient calculation at each pixel. These gradients are clustered based on the gradient magnitude and direction and then used in a regular least squares approach to fit connected components into line segments. Once the line segments have been constructed, the next phase is to join line segments into four-sided quads. This is achieved by a recursive depth-first-search with a depth of four. Each level in the search tree adds a line, where all lines are considered for the first depth entry. For the remaining depths, only *close enough* line segments to where the last segment ended are considered, obeying a counter-clockwise order. Once four connected lines are found, a quad candidate has been detected [65].

The third phase involves the estimation of the tag pose relative to the camera. First, the homography, which estimates the projection of points in the tag's coordinate system to the camera coordinate system, is estimated utilizing DLT. This homography matrix **H** can be written as the product of an unknown scale factor s, the camera matrix **P**, and the truncated extrinsic matrix **E** as seen in eq. (2.50). This truncated extrinsic matrix results from setting $z = 0$ for the planar surface in the tag's coordinate system. In order to estimate the homography, the physical tag size is required.

$$\mathbf{H} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} = s \underbrace{\begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{bmatrix} R_{00} & R_{01} & T_x \\ R_{10} & R_{11} & T_y \\ R_{20} & R_{21} & T_z \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{E}} \tag{2.50}$$

The entries of **H** can be used to solve for R_{ij} and T_x, T_y, T_z as seen in eq. (2.51), as the scale factor can be recovered by utilizing that the columns of the rotation matrix must have unit magnitude. First, the magnitude of s is calculated as the geometric average of the first two columns. The sign is determined by requiring that the tag should be in front of the camera, such that $T_z \geq 0$. The third and final column of the rotation matrix is determined by the cross product of the two first columns as these must be orthonormal.

$$\begin{aligned} h_{00} &= sR_{00}f_x \\ h_{01} &= sR_{01}f_x \\ h_{02} &= sT_xf_x \\ &\vdots \end{aligned} \tag{2.51}$$

The final phase is to determine the tag's identity by a payload decoding scheme. This is achieved by computation of the tag-relative coordinates for each bit in the tag, and then they are transformed to the image coordinate system utilizing the estimated homography. The resulting pixels are thresholded using two spatially-varying models for black and white pixels. These models are learned from the border of the tag, which contains known examples of white and black pixels. The resulting thresholded image bits can then be used to determine the tag's identity by the use of a modified lexicode. This lexicode is parameterized by the number of n bits in the code-words and the minimum distance d between any two words. The commonly used 36h10 tag has a 36-bit encoding with a minimum Hamming distance of 10.

Chapter 3

Methods

This chapter aims to establish the methodical approach used to obtain the results. Section 3.1 describes the container asset and assumptions made regarding the asset. The robotic Spot platform utilized in this work is briefly introduced in section 3.2, followed by section 3.3 which presents the gathered data set and the labeling process. Finally, the proposed novel visual asset tracking pipeline is introduced in section 3.4.

3.1 Asset description

Following the preliminary work in the project thesis, the asset of interest in this work is shipping containers. Two examples of such assets are showcased in fig. 3.1, where important attributes of the asset are presented below.

Geometrical shape

The geometrical shape of the container is a cuboid with dimensions width $\times$ height $\times$ length. These dimensions differ as there are a variety of different standards used. The most common dimensions used are a width of approximately 2.44 m, height of either 2.59 m or 2.90 m, and lengths typically ranging from 6.10 m to 12.19 m.

Color

Shipping containers come in a wide variety of colors, and no assumption is thus made on the color of the containers.

Static

All shipping containers are assumed to be static during the pipeline processing. This minimizes the effect of motion blur and simplifies the analysis.

Occlusions

A common problem at industrial sites such as Kvaerner Stord is the presence of object occlusions. An example of such occlusion is shown in the image below, where a red box partially occludes the blue container. Different and more severe occlusions are also typical, and thus there will be no assumptions made regarding occlusions in this work.

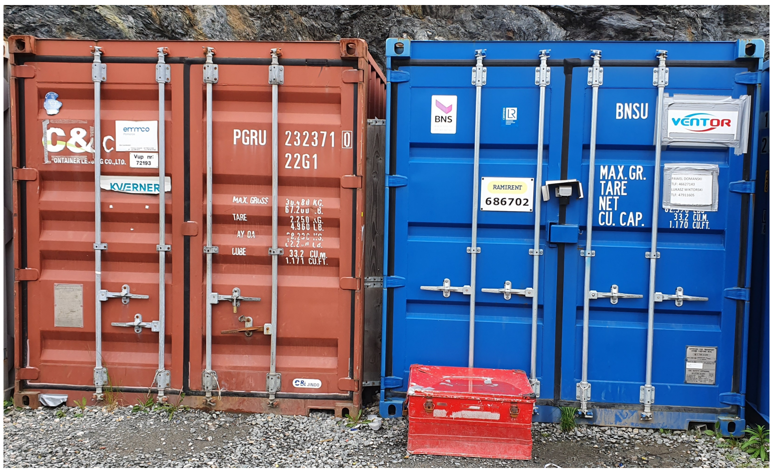


Figure 3.1: Two shipping containers found at Kvaerner Stord

3.2 The Spot platform

The robotic platform used in this work is the four-legged Spot robot from Boston Dynamics[1]. The robotic platform is ready to operate, right out of the box, where the platform can be extended by utilizing payloads. In fig. 3.2, the Spot platform used in this work is showcased with the Spot Cam+, and Spot CORE payloads equipped.

3.2.1 Spot CAM+

The SpotCAM+ payload, as seen in fig. 3.3 features a Pan-Tilt-Zoom (PTZ) camera on the top and a 360 ring camera on the bottom, where the latter is composed of five RGB fisheye cameras. The 360 ring camera is the primary sensor used, while the PTZ camera is not used in this work. Each camera in the ring has a resolution of 1080×1920.

3.2.2 Spot CORE

The Spot CORE payload provides developers with an additional processor to run code locally on the robot. This payload comes with an Ubuntu 18.04

[1]https://www.bostondynamics.com/products/spot

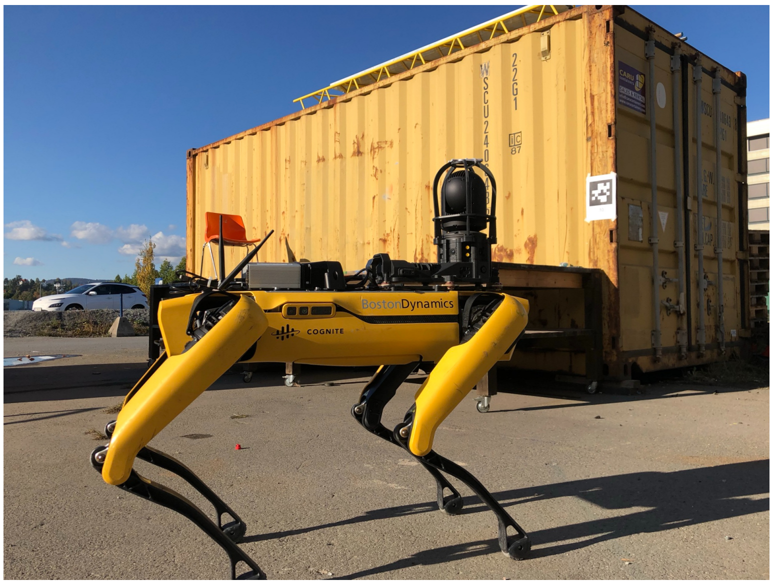


Figure 3.2: The Spot robotic platform was used in this work. The SpotCAM+ payload on the front and the SpotCORE on the robot's rear.

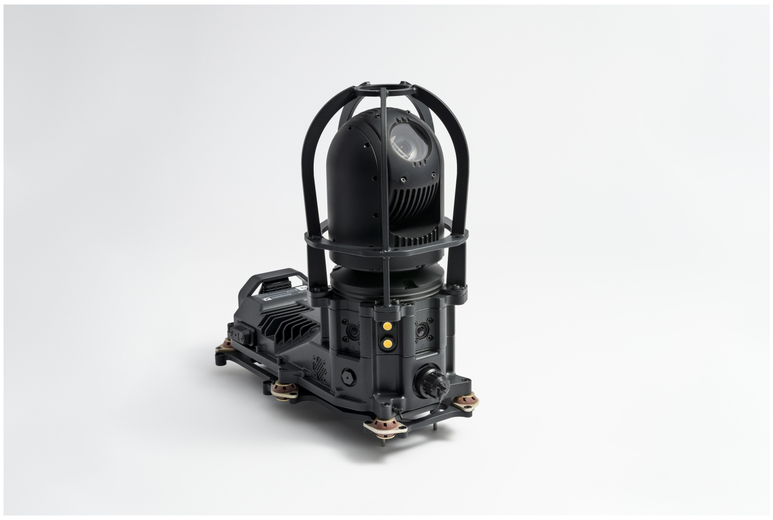


Figure 3.3: The SpotCAM+ payload features a PTZ camera on the top and a set of 5 fisheye ring cameras on the bottom. Source: https://shop.bostondynamics.com/spot-cam-ptz?cclcl=en_US (retrieved 2021)

operating system and is connected to the internals of the robot by Ethernet. Programming the robot is achieved by using the Spot Software Development Kit (SDK)[2].

[2]https://github.com/boston-dynamics/spot-sdk

Figure 3.4: The SpotCORE payload features an additional processor which comes with an Ubuntu 18.04 OS. Source: https://shop.bostondynamics.com/spot-core?cclcl=en_US (retrieved 2021)

3.2.3 Spot missions

An essential concept when using Spot is the recording and execution of missions. The recording of a mission is achieved by manually controlling the robot using the Spot Tablet through the desired route while adding actions along the way. Examples of such actions are gauge readings or 360 scans. In the end, the mission can be saved to execute this specific mission at a later time. Throughout the mission, the robot navigates the scene by utilizing visual information and robot kinematics. An AprilTag at the start location is required where additional tags are recommended for better visual navigation within the estimated map of the scene. Unfortunately, there is limited information on the inner workings of the navigation system, but this is at least a brief summary of what the documentation reveals.

3.2.4 Spot frames

Spot has two inertial frames, *odom* and *vision*. The former is a fixed location in the world that is determined by using the kinematics of the robot. The latter also utilizes visual analysis of the world to determine this fixed point. The *visual* inertial frame will be used in this work as it provides a *"better"* and more accurate navigation in the world, according to Boston Dynamics.

Figure 3.5 shows the configuration of the Spot *body* frame. This frame follows the robot as it is fixed in the robot's center of gravity. The internal state estimation of the robot estimates the relative transform $\mathbf{T}_{inertial,body}$ which we will refer to as the pose of the robot.

In order to triangulate points in the inertial frame, the relative transforms $\mathbf{T}_{body,c0}$, .., and $\mathbf{T}_{body,c4}$ are required. These are acquired through the Spot SDK and visualized in fig. 3.6.

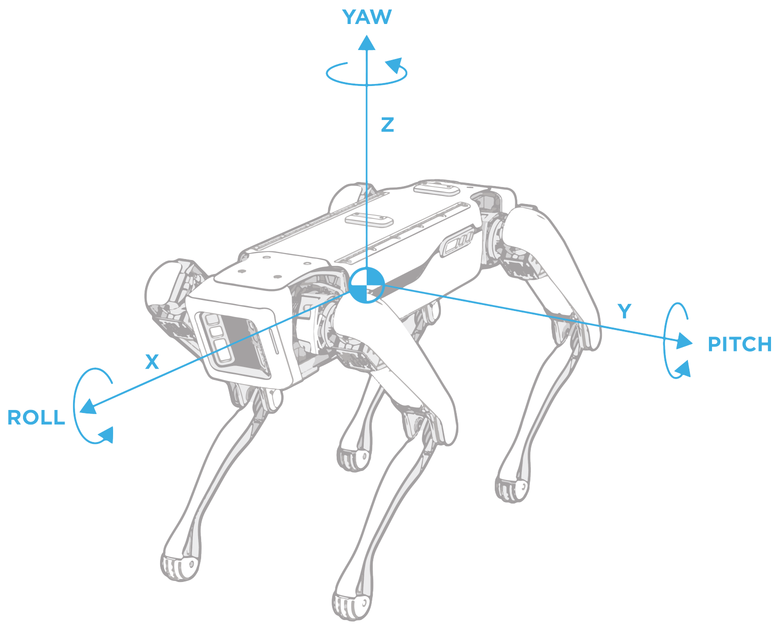


Figure 3.5: The Spot body frame. Source: https://dev.bostondynamics.com/docs/concepts/geometry_and_frames (retrieved 2021)

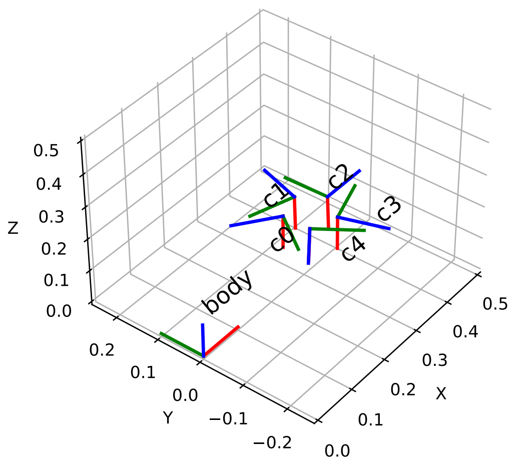


Figure 3.6: The configuration of the five ring camera frames c0, c1, c2, c3, and c4 relative to the Spot body frame.

3.3 The data set

In this section, the developed data set in this work is introduced. This data set is composed of two parts, the first being the images used for the development of the container detector, while the second consists of two videos which is used for testing and evaluation of the asset tracking pipeline.

3.3.1 Data gathering

The location of the data gathering in this work was at the H3 arena in Fornebu. This industrial location was chosen as it was close to the Cognite offices and had several containers in the area. In total, two gatherings were performed. The first in sunny conditions and the last in overcast conditions. A simple route mimicking what an autonomous inspection might look like was constructed. This route contained no loops and as little rotational movement as possible, making the robot primarily move in forward motion.

First, AprilTags were set up both along the chosen route and on containers. The non-container tags are used to aid the robot navigation, where the tags on the containers will also aid the robot, but more importantly, function as ground truths for the Identification (ID) and location of the containers in the analysis. The containers which were mounted with tags were the first blue (B1), the first red (R1), the second blue (B2), and the final yellow container (Y1). The gray (G1) container was not equipped with a tag. The respective AprilTag IDs for sunny conditions were 1, 2, 8, and 10, and for overcast conditions 1, 3, 4, and 18.

Then a mission without actions was recorded by manually walking the robot through the route using the Spot Tablet. A new mission had to be created for the overcast conditions, as the AprilTags had to be removed between the gatherings. As the mission was replayed, a custom-made data gathering script running on the Spot CORE logged and synchronized the 360 images, and the robot poses at a maximum frame rate of 1.7 FPS. The resulting robot trajectory for the sunny weather video is showcased in blue in fig. 3.7, where the robot starts at the orange triangle. The irregular gaps between the trajectory data points are due to the failure to acquire log-points from the SpotCAM+. This logging of robot poses and log-points from the camera and the acquisition of the intrinsic parameters of the ring cameras was achieved using the Spot SDK.

Figure 3.8 and fig. 3.9 show five frames from each gathered video for sunny and overcast weather, respectively, where the B1, R1, G1, B2, and Y1 container is shown. Each row in the images corresponds to frame 0, 10, 20, 30, and 40. The recorded robot trajectory is visualized as red circles by projecting the trajectory onto $c2$.

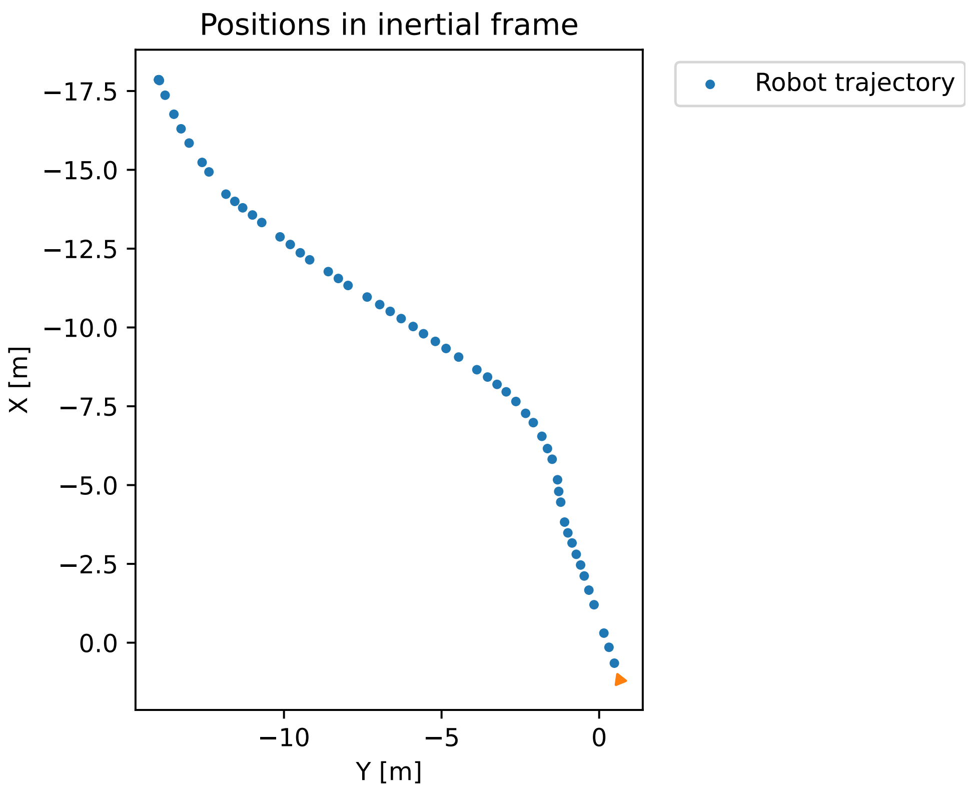


Figure 3.7: The robot trajectory in sunny conditions. The robot starts at the orange triangle and continues on the blue trajectory. Notice the irregular gaps between the data points.

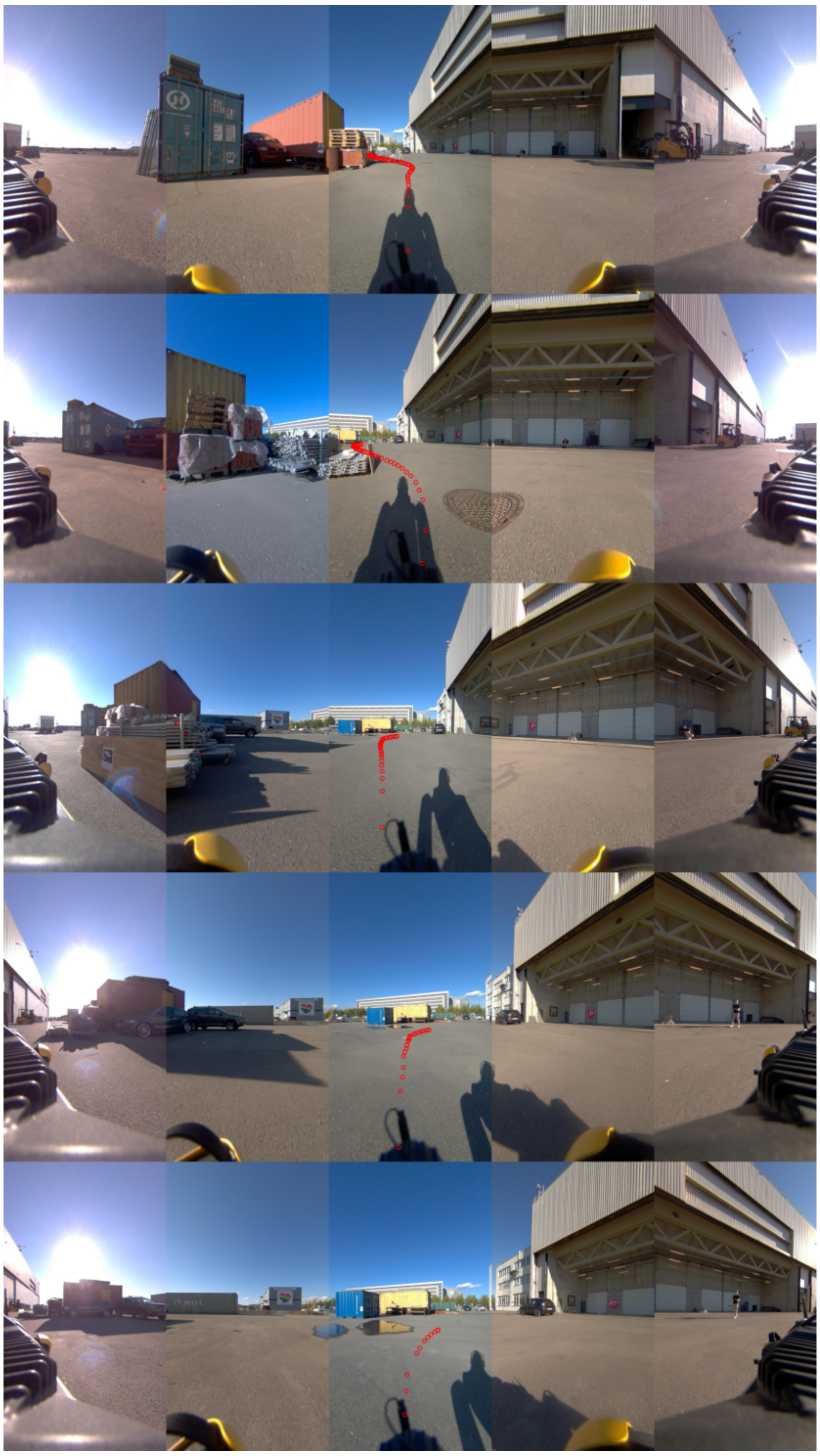


Figure 3.8: Frame 0, 10, 20, 30, and 40 of the sunny condition's video. The red circles showcase the robot's trajectory. The B1 and R1 containers are visible in the first frame, while G1, B2, and Y1 are visible in the fourth frame.

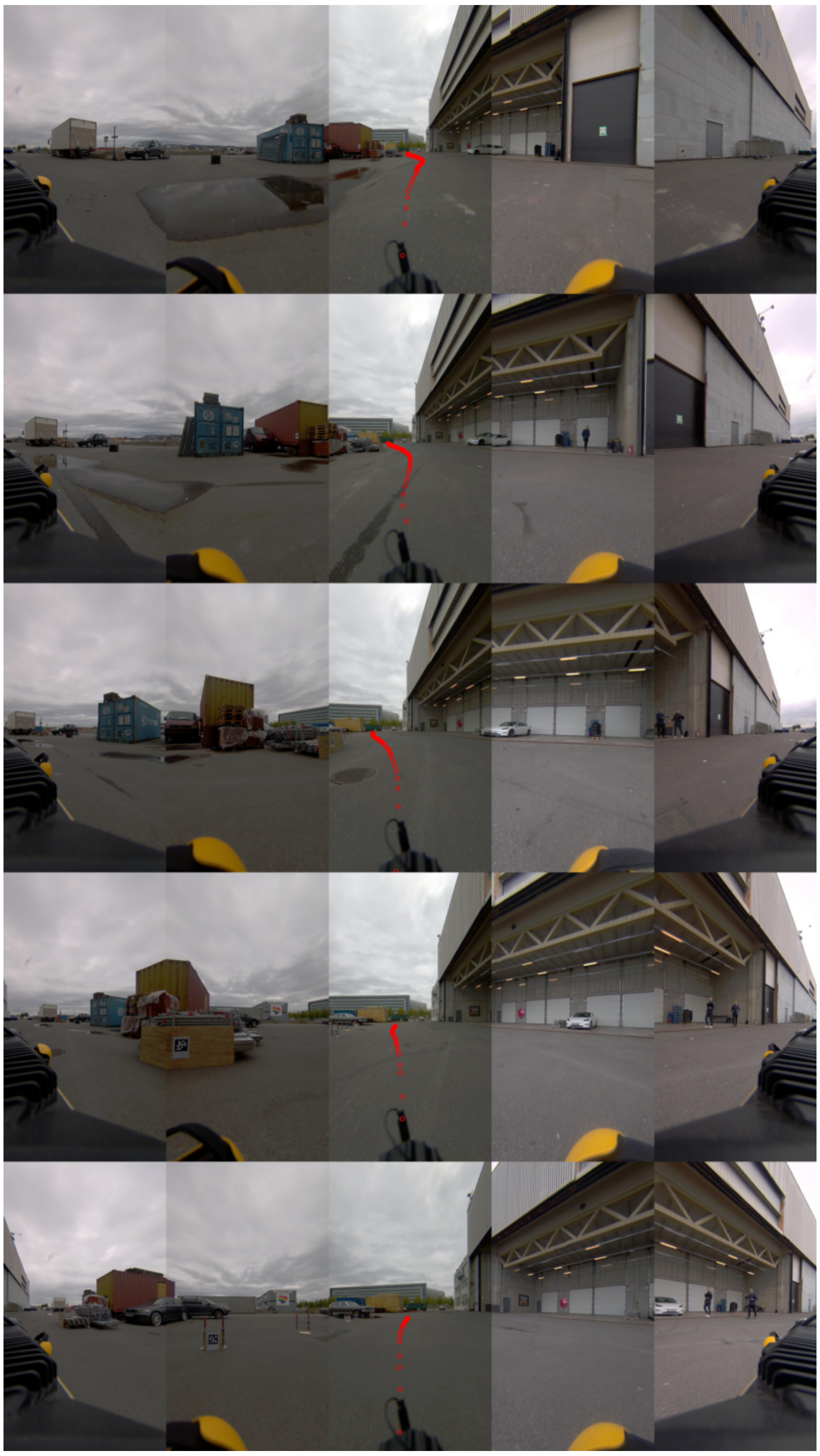


Figure 3.9: Frame 0, 10, 20, 30, and 40 of the overcast condition's video. The red circles showcase the robot's trajectory. The B1 and R1 containers are visible in the first frame, while G1, B2, and Y1 are visible in the fifth frame.

3.3.2 Data set annotation

During the data gathering process, in total, nine videos were recorded. Two of those, one for each weather condition, have been chosen to test and validate the visual asset tracking pipeline. The remaining seven videos have been sampled for images to extend the container detection data set. This was done as the detector developed in the project thesis did not generalize to the new image data. This might be due to the aspect ratio of the iPhone X images being 4:3, while the aspect ratio of the ring cameras is 9:16. This is a common problem in object detection, where models tend to generalize poorly to different aspect ratios.

The annotation tool chosen for labeling the images and videos in this work is the computer vision annotation tool[3]. This was found to be a satisfactory annotation tool in the project thesis, supporting both the PASCAL VOC and MOT[4] format for the images and videos, respectively.

The VOC annotation guidelines were used to label both the images and videos. These guidelines are summarized below:

- *What to label:* All objects of the predefined categories, unless:
  - you are not sure what the object is
  - the object is very *small*
  - less than 20% of the object is observable, such that it is not obvious what the object is
- *Bounding box:* Draw a bounding box around the visible area of the object, not the estimated area.
- *Occlusion:* If more than 5% of the area of the object is covered by another object, mark it as occluded.

Images

The images are used to train, evaluate, and test the container detector. A split of $80-10-10$ has been chosen for the training, evaluation, and test set, respectively, as this was found to be a commonly used split in the project thesis. The seven videos that were not used for testing and evaluation of the pipeline were sampled to extend this data set. This was achieved by sampling every tenth frame in each video, where images without any containers were discarded. This resulted in an additional 187 images labeled in PASCAL VOC format.

[3]https://github.com/openvinotoolkit/cvat
[4]https://motchallenge.net/instructions/

Videos

The two chosen videos are not sampled for images and are solely used to test and evaluate the asset tracking pipeline. The video in sunny conditions is composed of 59 frames, while the video in overcast conditions has 61 frames, where five frames from each video are showcased in fig. 3.8 and fig. 3.9. This results in a total of 295 and 305 images for the respective conditions, which have been labeled according to the MOT format.

3.3.3 Summary

The resulting data set can be summarized as follows:

Images

The detection data is labeled in PASCAL VOC format, and a split of $80-10-10$ has been chosen for the train, evaluation, and test set, respectively.

- *Train set size*: 466
  - 145 ring camera images
  - 321 iPhone X images
- *Evaluation set size*: 59
  - 22 ring camera images
  - 37 iPhone X images
- *Test set size*: 59
  - 20 ring camera images
  - 39 iPhone X images

Videos

The two videos have been labeled in MOT format and are used solely to test and evaluate the asset tracking pipeline. Both videos are synchronized with the recorded robot poses.

- Sunny conditions
  - *weather:* Sunny
  - *Number of frames:* 59
  - *Number of robot poses:* 59
  - *Number of images:* 295
  - *AprilTags IDs visible:* 1, 2, 4, 6, 8, 10

  - *AprilTags IDs on containers:* 1, 2, 8, 10
  - *Ring camera intrinsic parameters*

- Overcast conditions
  - *weather:* Overcast
  - *Number of frames:* 61
  - *Number of robot poses:* 61
  - *Number of images:* 305
  - *AprilTags IDs visible:* 1, 3, 4, 10, 14, 16, 18
  - *AprilTags IDs on containers:* 1, 3, 4, 18
  - *Ring camera intrinsic parameters*

3.4 The pipeline

The proposed novel visual asset tracking pipeline is shown in fig. 3.10. The input to the pipeline is the 360 ring camera frames from the SpotCAM+. These images are first undistorted using the retrieved camera intrinsic parameters. Then the container detector detects containers in all the images. These detections are provided to the container tracker, generating separate tracks for each of the five cameras. A fiducial detector is then run within the track bounding boxes to determine the tracked container's ID. The remaining non-identified containers will be localized by creating separate container models for each track. This is achieved by first extracting interest points inside the interior of the bounding box tracks, followed by matching across three consecutive frames. Initial reconstruction of a container point cloud is constructed before being optimized to reduce reprojection errors. The optimized point clouds are then fit to spherical models using RANSAC. As several models for the same container might exist, a final merging of container models is performed at the end. The location and potential ID and the number of containers can then be determined and uploaded to the cloud. These steps will be explained further in the following subsections.

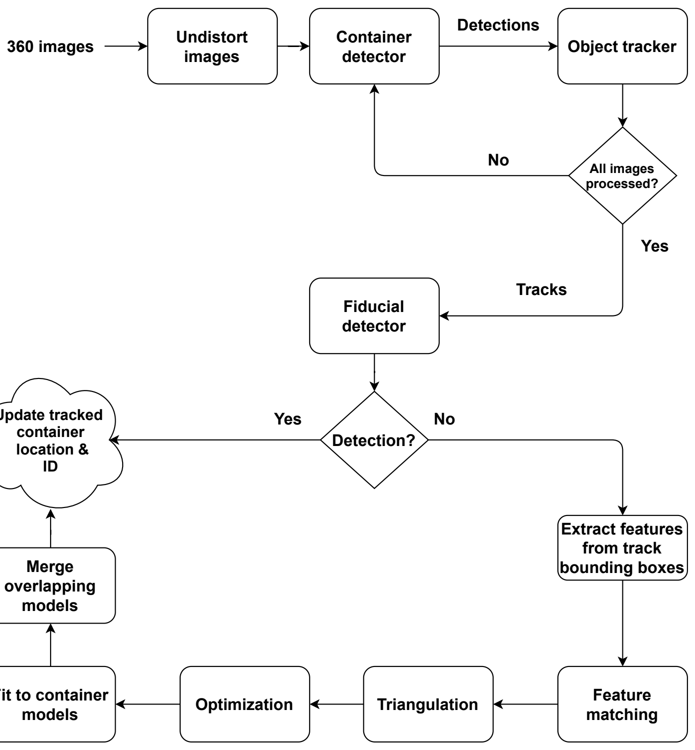


Figure 3.10: The novel visual asset tracking pipeline.

3.4.1 Undistortion of images

In fig. 3.11 the view from the five fisheye cameras which forms the 360-view is showcased. These fisheye cameras suffer from extreme radial distortion. Both the radial- and tangential distortion is removed by using the techniques of undistortion introduced in section 2.3.2. The resulting undistortion as seen in fig. 3.12 has been achieved by utilizing the OpenCV function *fisheye.undistortImage()*.

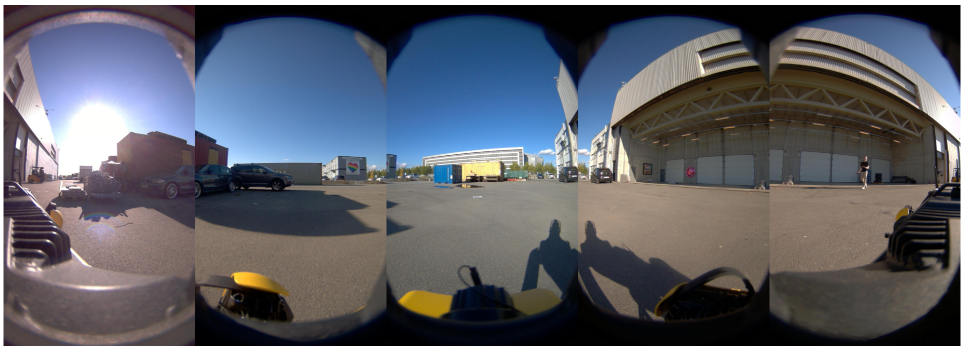


Figure 3.11: The view of the five fisheye cameras. Notice the severe radial distortion effect in the images.

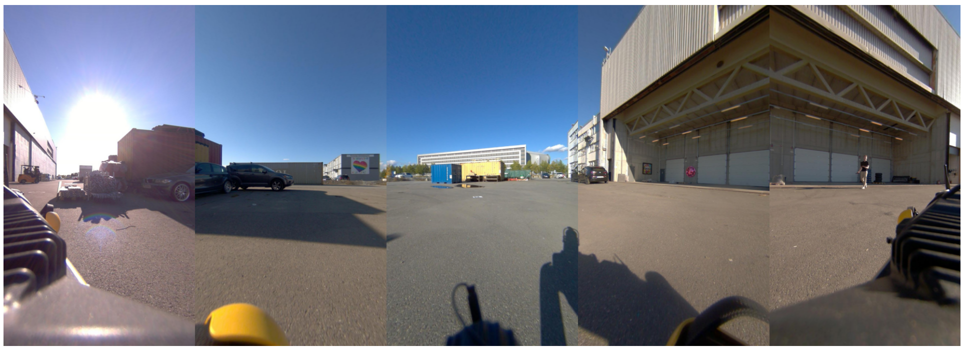


Figure 3.12: The view after undistortion of the five cameras.

3.4.2 Detection of containers

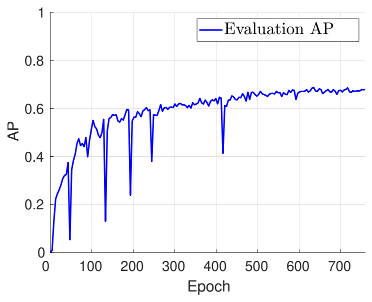
The detection of containers in the undistorted images is accomplished by utilizing the EfficientDet-D1 detector. The developed model in the project thesis did not generalize to the new image data and thus had to be retrained using the additional data as described in section 3.3.

Training and evaluation

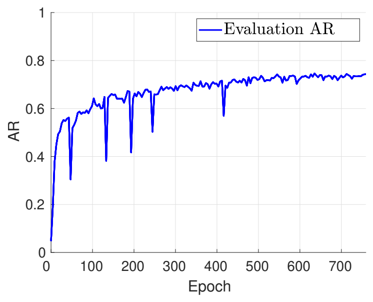
Following the conventions of the project thesis, a pre-trained EfficientDet-D1 model was retrieved from the TensorFlow object detection model zoo[5]. Smooth L1- and Focal loss is used for localization- and classification loss, respectively, with equal weighting between the two. For optimization, the SGD with a momentum term of $\alpha = 0.9$ was used, where a batch size of 2 was used due to memory limitations. The learning rate was chosen as 0.000999 and annealed down using cosine decay. Batch- and L2 normalization was used to reduce the effect of overfitting. During training, the image augmentations utilized were random horizontal flip, random scale, crop and pad to square, and random cutout. This combination of augmentations in the preliminary work resulted in the highest AP and AR metrics for the validation set.

In fig. 3.13 the training of the new EfficentDet-D1 container detector is shown. The evaluation metrics AP and AR are shown in 3.13a and 3.13b, while the classification- and localization losses are shown in 3.13c and 3.13d, respectively. The final model was exported at 716 epochs, as this was the point in the training process which resulted in the highest AP and AR scores and the lowest classification- and localization evaluation losses.

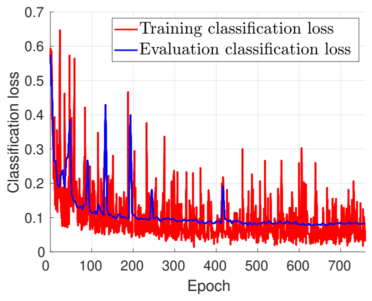
[5]https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md



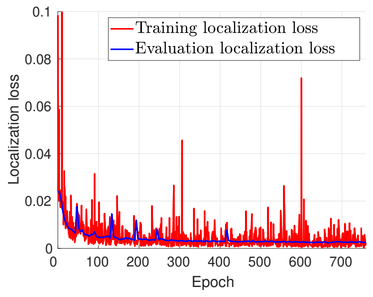
(a) AP on the validation set



(b) AR on the validation set



(c) Classification loss



(d) Localization loss

Figure 3.13: The training of the new EfficientDet-D1 model where the AP, AR, classification- and localization losses are plotted against the epoch number.

Detections

In fig. 3.14 below, four detections are shown with the confidence scores in the upper right corner of the respective bounding boxes. The output bounding boxes from the detector are scaled from the model resolution 640 × 640 to the original resolution of the images before down-scaling. All detections with confidence scores below 0.30 are discarded.

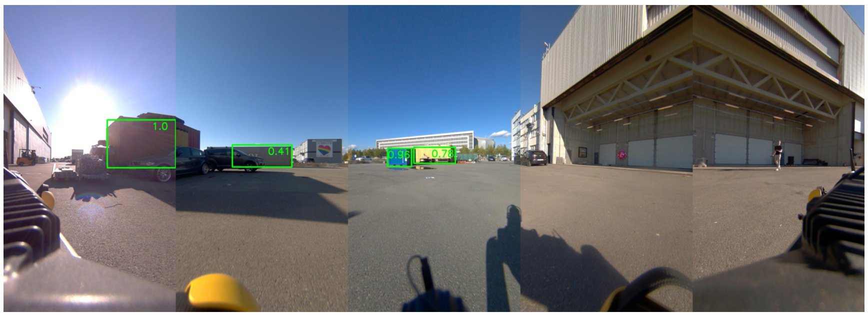


Figure 3.14: Four detections with confidence scores of 1.0, 0.41, 0.96, and 0.78 for the bounding boxes left to right in the five cameras.

3.4.3 Tracking of containers

The tracking of containers in the five cameras is accomplished by utilizing the tracking by detection algorithm SORT[6]. Each camera is processed separately, generating and sustaining tracks using the corresponding bounding boxes from the detector. Thus the tracking in each camera has no knowledge of the tracking in the other cameras. Extending this to track across all the cameras is discussed briefly in chapter 4.

Using SORT is highly motivated by the fact that this algorithm is very simple, yet provides an elegant solution to the tracking problem by utilizing the rudimentary techniques of Kalman Filtering and Hungarian Algorithm. In addition to being very simple, according to the MOT2020 challenge benchmark statistics[7], SORT is, at the best of my knowledge, the *best* performing algorithm which is open-sourced, not learning-based, and considered a tracking by detection approach. The first and second criteria are important due to this project's time limit, while the third criteria is a consequence of the continuation of the preliminary work.

The hyperparameters are the same parameters as the authors, except increasing the maximum number of frames to keep a track alive without detections from $T_{lost} = 1$ to $T_{lost} = 3$. This was found by manually tuning to yield higher MOT metric scores during implementation. The remaining default hyperparameters are $IoU_{min} = 0.30$ and $T_{min} = 3$.

Tracks

In fig. 3.15, the first, fifth, and tenth frame is shown with the generated tracks using SORT. The identity of the tracked bounding boxes is displayed in the upper right corner of the respective boxes. From the figure, IDs are carried across frames for each camera, but the IDs are not carried across cameras.

[6]`https://github.com/abewley/sort`
[7]`https://motchallenge.net/results/MOT20/`

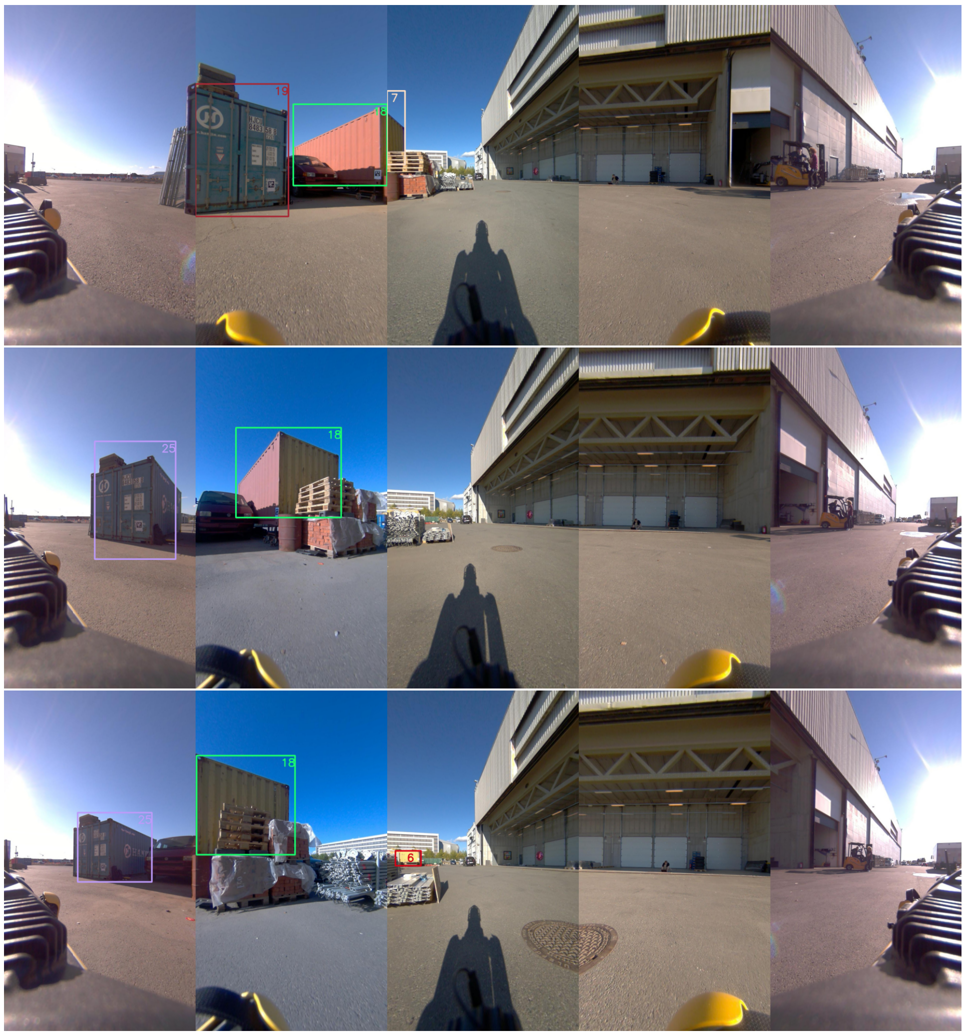


Figure 3.15: Five unique tracks for the first, fifth and tenth frame in a video sequence. Track IDs are found in the top right corner of the bounding boxes.

3.4.4 Identification of containers

The identification of containers is achieved by utilizing AprilTags of the *tag36h11* family, and the AprilTag 3-detector[8]. As these tags are commonly used for navigation and other tasks, only detections inside bounding boxes are kept. This is to ensure that the tag is correctly associated with a container.

The primary motivation for using AprilTags for identification is that the Spot platform is already dependent on these tags to operate autonomously. By using the same system for the identification of containers, the number of

[8] https://github.com/AprilRobotics/apriltag

fiducials in the scene increases. This will most likely have the advantageous side-effect of increased quality of the robot pose estimation. In addition to providing a unique ID for each container, the tag also yields the tag's position relative to the camera. These detections will therefore also serve as ground truths in locating containers in the pipeline.

Detections

Two detections are shown in fig. 3.16, which yields the spatial location in the image, the ID, and the relative pose of the AprilTag relative to the camera of which the detection was made. This relative pose can be transformed to the inertial frame as seen in eq. (3.1) to yield the pose of the tag relative to the inertial frame.

$$\mathbf{T}_{inertial,\, apriltag} = \mathbf{T}_{inertial,\, body}\mathbf{T}_{body,\, camera}\mathbf{T}_{camera,\, apriltag} \tag{3.1}$$

An essential element in acquiring an accurate pose estimate is that the physical size of the tag is known. The printed tags were measured to have a size of $0.167m$ and updated accordingly in the software.

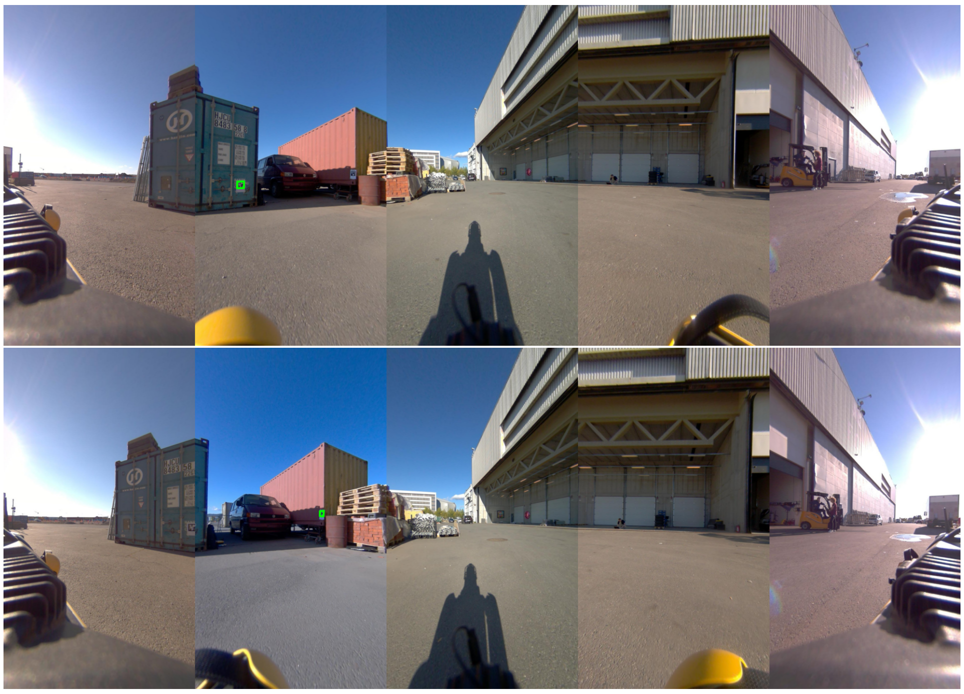


Figure 3.16: Two AprilTag fiducials are detected and visualized using green bounding boxes. Both detections are in the second camera for two consecutive frames.

3.4.5 Reconstruction of container models

The following section describes the 3D Euclidean reconstruction of the tracked containers. This reconstruction processes each track separately, where SIFT is applied to all bounding boxes in the track for feature extraction and description. This is followed by an initial matching between every three consecutive frames in the video. The epipolar constraint and the estimation of two homographies, one for each side of the container, are utilized to ensure good matches. A three-view reconstruction of the tracked container is achieved by an initial triangulation followed by a final refining bundle adjustment. Finally, container models are estimated and merged using RANSAC. Note that only two views are used to illustrate the feature detection and matching.

Feature extraction

The first step in the reconstruction process is to extract and describe the interior features of bounding boxes. The SIFT feature detector is chosen for this purpose as the research suggests that this detector should work *best* as there is no computational limit. In fig. 3.17 this process is shown, where SIFT features are shown in red and the bounding box in green.

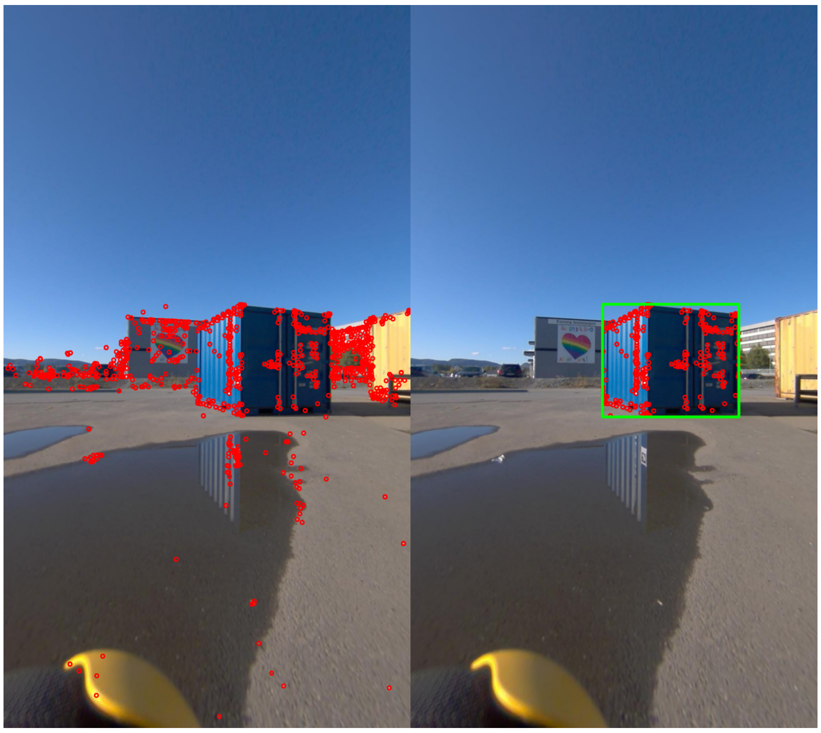


Figure 3.17: A SIFT detector is run on the image to the left. Only the features within the green bounding box in the image to the left are kept for further processing, as shown in the right image.

Feature matching

The second step is to match features between frames to find correspondences for the structure computation. First, each descriptor is compared to all the other descriptors in the set using brute force. The two closest matches for each descriptor are then evaluated against each other using the ratio test to determine the final match for each descriptor, where an NNDR ratio of 0.80 was utilized based on section 2.4.3. The choice of brute force is motivated by no computational limit, and brute force will generally yield the best initial set of matches.

In fig. 3.18 an initial set of matches between two frames is showcased. From the image, it is evident that there are quite a few bad matches that we need to remove in order to achieve a good reconstruction. These poor matches can be explained by the feature detector describing the features in a local manner, and thus global consistency is not guaranteed.

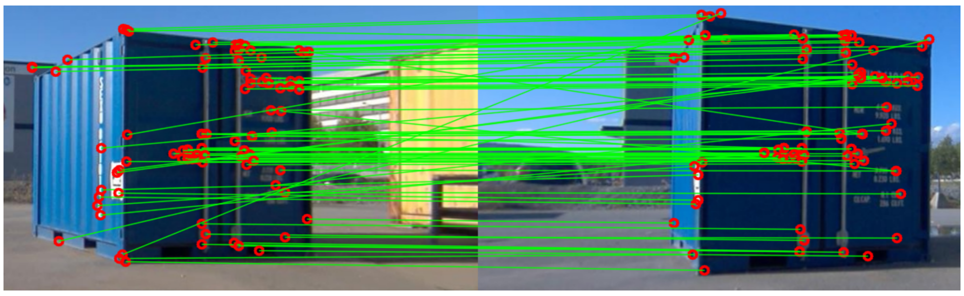


Figure 3.18: The brute force matching approach combined with the NNDR ratio test yields the following initial matches for two consecutive frames. Notice the evident mismatches.

Epipolar constraint filtering

The epipolar geometry between two views has the important property that given the fundamental matrix, a point in the first view should lie on the corresponding line in the second view as shown in eq. (2.32). However, this epipolar constraint will rarely be satisfied, as there is noise and uncertainties in the point correspondences, camera intrinsic parameters, and the motion between the two frames. Thus the best we can do is to check that each point lies within a neighborhood of the epipolar line. This neighborhood is chosen as 5 pixels, where the distance to the line is calculated using eq. (3.2).

$$d(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \tag{3.2}$$

In fig. 3.19 the upper image shows the epipolar lines for the point correspondences. After removing epipolar constraint violations, we end up with the matches shown in the bottom image. Notice how there still are mismatches that need to be removed.

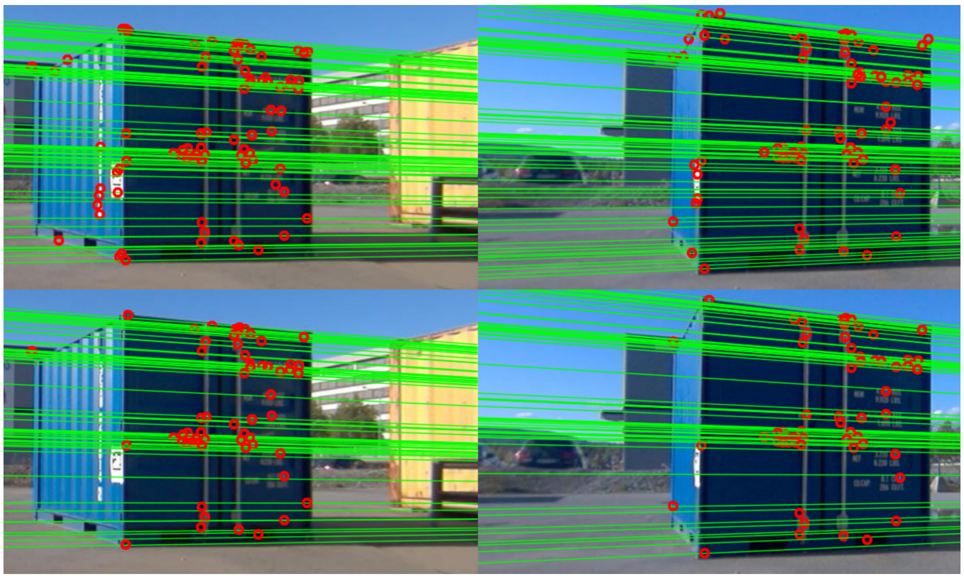


Figure 3.19: The upper image shows the epipolar lines of the correspondences. The bottom image shows the epipolar lines after the removal of constraint violations.

Homography filtering

By estimating two homographies between the two views, one for each visible side, correspondences are validated by checking that the reprojection error is below some threshold. This threshold is set to three pixels, lower than the epipolar filtering threshold. The estimation is performed inside a RANSAC scheme, where each homography is iteratively estimated by randomly using 4 point correspondences in each iteration. In fig. 3.20 the correspondences from the epipolar filtering, which also satisfies the estimated homographies are visualized. There are no remaining evident mismatches that need to be filtered out. This estimation has been carried out by utilizing the OpenCV function *findHomography()*.

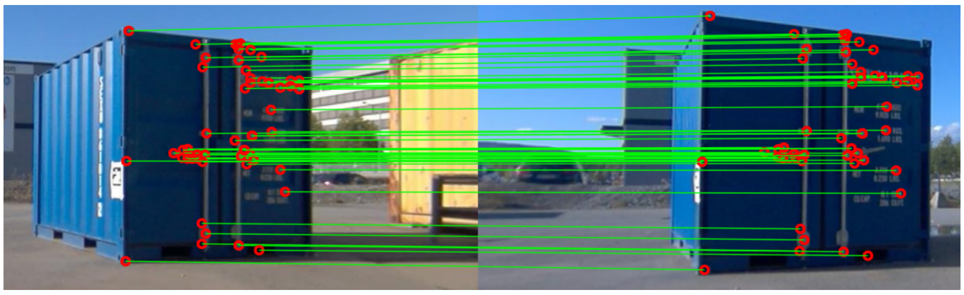


Figure 3.20: The final matches after homography filtering. There are no evident mismatches in these two images.

Point cloud generation of tracks

For each track, all matches across three consecutive frames in the track are triangulated using DLT to yield an initial reconstruction of the tracked container. This linear triangulation often provides good results with acceptable reprojection errors but will occasionally fail and produce poor estimates of the 3D points. Therefore, a refining SOBA was implemented to perform a nonlinear least-squares optimization of the reprojection errors. This optimization was carried out by utilizing the SciPy function *least_ squares()*[9] part of the optimize library. The Levenberg Marquardt method was chosen to perform the optimization, as this is the standard optimizer for BA tasks. The resulting track point clouds for sunny conditions, using ground truth tracks, are visualized in fig. 3.21. The robot trajectory is shown in blue in this plot and starts at the orange triangle. The detected AprilTags are visualized as squares and the point clouds as circles of different colors. From this plot, it is clear that there are quite a few outliers and overlapping point clouds.

[9] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html

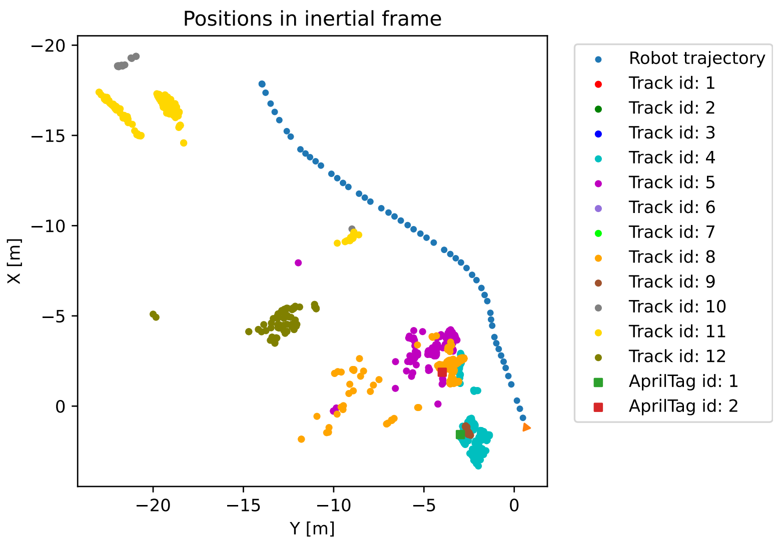


Figure 3.21: The resulting track point clouds in sunny conditions using ground truth tracks. The robot trajectory is shown in blue, starting at the orange triangle and displaying AprilTags as squares.

Reducing the reconstruction uncertainty

A two-view reconstruction of a point for three different configurations is shown in fig. 3.22. The shaded region in each case illustrates the uncertainty of the estimated 3D point. The angle which forms between these rays is known as the parallax, and an increasing parallax generally reduces the uncertainty. In the case of forward motion, as seen in the right image, the uncertainty of reconstruction increases. This parallax can easily be determined using the dot product between the rays and solving for the angle. Another concept to address this problem is by utilizing a large enough effective baseline, which is the distance along the camera Y axis from the first frame to the second frame.

The 1:30 rule is a proposed method to calculate the baseline for a specific 3D depth. For a 3D point located 12 meters from the camera, the suggested required baseline is 0.40m. This baseline is used as the effective baseline for the three views. Then a minimum parallax of 5 degrees is imposed on the estimated 3D points. This parallax value is inspired by ORB-SLAM3[10], where they utilize a minimum parallax of 1.15 degrees.

[10]https://github.com/UZ-SLAMLab/ORB_SLAM3

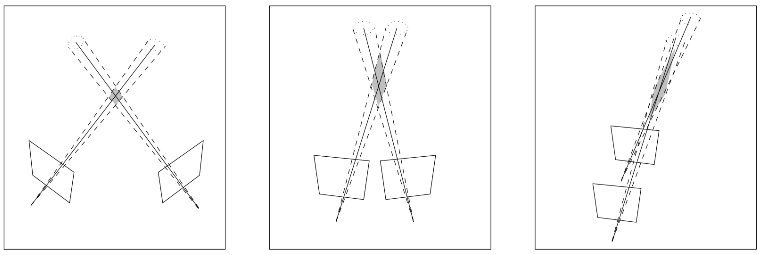


Figure 3.22: The angle between the rays determines the uncertainty region shaded in gray for the estimated point. A larger angle reduces the uncertainty. Source: [28] p. 321 (retrieved 2021)

Estimation of container models

The container models are robustly estimated using RANSAC, fitting each point cloud track to spherical models given by eq. (3.3). This simplification, approximating the container cuboids into spheres, is highly motivated by spherical models being simpler to estimate and apply. A preciser approach would have been to estimate cuboid models instead. However, this is out of the scope of this thesis, as both the estimation process and working with the model are substantially more tedious. This is discussed in chapter 4.

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \tag{3.3}$$

A maximum radius of 1.8 m is imposed, which was determined by utilizing Pythagoras's theorem and the width and height of the container. A minimum radius of 1.0 m was imposed to neglect questionable point clouds that can be triangulated container occlusions such as cars, scaffolding, or pallets.

The resulting inliers of the estimated spherical models are shown in fig. 3.23. This plot shows that outliers are removed, and some track point clouds are overlapping. Spherical models with fever than 20 inliers are discarded.

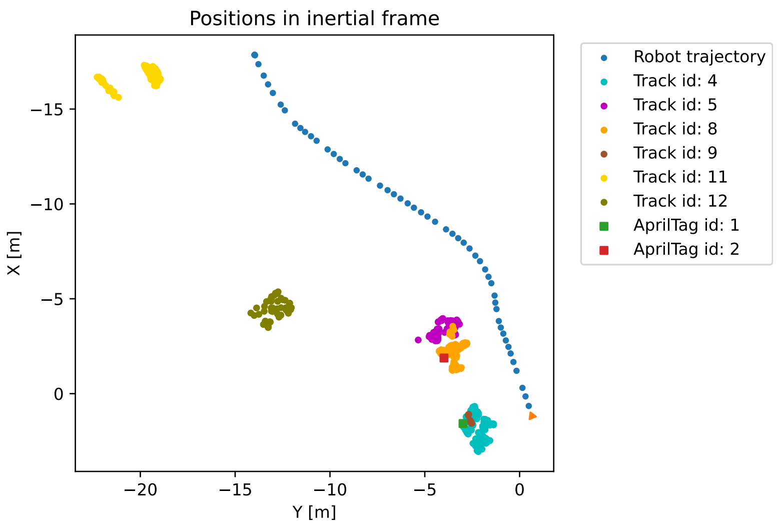


Figure 3.23: The resulting model inliers in sunny conditions using ground truth tracks. Notice how outliers are removed, some models overlap, and models with too few inliers are discarded.

3.4.6 Merging of container models

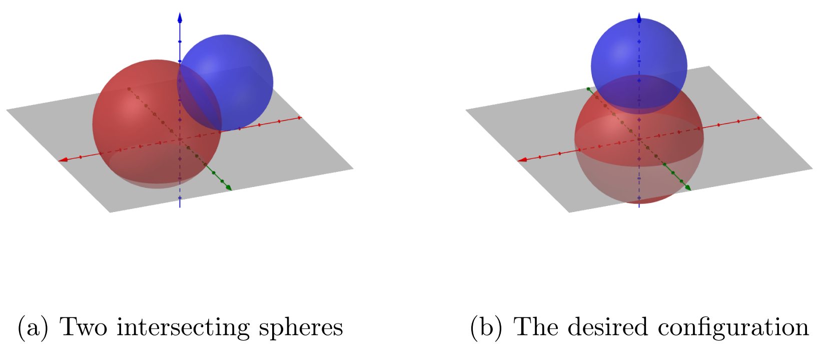
A consequence of creating a container model for each track is that there will be several models of the same container. Thus it would be favorable to merge all these into one single model. By utilizing the estimated container models, two models intersect if the sum of the two model radii is larger than the distance between the centers. Merging models based solely on if the models are intersecting is naive. An addition to this is to compute the intersection volume and determine the factor of which this volume makes up for the smallest model volume. If this factor is greater than some threshold, we merge the two models.

In fig. 3.24a, two intersecting spheres are shown of which can be described by the two mathematical formulas as seen in eq. (3.4). This intersection forms two spherical caps which entails the volume of intersection $V_I = V_{c1} + V_{c2}$.

$$\begin{aligned}(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_2^2\end{aligned} \tag{3.4}$$

In order to determine the volume of the caps, we transform the spheres such that the first sphere is centered at the origin and the second sphere above the first sphere aligned with the z-axis as seen in fig. 3.24b. This is achieved by subtraction of the first sphere center coordinates followed by a calculation of the distance between the centers d and moving the second sphere center such that the center is at Z $= d$. The resulting transformed models are seen in eq. (3.5).

$$\begin{aligned} x^2 + y^2 + z^2 &= r_1^2 \\ x^2 + y^2 + (z-d)^2 &= r_2^2 \end{aligned} \tag{3.5}$$



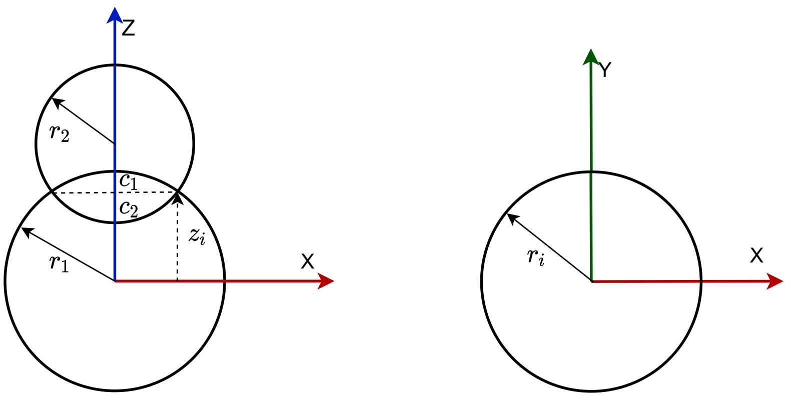
(a) Two intersecting spheres (b) The desired configuration

Figure 3.24: The spheres before transformation are shown in (a), while (b) showcases the desired transformed configuration.

The resulting geometry of the two intersecting spheres are shown in fig. 3.25. In fig. 3.25a the intersection in the X-Z plane is showcased. The two spherical caps are denoted as c_1, c_2 and the intersection height as z_i. The intersection forms a circle of radius r_i in the X-Y plane as shown in fig. 3.25b. By equating the equations in eq. (3.5) the intersection height as shown in eq. (3.6) is determined. Inserting z_i in to one of the equations in eq. (3.5) yields r_i as given in eq. (3.7).

$$z_i = \frac{r_1^2 - r_2^2 + d^2}{2d} \tag{3.6}$$

$$r_i = \sqrt{r_1^2 - z_i^2} \tag{3.7}$$



(a) The intersection in the X-Z plane with the two spherical caps c_1, c_2 and the intersection height z_i.

(b) The intersection forms a circle in the X-Y plane of radius r_i.

Figure 3.25: The geometry of the two intersecting spheres in the X-Z and X-Y planes for (a) and (b), respectively.

The resulting volume of the first spherical cap V_{c1} can then be determined by the volume integral as shown in eq. (3.8), where cylindrical coordinates are utilized. The upper limit in the inner integral is determined by solving the first equation in eq. (3.5) for z.

$$V_{c1} = \int_0^{2\pi} \int_0^{r_i} \int_{z_i}^{\sqrt{r_1^2 - r^2}} r\, dz\, dr\, d\theta \tag{3.8}$$

The final closed-form solution to this integral is given by eq. (3.9). By symmetry, the volume of the second spherical cap is given by eq. (3.10).

$$V_{c1} = \pi\left(\frac{z_i^3}{3} - r_1^2\left(z_i - \frac{2r_1}{3}\right)\right) \tag{3.9}$$

$$V_{c2} = \pi\left(\frac{(d - z_i)^3}{3} - r_2^2\left((d - z_i) - \frac{2r_2}{3}\right)\right) \tag{3.10}$$

The portion P of this volume for the smallest sphere is then determined as shown in eq. (3.11), where V_1, V_2 is the volumes of the two spheres.

$$P = \frac{V_{c1} + V_{c2}}{\min\{V1, V2\}} \tag{3.11}$$

The merging of M container models follows a greedy scheme, where model pairs with the highest calculated P are merged by utilizing RANSAC to estimate a new spherical model. Once a model is merged, we have $M - 1$

models remaining, of which we repeat this merging procedure. The merging terminates once there are no sphere pairs with a factor of volume intersection for the smallest sphere, above a threshold of 0.20. This threshold was found to be a good choice empirically by testing different thresholds.

The resulting inliers of the merged spherical models are shown in fig. 3.26. The overlapping tracks have been merged, and the final container locations can be determined as the center of the estimated spheres.

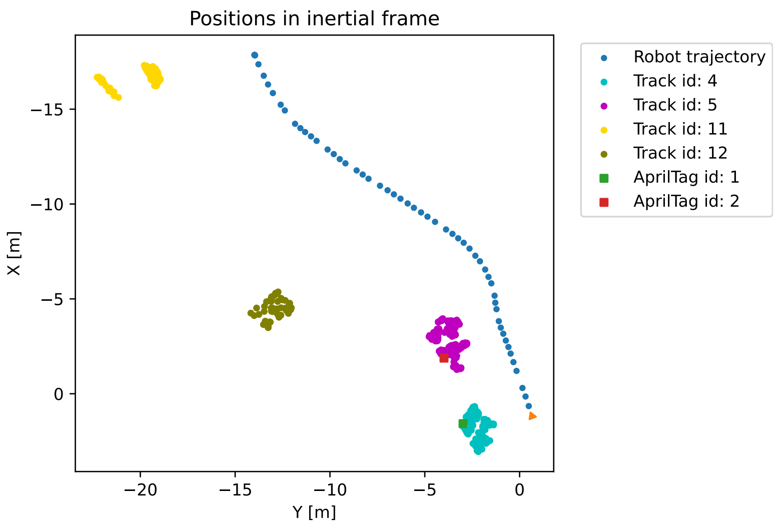


Figure 3.26: The resulting merged model inliers in sunny conditions using ground truth tracks. Notice how overlapping models are merged.

3.4.7 Final localization, identification, and counting

The last step in the pipeline is determining the final container locations, identity, and the number of containers present in the scene. In order to quantify the success of the pipeline, the metrics Localized Assets (LA), Localized Assets correct with respect to the Ground Truth (LAGT), Identified Assets (IDA), and Duplicate Models (DM) are introduced. These metrics are constructed as simple as possible to lighten the interpretation of the results.

Localization

From the estimated models of the containers, the final container locations are determined as the center of the respective models. For sunny conditions using ground truth tracks as seen in fig. 3.26, this results in a total of four container locations. The LA metric is introduced to measure the fraction of assets in the scene which is localized. The calculation of this metric is shown in eq. (3.12), where DM are not included in the numerator. The resulting LA metric for sunny conditions using ground truth tracks becomes $\frac{4}{5}$, as there are a total of five containers in the scene.

$$\text{LC} = \frac{\text{number of localized assets}}{\text{total number of assets in the scene}} \tag{3.12}$$

In order to measure the correctness of the localization of the containers, the detected container AprilTags are utilized as ground truths. A container is said to be correctly localized if the estimated model contains the correct ground truth tag. The LAGT metric is then defined as seen in eq. (3.13). The resulting LAGT metric for sunny conditions using ground truth tracks becomes $\frac{2}{2}$, as there are only two detected ground truth container tags.

$$\text{LCGT} = \frac{\text{number of correctly localized assets}}{\text{total number of assets with ground truth location}} \tag{3.13}$$

Identification

The identification of containers is achieved by the detection of container AprilTags. The IDA metric as shown in eq. (3.14) is introduced to measure the fraction of assets that are identified in the scene. Only assets equipped with an AprilTag are considered in the calculation of the metric. The resulting LA metric for sunny conditions using ground truth tracks becomes $\frac{2}{4}$, as only two out of the four containers with AprilTags are identified.

$$\text{IDC} = \frac{\text{number of identified assets}}{\text{number of assets with tags in the scene}} \tag{3.14}$$

Counting

The counting of containers is achieved by counting the number of models estimated, including duplicate models. In fig. 3.26 this would count to a total of four containers. The DM metric as shown in eq. (3.15) measures the number of duplicate models.

$$\text{DM} = \text{number of duplicate models} \tag{3.15}$$

Chapter 4

Results and Discussions

This chapter will present and discuss the results from the implemented visual asset tracking pipeline. All results presented in this chapter are primarily based on the two videos gathered in sunny and overcast weather. Section 4.1 presents the EfficientDet-D1 detection metrics, while section 4.2 presents the SORT tracking metrics and discusses why the tracking is better for overcast conditions. In section 4.3, the estimated container models using both ground truth- and generated tracks from the SORT tracker are presented. Finally, in section 4.4 the final pipeline metrics are presented. For the remainder of this chapter, section 4.5 discusses the reconstruction accuracy, section 4.6 the parameters and threshold in the pipeline, section 4.7 the sphere approximation, and section 4.8 discusses object occlusions.

4.1 Container detection

The retrained container detector achieved a final test AP and AR of 69.1% and 74.4%, respectively. In table 4.1 the metrics for sunny- and overcast conditions are shown. These are used to summarize the 12 COCO metrics, which are given in appendix A. The highest metric scores are given in the bold text in this table. It is clear that the developed model performs better for overcast weather. This might be explained by an imbalance in the data set used during training, where approximately 70% of the images are in overcast- or rainy weather. Despite this, the AP and AR for both conditions are impressive.

Table 4.1: Container detection metrics

Weather	AP	AR	AP_{50}	AP_S	AP_M	AP_L
Sunny	0.632	0.706	0.922	-	0.149	0.660
Overcast	**0.672**	**0.735**	**0.982**	-	**0.383**	**0.680**

4.2 Container tracking

Table 4.2 shows the resulting MOT metrics for each ring camera $c0, c1, c2, c3$, and $c4$ in sunny and overcast conditions, where the highest metrics are given in bold. Like in the detection of containers, the tracking metrics are better in overcast weather. The MOTP scores are quite similar, while the MOTA scores in sunny- and overcast weather are 58.3% and 83.1%, respectively. Additionally, the sunny conditions have ten identity switches, while there is only one identity switch in overcast conditions. These notable differences can be partially explained by that SORT is a DBT approach, where the detections in sunny conditions were worse than in overcast conditions. This is a known drawback of tracking-by-detection trackers, where the tracking quality is heavily dependent on the quality of the detections.

Despite the large difference in the MOTA score and IDs for sunny and overcast weather, the containers are generally mostly tracked or partially tracked. This is reflected in the overall scores, where the MT and PT scores are 45.5% and 50.0%, respectively. The MOT metrics were calculated using *py-motmetrics*[1].

Table 4.2: Container tracking metrics

Weather	Camera	MOTA	MOTP	MT	PT	ML	IDs
Sunny	c0	0.628	0.789	0.250	0.500	0.250	2
	c1	0.450	0.780	0.000	1.000	0.000	5
	c2	0.671	0.795	0.333	0.666	0.000	3
	c3	-	-	0	0	0	**0**
	c4	-	-	0	0	0	**0**
Overcast	c0	0.886	0.778	**1.000**	0.000	0.000	**0**
	c1	0.677	0.780	0.600	0.400	0.000	1
	c2	**0.929**	**0.820**	**1.000**	0.000	0.000	**0**
	c3	-	-	0	0	0	**0**
	c4	-	-	0	0	0	**0**
Overall	-	0.681	0.790	0.455	0.500	0.045	11

[1]https://github.com/cheind/py-motmetrics

Linear constant velocity model

The SORT algorithm uses linear constant velocity to model the dynamics of the bounding boxes in the Kalman Filter. For this specific data set, the robot is mainly moving at the preset *nominal* walking speed, making this approximation justifiable. However, as rotation movement is introduced to the system, a linear acceleration will be induced. Additionally, as we saw in section 3.3, the data is not gathered at equal time intervals due to the failure of log-point acquisitions, resulting in additional linear accelerations. In fig. 4.1 the issue of rotational movement is showcased for frame 46 and 47 in sunny conditions. All tracks are lost in the next frame as the robot turns clockwise. This resulted in new track IDs once the robot stopped rotating. Figure 4.2 and fig. 4.3 shows the angular acceleration of the robot body Z-axis α_z between consecutive frames in the sunny and overcast videos respectively. The red circles indicate when tracks are lost in the video, while the green circles at which the previous frames were skipped. The failure to acquire log points can amplify the acceleration and cause the tracks to be lost, as shown by the overlapping red and green circles. However, causation is not possible to determine from these plots. The only thing for certain is that the acceleration is nonzero, violating the linear constant velocity assumption.

Additionally, the calculated mean acceleration for sunny conditions is approximately four times the mean acceleration in overcast conditions, while the variance is 12% higher for overcast conditions. This suggests that the sunny conditioned video is exposed to additional rotational movement compared to the overcast video, which might explain why the tracking metrics are worse for sunny conditions.

Using SORT is therefore limited to inspections where the robot is mainly moving in constant forward motion. If this is not the case, then more eminent trackers such as DeepSORT [88], or TransCenter [89] should be considered.

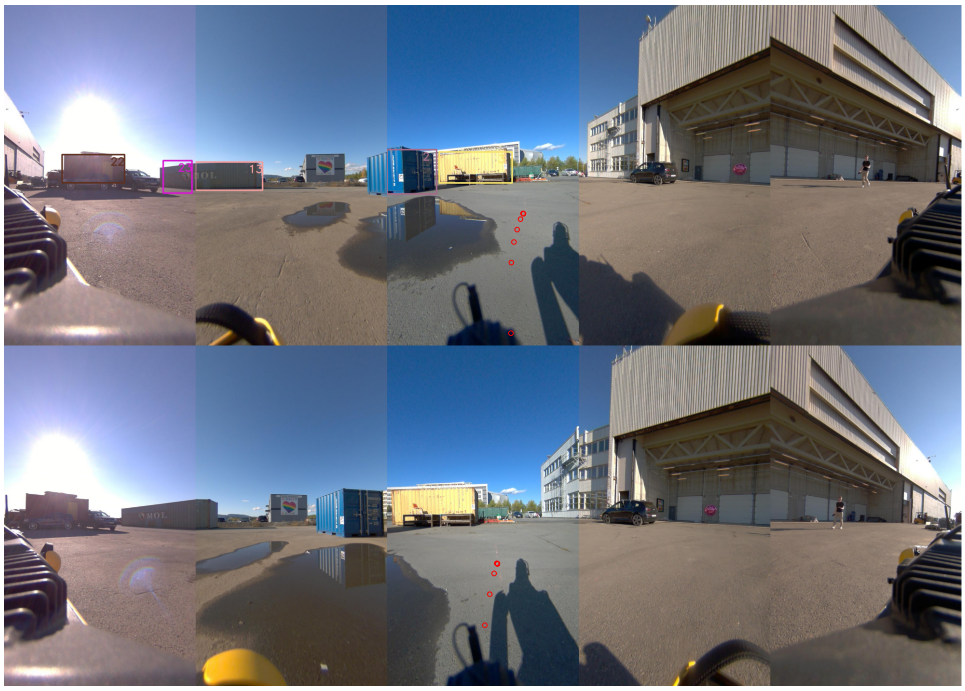


Figure 4.1: The tracks are lost in the two consecutive frames 46 and 47 due to the rotation of the robot.

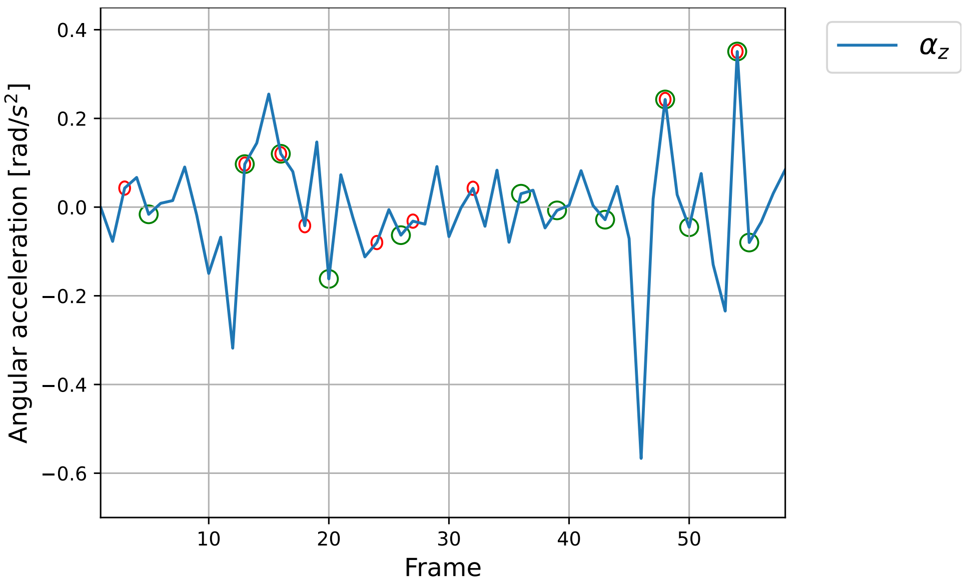


Figure 4.2: The angular acceleration of the robot body Z-axis between frames in the *sunny* conditions video. The red circles indicate at which angular accelerations tracks are lost, while the green circles at which frames the previous frame were skipped.

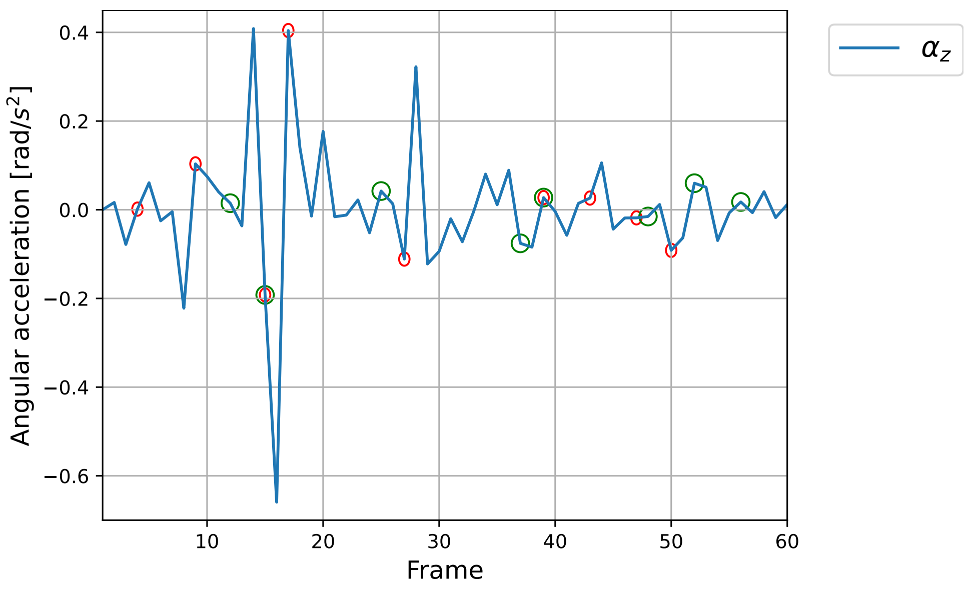


Figure 4.3: The angular acceleration of the robot body Z-axis between frames in the *overcast* conditions video. The red circles indicate at which angular accelerations tracks are lost, while the green circles at which frames the previous frame were skipped.

4.3 Container model estimation

This section will present the resulting estimated container models in sunny- and overcast conditions using ground truth- and generated tracks from SORT. As explained in section 3.3, the containers are given the names B1, R1, G1, B2, and Y1 in order to separate them more conveniently in the analysis.

Sunny conditions

In fig. 4.4 the estimated container model inliers are shown, where ground truth tracks have been utilized. This is the same plot as we saw in chapter 3, which shows that there are an estimated four containers in the scene. The B1, R1, G1, and B2 containers are modeled with track identities 4, 5, 12, and 11, respectively. The Y1 container is not localized. This is due to this container only being tracked in the forward-facing $c2$ camera, which due to forward motion are generally not considered as the baseline between the frames is too small. The noticeable gap between the inliers for tracks 4 and 5 is due to merging two or more overlapping models.

The resulting model inliers using the generated tracks are shown in fig. 4.5. The B1, R1, G1, and B2 containers are modeled with track identities 19, 18, 14, and 12, respectively. In contrast to using ground truth tracks, there are an estimated number of five containers. The R1 container is modeled as two containers with track identities 18 and 22. Even though the models seem to overlap, this is not the case as the 2D plot does not account for the placement of the spheres in the Z-direction. The two quite different models of the same container are a consequence of the triangulation of an occluding car for track id 22, which is discussed in section 4.8.

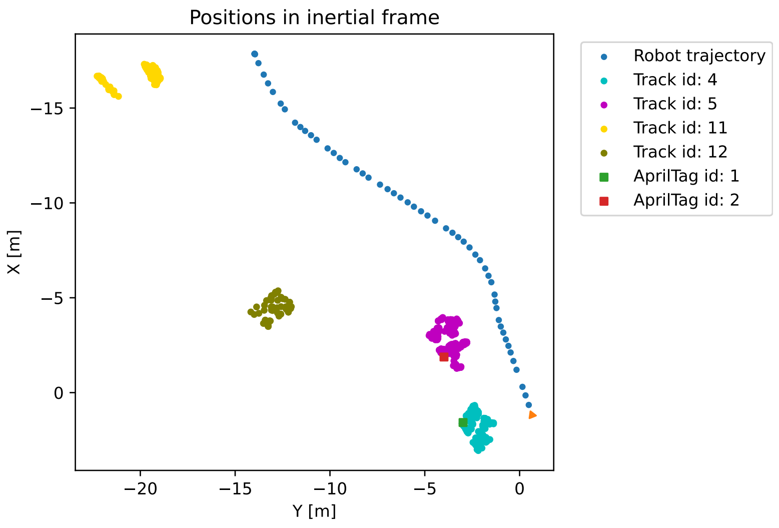


Figure 4.4: The estimated container model inliers in sunny conditions using ground truth tracks.

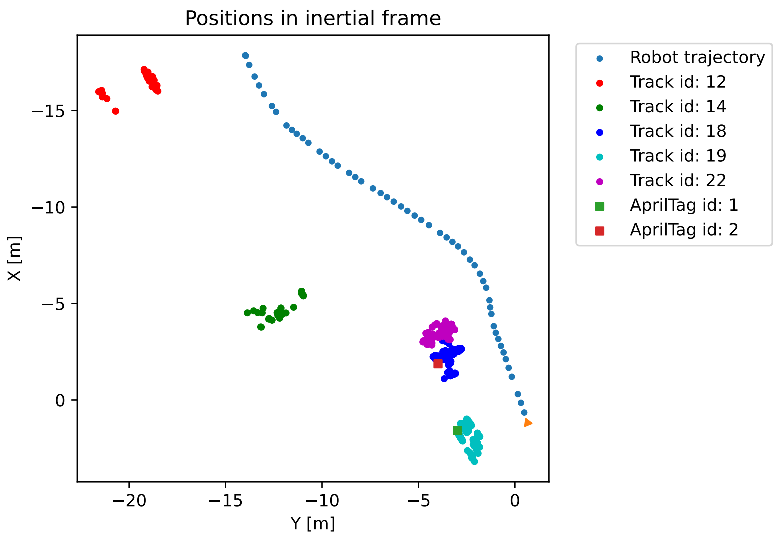


Figure 4.5: The estimated container model inliers in sunny conditions using generated tracks.

Overcast conditions

In fig. 4.6 the estimated container model inliers are shown, where ground truth tracks have been utilized. This plot shows an estimated four containers in the scene with track identities 3, 4, 8, and 9 for B1, R1, G1, and B2, respectively. Like in sunny conditions, the Y1 container is not localized. A notable difference from sunny conditions is the localized position of the G1 container, which by visual inspection of the plot looks to be very close to the R1 container. This is not the case and suggests that the estimated depth of the G1 container is not satisfactory. Reasons for this poor reconstruction is discussed in section 4.5.

The resulting model inliers using the generated tracks are shown in fig. 4.7. The R1, B1, G1, and B2 containers are modeled with the track identities 13, 10, 8, and 6, respectively. From this plot, we observe the same issue as in sunny conditions, and the R1 container is modeled twice with the track identities 10 and 15. The Y1 container is not localized, and the depth of the G1 container is arguable of higher quality than using ground truth tracks.

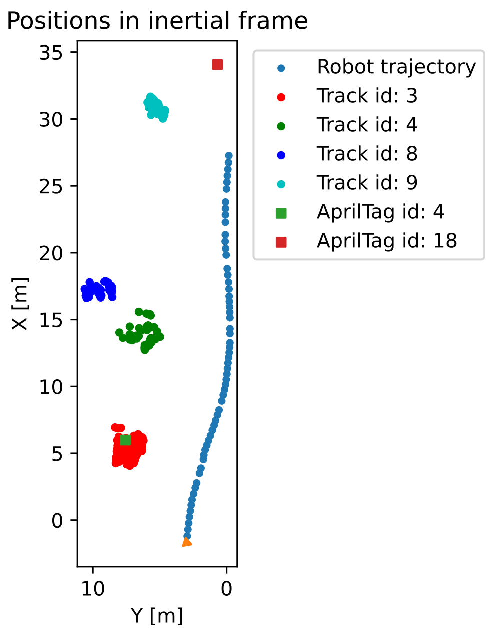


Figure 4.6: The estimated container model inliers in overcast conditions using ground truth tracks.

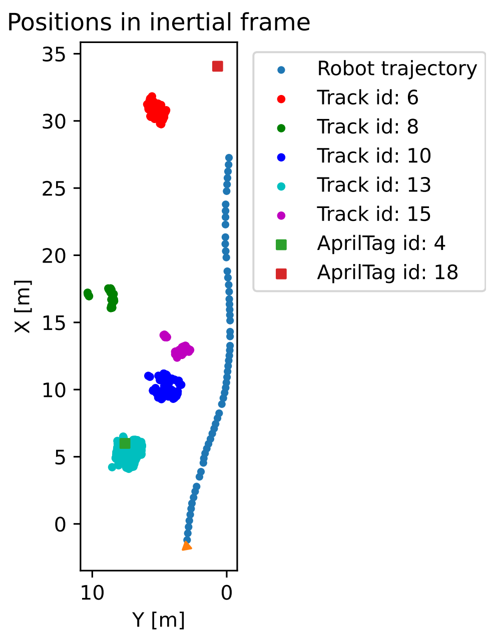


Figure 4.7: The estimated container model inliers in overcast conditions using generated tracks.

Summary

From the plots showcased in this section it is clear that the pipeline does an excellent job localizing four out of the five containers in the scene. However, it is impossible to determine the estimated models' correctness with no ground truth AprilTag. For the models with ground truths, such as B1 and R1 in sunny conditions, and B1 in overcast conditions, the AprilTag is contained within the estimated models. The Y1 container is not localized in any of the cases due to poor baseline in the forward-facing camera, and the G1 container's depth is likely very uncertain.

4.4 Final pipeline metrics

Table 4.3 shows the final pipeline metrics as introduced in section 3.4.7, for both sunny and overcast conditions. The column GTT states whether or not Ground Truth Tracks (GTT) are utilized. The fraction of localized containers is 80% for all four cases in the table. The LAGT is 100% and 50% for sunny and overcast conditions, respectively, while the IDA is 50% for all four cases. There is an additional duplicated model for both conditions when using generated tracks. This results in the final average LA, LAGT, IDA, and DM when using the generated SORT tracks of 80%, 75%, 50%, and 1, respectively.

The arguably high LA and LAGT metrics show that the pipeline does a good job locating the containers in the scene. The low IDA metric is explained by the poor performance of the AprilTag detector. This is surprising, as in the paper [85] they claim an approximately 100% detection accuracy for tags 7 m or closer to the camera. This is the case for all the tags in the scene as the robot traverses the route. A possible reason this fails is that the conducted experiments in the paper are inside a laboratory and not outside in the weather. The sunny weather and overcast conditions introduce several challenges such as sun glare, contrast, and brightness degradation. These are well-known issues in computer vision that are known to degrade the performance of vision algorithms.

It is worth noting that the LA, LAGT, IDA, and DM metrics were proposed in this thesis due to the lack of existing metrics for this novel approach. The metrics were constructed as simple as possible and are by no means perfect compared to the thoroughly researched object detection and MOT metrics.

Table 4.3: Final pipeline metrics.

Weather	GTT	LA	LAGT	IDA	DM
Sunny	yes	$\frac{4}{5}$	$\frac{2}{2}$	$\frac{2}{4}$	0
	no	$\frac{4}{5}$	$\frac{2}{2}$	$\frac{2}{4}$	1
Overcast	yes	$\frac{4}{5}$	$\frac{1}{2}$	$\frac{2}{4}$	0
	no	$\frac{4}{5}$	$\frac{1}{2}$	$\frac{2}{4}$	1
Average	yes	0.800	0.750	0.500	0
	no	0.800	0.750	0.500	1

4.5 Accuracy of reconstruction

The *accuracy* of the reconstruction process depends on multiple factors, which contributes to the overall reprojection errors of the 3D points. For the four cases, as shown in table 4.3, the overall average reprojection error is 7.266 pixels. This is conceivably a high reprojection error, as it is common to achieve substantially smaller reprojection errors during camera calibration, though this is a much simpler process than 3D reconstruction. Four critical factors contributing to this reprojection error are the camera parameters, robot poses, feature matches, and the distance to the 3D point.

Camera parameters

The camera parameters are composed of the camera intrinsic- and extrinsic parameters. These parameters were retrieved using the Spot SDK and were verified by Boston Dynamics to be the factory calibration of the cameras. Camera parameters are in general dynamic, which makes acquiring and sustaining the *correct* camera parameters impossible. This is partly because the parameters are sensitive to movement and temperature changes. The quality of the reconstruction could thus have been improved by conducting a camera calibration during the two data gatherings or by including the parameters in the bundle adjustment. This would have resulted in more accurate camera parameters and thus increased the quality of the estimated euclidean reconstruction.

Robot poses

The robot poses utilized in this work were acquired through the Spot SDK during the data gathering. The drawback of this is that it is not possible to assess the accuracy of these poses. Thus the high reprojection error might be due to poor state estimation of the Spot platform. A solution to this problem is to utilize these poses as an initial starting point in a global full bundle adjustment scheme, refining both structure and motion. However, this was out of the scope of this thesis, as taking such an approach essentially is the task of photogrammetry.

Feature mismatches

When matching features it is usually simple to obtain a set of good matches. However, it is arguably impossible to guarantee a set of correct matches. This is a result of noise in the images, which becomes evident in the estimation of the homography between two images. In this process, a maximum reprojection error threshold is set to accept inliers in the RANSAC estimation scheme. Thus in the process of feature matching, we are essentially accepting good matches, not the *correct* matches. As a consequence of this, the accuracy of the reconstruction is reduced.

Distance to point

A common notion in computer vision is that as the distance to a 3D point increases, the reconstruction uncertainty of the point increases. Two well-known approaches to address this issue are ensuring a sufficient baseline or parallax between the views. Determining the baseline or parallax depends on the application, where a higher baseline and parallax generally reduce the uncertainty of the reconstruction. However, choosing too large baselines or parallax is not practical.

4.6 Parameters and thresholds

Almost every component in the pipeline, as shown in fig. 3.10 require parameters or thresholds to be set. Finding the *best* set of parameters is a demanding task and arguably impossible for this pipeline. In this thesis, satisfactory parameters have been found empirically by first using the default or suggested parameters by the literature as a starting point. The parameters have then been tuned to provide satisfactory results, where the elements in the pipeline are tuned according to the flow of the data. The container detector's parameters are set before the container tracker and so on. As these parameters are tuned to produce satisfactory results for the two videos, this does not guarantee that these will yield similar results for new data.

4.7 Sphere approximation and merging

The containers in this work are approximated as spheres, which yielded satisfactory results for the gathered data. This approximation was motivated by that spheres are mathematically more pleasing to work with. A cubic model can be described by six individual plane equations, whereas a single equation describes a spherical model. The parameters subject to tuning are the radius for the spheres and the width, height, and length for the cubic models.

Each container in the scene can be modeled several times during the tracking. This can be due to the disappearance of a tracked container in one camera that appears in another or simply due to the track being lost and later created with a new identity. A greedy merging scheme was proposed to address the issue of duplicate models. This scheme incrementally merged the most overlapping models until no overlapping models remained above a minimum threshold overlap. This approach successfully merged the multiple models for this given data set. However, in this data set, the containers are well separated. This is seldom the case as containers are typically placed as tight as possible to utilize the space efficiently. As the data set does not capture any closely placed containers, it is impossible to determine the pipeline performance in such scenarios. This issue can be mitigated by improving the tracking performance and extending the tracking to sustain track identities across the five ring cameras. These improvements will reduce the number of duplicate models and thus reduce the number of potential candidates for merging. Assuming perfect tracking across the cameras, there will be no candidates and thus no need for merging.

4.8 Occlusions

An issue with utilizing bounding boxes for the extraction of interest points is that the bounding box may contain occluding objects. This results in a triangulation of non-container points and, consequently, duplicate models as we saw in section 4.3 for the R1 container might appear. In fig. 4.8, matches between three consecutive frames are shown. The SIFT feature detector finds the occluding car more interesting, which results in 70% of the matches being non-container points. A solution to this issue would have been to replace the bounding boxes with segmentation masks. This would have reduced the number of triangulated non-container points substantially and likely reduced the number of duplicate models in the pipeline. According to the EfficientDet paper [81], the EfficientDet model can be adjusted to also work for tasks such as semantic segmentation.

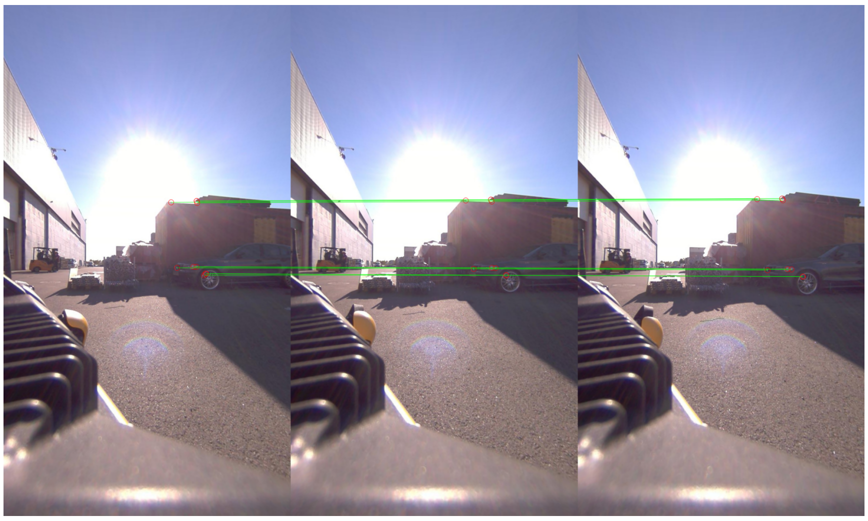


Figure 4.8: 70% of the matches between three frames are from an occluding car. Notice the sun glare (top) and flare (bottom) in the images.

4.9 Weather

During this thesis, we have analyzed the performance of the pipeline in sunny and overcast weather. The container detector and tracker achieve higher metrics in overcast conditions, while the overall pipeline metrics are higher for sunny weather. This higher pipeline metric is explained by the higher LAGT score, which measured the factor of models associated with the correct ground truth AprilTag. The Y1 model was not modeled for both weather conditions, but the ground truth was detected in overcast conditions. This resulted in an *unfair* comparison of the pipeline performance of these two weather conditions, as the LAGT metric was reduced in overcast weather due to the failure of associating the AprilTag with the Y1 model. Thus it is not correct to say that the pipeline performs better for either weather condition using the constructed metrics.

The poor performance of the AprilTag detector might be explained by a combination of weather degradation and the distance to the tags from the cameras. Prominent effects such as sun glare or flare, as seen in fig. 4.8, as well as more common brightness and contrast degradation, are known to degrade the quality of vision algorithms. According to the AprilTag paper, the detector should detect all tags as long as the distance to the tag is less than 7 m. However, it appears that these tags are placed in front of the camera with little to zero relative orientation. This is seldom the case as the robot traverses the route.

As we saw in section 4.1 and section 4.2, the detection and tracking metrics were higher for overcast conditions. This is most likely explained by a weather imbalance in the images used during training. Approximately 70 % of the images are in overcast or rainy weather. A way to improve the performance would have been to include more container images in sunny conditions.

Chapter 5

Conclusion

This thesis aimed to implement a novel visual asset tracking pipeline to localize, identify, and count the number of containers in an outdoor industrial environment. The pipeline detects containers with high accuracy in both sunny and overcast weather, where the resulting AP scores are 63.2% and 67.2%, respectively. The notable difference in the detection metrics is likely due to an imbalance in the training data, where approximately 70% of the images are in rainy- or overcast weather. Consequently, the tracking-by-detection algorithm SORT achieved higher MOT metrics in overcast conditions, where the average MOTA score in sunny- and overcast conditions were 58.3% and 83.1% respectively. Despite the significant difference in the MOTA score, the tracks are generally mostly tracked or partially tracked, with respective average- MT and PT scores of 45.5% and 50.0%. The tracked containers are triangulated and modeled using spherical models, where the proposed asset tracking metrics are utilized to evaluate the performance of the pipeline. The pipeline achieves a final average LA, LAGT, IDA, and DM scores of 80%, 75%, 50%, and 1, respectively, where the LAGT score was better in sunny conditions. This was determined to be a consequence of poor ground truth detections, which can be addressed by improving- the proposed metrics or the data set. Hopefully, these initial results will pave the way for improved- and new visual asset tracking solutions.

5.1 Future work

This section discusses some elements of this thesis that can be improved in future work. First, improvements of the data set are discussed, followed by discussions on how the tracking and reconstruction can be upgraded to increase the pipeline performance. The last subsection discusses improving the constructed asset tracking metrics.

Data set

The gathered data set utilized to evaluate the performance of the pipeline can be improved in multiple ways. Below, four essential elements are discussed. The first improvement is the lack of adversity of weather conditions in the data set. Interesting extensions would have included snowy and rainy conditions to the data set, which are known to introduce several challenges for vision systems. Apart from introducing different brightness and contrast, these conditions will result in moving occlusions in the form of raindrops and snowflakes visible in the camera frames.

The second improvement is ensuring high-quality ground truths for the container locations. Only two out of the five containers in the data sets had ground truths due to the poor performance of the AprilTag detector. A proposal to address this issue is to manually walk the robot through the scene while logging and ensuring that all tags are detected. As long as the same Spot mission is uploaded to the robot, the same coordinate system will be used to gather new data for the pipeline.

The third improvement is to perform a camera calibration during the data gathering. This will improve the reconstruction accuracy and the AprilTag detection performance. Additionally, a calibration in different weather will most likely result in better performance during inspections where the current calibration is unknown.

The final improvement is to include data sets where the containers are closely placed. Due to the lack of closely placed containers in the current data set, it is impossible to determine the model's merging behavior in such scenarios. The current configuration will likely merge closely placed containers, which is undesirable.

Tracking

The tracking in this work was limited to tracking the containers separately in each camera. This resulted in duplicate models for the containers in the scene, which had to be merged. By extending the system to sustain track identities across the five ring cameras, duplicate models will likely be reduced, and the DM metric will be improved. It turns out that this is a common problem in video surveillance, and approaches such as [39] and [82] appears promising for tracking in non-overlapping cameras.

One can also improve the tracking in each view by utilizing more eminent open-sourced trackers such as DeepSort [88], or TransCenter [89]. The former is a well-known supervised tracker commonly used in MOT, but like TransCenter, it requires a large-scale re-identification data set for the object to track. Thus a customized data set has to be created to utilize these trackers.

Identification

The identification of containers in this work was limited to the association of detected AprilTag's to container tracks. An interesting extension would be to apply optical character recognition to the interior of container bounding boxes for the potential extraction of the container number. This standardized container number uniquely identifies containers and is typically printed on the front of the containers. In fig. 1.1, the red container is identified with the container number *PGRU 232371 0*, while the blue container number is covered and thus not possible to extract. Even though optical character recognition introduces additional challenges such as covered or partially covered container numbers, the quality of the identification of containers will likely improve.

Visual asset tracking metrics

The visual asset tracking metrics LA, LAGT, IDA, and DM, were proposed in this thesis due to the absence of existing visual asset tracking metrics. The metrics were constructed as simple as possible and are by no means perfect compared to the object detection and MOT metrics which have been thoroughly researched. Thus further research is required to develop a high-quality set of metrics to measure the performance of new visual asset tracking solutions.

Reconstruction

The reconstruction of container point clouds resulted in relatively high reprojection errors. The accuracy of the reconstruction is an essential element in achieving accurate models that can reduce the number of duplicate models. A way to improve the reconstruction is to replace the SOBA with a full BA. This full BA will optimize both the camera extrinsic- and intrinsic parameters and the scene structure. Additionally, extending the optimization to optimize across as many views as possible makes the effective baseline larger and the resulting reconstruction better. However, this requires the features to be matched across multiple views, which is a challenging problem. This approach is what Photogrammetry does to create photo-realistic 3D models of a scene from images. Thus, creating a customized photogrammetry pipeline might be a good direction for the continuation of this work.

Appendix A

Complete container detector performance tables

Table A.1: Container detection precision metrics

Weather	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Sunny	0.632	0.922	0.741	-	0.149	0.660
Overcast	0.672	0.982	0.755	-	0.383	0.680

Table A.2: Container detection recall metrics

Weather	AR_1	AR_{10}	AR	AR_S	AR_M	AR_L
Sunny	0.419	0.684	0.706	-	0.306	0.730
Overcast	0.535	0.727	0.735	-	0.562	0.741

Bibliography

[1] Abhishek Patil et al. “BlueBot: asset tracking via robotic location crawling.” In: *ICPS ’05. Proceedings. International Conference on Pervasive Services, 2005.* 2005, pp. 117–126.

[2] Sameer Agarwal et al. “Building Rome in a day.” In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 72–79.

[3] Fred Attneave. “Some informational aspects of visual perception.” In: *Psychological review* 61 3 (1954), pp. 183–93.

[4] Adam Baumberg. “Reliable feature matching across widely separated views.” In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662).* Vol. 1. IEEE. 2000, pp. 774–781.

[5] Herbert Bay et al. “Speeded-Up Robust Features (SURF).” In: *Computer Vision and Image Understanding* 110.3 (2008). Similarity Matching in Computer Vision and Multimedia, pp. 346 –359. URL: http://www.sciencedirect.com/science/article/pii/S1077314207001555.

[6] Alex Bewley et al. “Simple online and realtime tracking.” In: *2016 IEEE International Conference on Image Processing (ICIP)* (2016). URL: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[8] Amanda Bouman et al. “Autonomous Spot: Long-Range Autonomous Exploration of Extreme Environments with Legged Locomotion.” In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 2518–2525.

[9] Michael D. Breitenstein et al. “Robust tracking-by-detection using a detector confidence particle filter.” In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 1515–1522.

[10] P. Brown et al. “2E-3 Asset Tracking on the International Space Station Using Global SAW Tag RFID Technology.” In: *2007 IEEE Ultrasonics Symposium Proceedings*. 2007, pp. 72–75.

[11] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features." In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792.

[12] G. Chen, H. Zhou, and J. Yan. "A Novel Method for Moving Object Detection in Foggy Day." In: *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*. Vol. 2. 2007, pp. 53–58.

[13] Yu Chen, Yisong Chen, and Guoping Wang. *Bundle Adjustment Revisited*. 2019. arXiv: 1912.03858 [cs.CV].

[14] Jifeng Dai et al. *R-FCN: Object Detection via Region-based Fully Convolutional Networks*. 2016. arXiv: 1605.06409 [cs.CV].

[15] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)* 2 (June 2005).

[16] Terrance DeVries and Graham Taylor. "Improved Regularization of Convolutional Neural Networks with Cutout." In: (Aug. 2017).

[17] Pierre Duthon et al. "Methodology Used to Evaluate Computer Vision Algorithms in Adverse Weather Conditions." In: *Transportation Research Procedia* 14 (2016). Transport Research Arena TRA2016, pp. 2178 –2187. URL: http://www.sciencedirect.com/science/article/pii/S2352146516302368.

[18] M. A. Fischler and R. A. Elschlager. "The Representation and Matching of Pictorial Structures." In: *IEEE Transactions on Computers* C-22.1 (1973), pp. 67–92.

[19] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography." In: *Commun. ACM* 24.6 (June 1981), 381–395. URL: https://doi.org/10.1145/358669.358692.

[20] Ruth Fong and Andrea Vedaldi. *Occlusions for Effective Data Augmentation in Image Classification*. 2019. arXiv: 1910.10651 [cs.CV].

[21] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[22] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. arXiv: 1311.2524 [cs.CV].

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[24] Banglei Guan et al. "Self-calibration approach to stereo cameras with radial distortion based on epipolar constraint." In: *Appl. Opt.* 58.31 (2019), pp. 8511–8521. URL: http://www.osapublishing.org/ao/abstract.cfm?URI=ao-58-31-8511.

[25] Alon Halevy, Peter Norvig, and Fernando Pereira. “The Unreasonable Effectiveness of Data.” In: *IEEE Intelligent Systems* 24 (2009), pp. 8–12. URL: http://www.computer.org/portal/cms_docs_intelligent/intelligent/homepage/2009/x2exp.pdf.

[26] Chris Harris, Mike Stephens, et al. “A combined corner and edge detector.” In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[27] R.I. Hartley. “In defense of the eight-point algorithm.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.6 (1997), pp. 580–593.

[28] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. USA: Cambridge University Press, 2003.

[29] Jan Hartmann, Jan Helge Klüssendorff, and Erik Maehle. “A comparison of feature descriptors for visual SLAM.” In: *2013 European Conference on Mobile Robots*. 2013, pp. 56–61.

[30] Sinan Hasirlioglu and Andreas Riener. “Challenges in Object Detection Under Rainy Weather Conditions.” In: *Intelligent Transport Systems, From Research and Development to the Market Uptake*. Ed. by Joao Carlos Ferreira, Ana Lúcia Martins, and Vitor Monteiro. Cham: Springer International Publishing, 2019, pp. 53–65.

[31] Kaiming He et al. *Mask R-CNN*. 2017. arXiv: 1703.06870 [cs.CV].

[32] J. Heikkila and O. Silven. “A four-step camera calibration procedure with implicit image correction.” In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1997, pp. 1106–1112.

[33] Dirk Helbing and Péter Molnár. “Social force model for pedestrian dynamics.” In: *Physical Review E* 51.5 (1995), 4282–4286. URL: http://dx.doi.org/10.1103/PhysRevE.51.4282.

[34] Y. Hirohashi et al. “Removal of Image Obstacles for Vehicle-mounted Surrounding Monitoring Cameras by Real-time Video Inpainting.” In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, pp. 857–866.

[35] Mazin Hnewa and Hayder Radha. *Object Detection under Rainy Conditions for Autonomous Vehicles*. 2020. arXiv: 2006.16471 [cs.CV].

[36] Marco Hutter et al. “ANYmal - a highly mobile and dynamic quadrupedal robot.” In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 38–44.

[37] Marco Hutter et al. "Towards a Generic Solution for Inspection of Industrial Sites." en. In: (2017-09). 11th Conference on Field and Service Robotics (FSR) 2017; Conference Location: Zurich, Switzerland; Conference Date: September 12-15, 2017.

[38] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[39] Omar Javed et al. "Tracking Across Multiple Cameras With Disjoint Views." In: vol. 2. Nov. 2003, 952–957 vol.2.

[40] Z. Jia et al. "A two-step approach to see-through bad weather for surveillance video quality enhancement." In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 5309–5314.

[41] Luo Juan and Oubong Gwun. "A comparison of sift, pca-sift and surf." In: *International Journal of Image Processing (IJIP)* 3.4 (2009), pp. 143–152.

[42] Kevin Kaldvansvik. "Spot, a mobile four-legged visual asset tracking robot." In: (2020).

[43] C. Kao et al. "A hybrid indoor positioning for asset tracking using Bluetooth low energy and Wi-Fi." In: *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*. 2017, pp. 63–64.

[44] Alex Krizhevsky. *One weird trick for parallelizing convolutional neural networks*. 2014. arXiv: 1404.5997 [cs.NE].

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Neural Information Processing Systems* 25 (Jan. 2012).

[46] M. Krišto, M. Ivasic-Kos, and M. Pobar. "Thermal Object Detection in Difficult Weather Conditions Using YOLO." In: *IEEE Access* 8 (2020), pp. 125459–125476.

[47] Harold W Kuhn. "The Hungarian method for the assignment problem." In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[48] Laura Leal-Taixé et al. *MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking*. 2015. arXiv: 1504.01942 [cs.CV].

[49] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database." In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[50] Liu Li, Ouyang Wanli, and Wang Xiaogang. "Deep Learning." In: (2019). https://doi.org/10.1007/s11263-019-01247-4.

[51] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[52] H. Liu et al. "Survey of Wireless Indoor Positioning Techniques and Systems." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (2007), pp. 1067–1080.

[53] Li Liu et al. *Deep Learning for Generic Object Detection: A Survey.* 2019. arXiv: 1809.02165 [cs.CV].

[54] Wei Liu et al. "SSD: Single Shot MultiBox Detector." In: *Lecture Notes in Computer Science* (2016), 21–37. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[55] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts.* 2017. arXiv: 1608.03983 [cs.LG].

[56] D. G. Lowe. "Object recognition from local scale-invariant features." In: *Proceedings of the Seventh IEEE International Conference on Computer Vision.* Vol. 2. 1999, 1150–1157 vol.2.

[57] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints." In: *Int. J. Comput. Vision* 60.2 (2004), 91–110. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[58] Wenhan Luo et al. *Multiple Object Tracking: A Literature Review.* 2017. arXiv: 1409.7618 [cs.CV].

[59] Yi Ma et al. *An Invitation to 3-D Vision: From Images to Geometric Models.* SpringerVerlag, 2003.

[60] Anton Milan, Konrad Schindler, and Stefan Roth. "Detection- and Trajectory-Level Exclusion in Multiple Object Tracking." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition.* 2013, pp. 3682–3689.

[61] S. G. Narasimhan and S. K. Nayar. "Contrast restoration of weather degraded images." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.6 (2003), pp. 713–724.

[62] J. C. Nelson et al. "Locating and tracking data center assets using active RFID tags and a mobile robot." In: *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT).* 2013, pp. 1–6.

[63] John C. Nelson et al. "Data Center Asset Tracking Using a Mobile Robot." In: SIGMETRICS '13 (2013), 339–340. URL: https://doi.org/10.1145/2465529.2466584.

[64] Farzan Erlik Nowruzi et al. *How much real data do we actually need: Analyzing object detection performance using synthetic and real data.* 2019. arXiv: 1907.07061 [cs.CV].

[65] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system." In: *2011 IEEE International Conference on Robotics and Automation.* IEEE. 2011, pp. 3400–3407.

[66] Asil Oztekin et al. “An RFID network design methodology for asset tracking in healthcare.” In: *Decision Support Systems* 49.1 (2010), pp. 100 –109. URL: http://www.sciencedirect.com/science/article/pii/S0167923610000205.

[67] Rania Rebai Boukhriss, Emna Fendri, and Mohamed Hammami. “Moving object detection under different weather conditions using full-spectrum light sources.” In: *Pattern Recognition Letters* 129 (2020), pp. 205 –212. URL: http://www.sciencedirect.com/science/article/pii/S0167865519303186.

[68] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[69] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[70] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[71] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: 1506.01497 [cs.CV].

[72] E. Rublee et al. “ORB: An efficient alternative to SIFT or SURF.” In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571.

[73] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[74] Pierre Sermanet et al. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2013. arXiv: 1312.6229 [cs.CV].

[75] Jianbo Shi and Tomasi. “Good features to track.” In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600.

[76] Connor Shorten and T. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning.” In: *Journal of Big Data* 6 (2019), pp. 1–48.

[77] Bing Shuai et al. *Multi-Object Tracking with Siamese Track-RCNN*. 2020. arXiv: 2004.07786 [cs.CV].

[78] Krishna Kumar Singh and Yong Jae Lee. *Hide-and-Seek: Forcing a Network to be Meticulous for Weakly-supervised Object and Action Localization*. 2017. arXiv: 1704.04232 [cs.CV].

[79] Daisuke Sugimura et al. "Using individuality to track individuals: Clustering individual trajectories in crowds using local appearance and frequency trait." In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 1467–1474.

[80] Richard Szeliski. *Computer vision algorithms and applications*. 2011. URL: http://dx.doi.org/10.1007/978-1-84882-935-0.

[81] Mingxing Tan, Ruoming Pang, and Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*. 2019. arXiv: 1911.09070 [cs.CV].

[82] Yonatan Tariku Tesfaye et al. "Multi-Target Tracking in Multiple Non-Overlapping Cameras using Constrained Dominant Sets." In: *CoRR* abs/1706.06196 (2017). arXiv: 1706.06196. URL: http://arxiv.org/abs/1706.06196.

[83] Tinne Tuytelaars and Krystian Mikolajczyk. *Local invariant feature detectors: a survey*. Now Publishers Inc, 2008.

[84] Athanasios Voulodimos et al. "Deep Learning for Computer Vision: A Brief Review." In: *Computational Intelligence and Neuroscience* 2018 (Feb. 2018), pp. 1–13.

[85] J. Wang and E. Olson. "AprilTag 2: Efficient and robust fiducial detection." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4193–4198.

[86] Zhongdao Wang et al. *Towards Real-Time Multi-Object Tracking*. 2020. arXiv: 1909.12605 [cs.CV].

[87] Greg Welch, Gary Bishop, et al. "An introduction to the Kalman filter." In: (1995).

[88] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple Online and Realtime Tracking with a Deep Association Metric." In: *CoRR* abs/1703.07402 (2017). arXiv: 1703.07402. URL: http://arxiv.org/abs/1703.07402.

[89] Yihong Xu et al. *TransCenter: Transformers with Dense Queries for Multiple-Object Tracking*. 2021. arXiv: 2103.15145 [cs.CV].

[90] Qian Yu, Gerard Medioni, and Isaac Cohen. "Multiple Target Tracking Using Spatio-Temporal Markov Chain Monte Carlo Data Association." In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8.

[91] Yifu Zhang et al. "FairMOT: On the Fairness of Detection and Re-identification in Multiple Object Tracking." In: *International Journal of Computer Vision* 129.11 (2021), 3069–3087. URL: http://dx.doi.org/10.1007/s11263-021-01513-4.

[92] Z. Zhang. “A flexible new technique for camera calibration.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334.