TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Marek Lahk 221314IAPM

WAREHOUSE EQUIPMENT INSTANCE SEGMENTER TRAINING FOR DIGITAL TWIN USING SYNTHETIC DATA AND FINE TUNING

Master's Thesis

Supervisor: Juhan-Peep Ernits
PhD

Tallinn 2024

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Marek Lahk

24.05.2024

Abstract

Automated computer vision based pallet tracking at an affordable price for small and medium-sized businesses remains a challenging research problem. Currently widely used systems require a large investment that may not be feasible for businesses. A camera-based pallet tracking system is being developed address this problem.

The basis of such a tracking system is a segmentation model that can accurately identify and label forklifts, stackers and pallet jacks in real-world environments. The current work focuses on training and testing a segmenter for such a tracker.

To mitigate the problem of availability of data and low variety in it, we use synthetic data in addition to real data in the model training process. Through various experiments, we evaluate the impact of synthetic data on model performance for YOLOv8, EVA-02 and Mask DINO instance segmentation models. Based on experiments we found which combinations of mixing real and synthetic data yielded best results among the experiments we carried out with training the segmentation models.

For creating the synthetic data to train our models, we devise an NVidia Omniverse based system for automatic data generation. During this process, we will compare different solutions for generating data and what the process of generating synthetic data looks like.

As a result of the work, we have a large corpus of synthetic data and a small corpus of manually labelled real data. Evaluating the models trained on them, we found the optimal combination out of the combinations we tested to train new models. In addition, we identified that from among the models we tested, Mask DINO performed best in instance segmentation for our task.

These results help us take a significant step closer to developing an automated computer vision based pallet tracking system. In addition, interested parties can use the synthetic data we created for their own projects.

The thesis is written in English and is 53 pages long, including 8 chapters, 7 figures and 11 tables.

Annotatsioon

Laotehnika segmenteerija treenimine sünteetiliste andmete ja peenhäälestuse abil digitaalse kaksiku jaoks

Automaatne aluste liikumise jälgimine väikeste ja keskmise suurusega ettevõtetele taskukohase hinnaga on senini suuresti lahendamata probleem. Praegused laialdaselt levinud süsteemid nõuavad suurt investeeringut, mis ei pruugi ettevõtetele võimalik olla. Sellele probleemile lahenduseks on arendamisel kaamera põhine aluste jälgimise süsteem.

Sellise jälgimise süsteemi alustalaks on segmenteerimise mudel, mis suudab täpselt eristada ja märgendada kahveltõstukeid, virnastajaid ja kahvelkärusid reaalses keskkonnas. Käimas olev töö keskendub sellisele jälgijale mõeldud segmenteerija treenimise ja katsetamisega.

Andmete kättesaadavuse ja nende vähese mitmekesisuse probleemi leevendamiseks kasutame treenimisel lisaks reaalsetele andmetele ka sünteetilisi. Läbi erinevate eksperimentide hindame sünteetiliste andmete mõju YOLOv8, EVA-02 ja Mask DINO põhiste mudelite headusele. Katsete põhjal leidsime, millised kombinatsioonid reaalsetest ja sünteetilistest andmedest, meie katsete hulgast, andsid kõige paremad tulemused.

Sünteetiliste treening andmete loomiseks kasutame NVidia Omniverse'l põhinevat süsteemi. Lisaks teeme ülevaate andmete loomise protsessist ning alternatiivsetest rakendustest.

Töö tulemusena on meil suur sünteetiliste andmete kogu ning väike käsitsi märgendatud reaalsete andmete kogu. Nende peal treenitud mudeleid hinnates leidsime meie testitud lahenduste hulgast optimaalse uute mudelite treenimiseks. Lisaks sellele leidsime, et Mask DINO põhine mudel saavutas meie testidel parima tulemuse.

Need tulemused aitavad sammu võrra lähemale kaamerate põhise aluste jälgimise süsteemi loomisele. Lisaks saavad asja huvilised kasutada meie loodud sünteetilisi andmeid oma projektide tarbeks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 53 leheküljel, 8 peatükki, 7 joonist,

11 tabelit.

List of Abbreviations and Terms

RF	Radio Frequency
RFID	Radio Frequency Identification
IoT	Internet of Things
GNSS	Global Navigation Satellite System
MCHT	Multi Camera Hybrid Tracer
API	Application Programming Interface
SDK	Software Development Kit
COCO	Common Objects In Context
AP	Average Precision
AR	Average Recall
IoU	Intersection over Union
RAM	Random Access Memory
VRAM	Video RAM
CPU	Central Processing Unit
GPU	Graphical Processing Unit

Table of Contents

List of Figures

List of Tables

1. Introduction

Digitalization and application of robots and AI are technologies that are actively researched in the manufacturing industries to increase productivity and optimize bottlenecks. The current thesis focuses on technology that allows to reduce time delays of workers caused by searching for appropriate inventory in intermediate storage areas in a manufacturing company. Since all inventory in the particular manufacturing company is placed on pallets, they would benefit from the digital twin of the storage area where the precise locations of all pallets would be digitally available. If that were the case the workers who need to fetch some inventory from the storage area could look up the identifier of the shelf in the information system and proceed directly to the location where the items are stowed.

The overall problem was identified in [1] and an initial solution was provided, but the tracking accuracy achieved was not yet sufficient for real world deployment. On the other hand, a clear problem and initial steps towards a solution are provided which we chose to develop further in collaboration with a manufacturing company. The goal of the current thesis is to focus on key problems that would be necessary to solve to achieve an industrially applicable visual inventory tracking solution tailored to the particular needs for pallet tracking.

The work described in the current thesis was done as part of a team of 5 people. Different sub problems in the project were solved by different team members. While we initially worked with the object detector developed in [1], we soon discovered that e.g. stackers were not detected at all with the previous solution, because they were not used back in 2020 in the company. No matter how much effort we invest into improving filtering, the actual quality of the whole system depends on the assumption that we are able to localize warehouse assets with reasonable quality. Thus, the quality of the object detection solution (it does not have to strictly be object detection) is of key importance in making the tracking solution work.

A brief overview of the overall tracking solution developed in the project is given in Chapter 2. Basically, we combine information from multiple cameras, rely on the calibration information of the cameras obtained automatically, to perform tracking and filtering of areas in the scene which involve optical flow, which in our setup works as motion detector.

It was identified that it is critical to have a reasonable detection model to be able to detect

warehouse assets: stackers, pallet jacks and forklifts. We do not attempt to provide a generic pallet object detector, because the shape of goods on a pallet varies from sheets of metal to ready made products in packaging just within this single company. Instead, we set out to train a detector which is able to distinguish full and empty assets, i.e. whether they are carrying a pallet or not.

As the solution described in [1] used object detection, we noticed that it may be quite hard to identify the actual pallet from the bounding box of e.g. a forklift carrying a pallet. Thus, it was chosen to train a segmentation model to provide more precise detection of assets and their statuses.

But training a custom model requires annotated data. We had just a few hours worth of security videos from the storage area of the production company and it seems neither to produce enough variety nor enough data instances to train a reasonable model.

The literature proposes using synthetic data, e.g. in metal defect detection [2] where fine tuning boost of 5-10% is achieved by incorporating synthetic data. Another recent result in combining synthetic and real data for fine-tuning was shown in [3]. The authors first generated a large dataset of synthetic data. Then they gathered a small amount of real world data. In the experiment they trained 3 models: synthetic data only, real world data only, synthetic base model fine tuned on real world data. The results showed up to 6.4 % increase in mask average precision compared to real world data only model. In [4] the setting is similar to ours: the authors set out to investigate how synthetic data can be utilized to improve object detection and segmentation models in the context of production systems. The conclusion therein is quite vague. They state that the results suggest "that it has the potential of becoming a valuable tool".

We set out to seek answers for the following research questions:

RQ1 How can we produce a warehouse asset and asset status detection model that can be customized for a specific situation with limited amount of annotated real-life data?
RQ2 How can synthetic data be used to mitigate the scarcity of annotated real life data and the limited diversity of it?

1.1 Research design

In this section we will explain how we plan to accomplish the task at hand. We will go over the steps necessary, the tests that need to be ran and how we are going to evaluate our results.

First we will tackle the aspect of synthetic data. We will evaluate options for software to generate it and decide on the most suitable one. Then we need to construct an environment that closely matches real world warehouse environments. Lastly we will setup a way to generate the synthetic data we require.

After synthetic data we will turn our experiments. The first hypothesis we need to evaluate is that synthetic data actually improves our models performance. Secondly, we will look into optimal ration of real and synthetic data in a dataset.

After that we will begin testing on our candidate segmenters. We will look at each ones performance on various datasets and training methods. For example, if at first training a base model on synthetic data and the fine tuning it helps our model performance. We will gather different metrics to evaluate the models performance on real world data. In addition to model goodness metrics, we will also look at the performance of each because we cannot choose a segmenter with unreasonable hardware requirements.

All of the models will be evaluated and the best one chosen for our task considering their metrics and requirements.

1.2 Alternative tracking methods

Before we started developing our own system, we looked at alternative solutions available on the market and assessed pros and cons. The systems available on the market for tracking pallets rely on a few different techniques:

- Visual codes – barcodes, QR codes, etc, where a camera or a specialized scanner is used to read the data encoded in the tags[1].
- Radio frequency (RF) tags. An RF scanner sends out a pulse to which the tags either respond by transmitting data stored in them or modulating the energy received from the scanner[2].
- IoT GNSS trackers attached to pallets[3]
- Tracking the movement of forklifts[4]

Visual codes and their scanners are cheap[5]. The codes are accurately read by scanners if

[1] `https://www.magestore.com/blog/what-is-a-barcode/` Accessed 19.04.2024
[2] `https://www.atlasrfidstore.com/rfid-resources/rfid-beginners-guide/` Accessed 19.04.2024
[3] `https://www.digitalmatter.com/applications/gps-pallet-tracking/` Accessed 19.04.2024
[4] `https://forknav.eu/` Accessed 19.04.2024
[5] `https://www.infopluscommerce.com/blog/rf-vs-barcode-scanning`

tags are printed correctly. But the scanner needs a line of sight with the tag to scan it. In addition, tags can get damaged or dirty and they may need to be replaced, wasting time.

RF tags can be scanned without a line of sight and at a distance[6]. They can be placed in a more protected location to avoid damage. Also, multiple tags can be scanned at the same time which can save time[6]. But this can also create problems – the scanner may not be able to distinguish which tag the user wanted to scan[5]. Multiple scanners scanning at the same time can also cause problems with ambiguity during data collection. Scanning can also be hampered by metallic surfaces absorbing and reflecting signals[2]. Lastly, the cost of tags can be orders of magnitude higher than barcodes[2].

GNSS (Global Navigation Satellite System)[7] based trackers offer long range tracking and decent accuracy outdoors but struggle indoors because of the surrounding building absorbing the signals. This makes them good for in-transit tracking where we do not need to know the exact location. Some trackers also incorporate other means of tracking for indoor situations, for example 5G networks. All this technology makes these trackers quite expensive.

There are also companies[4] who offer similar solutions as ours where they track forklifts. Because they do not reveal their techniques on their website we asked them directly but they were very tight-lipped about this topic. The only thing we were able to gather was that they used GNSS based tracking for outside situations and cameras for indoors.

In conclusion, while existing tracking methods offer some advantages, they also come with limitations such as the need for line-of-sight, potential damage to tags, and high costs. Our camera-based approach with instance segmentation offers a viable alternative. It is cost-effective, minimally disruptive to existing workflows, and does not require any modifications to pallets or warehouse equipment.

Accessed 19.04.2024

[6] https://www.bradyid.com/intelligent-manufacturing/rfid-vs-barcode
Accessed 19.04.2024

[7] https://www.euspa.europa.eu/european-space/eu-space-programme/what-gnss
Accessed 19.04.2024

2. Background

2.1 Multi Camera Hybrid Tracker

The tracking solution developed within the project is called Multi Camera Hybrid Tracker (MCHT). The overall goal is to develop a cost effective, camera-based system for tracking the movement of pallets in a warehouse. We want create a pallet tracking system that does not require any markings or markers on pallets or warehouse equipment and is minimally disruptive to deploy.

We opt to use only security cameras for tracking pallet movement across a warehouse. This will eliminate the need for any kind of physical tracking devices or stickers. Attaching any kind of tracker on a pallet takes time, they can break or be damaged and also makes the system only compatible with pallets that have a tracker. Also, in many cases cameras needed for this have already been installed which reduces the upfront cost and allows for a faster roll-out of the system.

One of the most important questions is how to track the movement of a pallet? The problem is challenging for object detection approach because while empty pallets could be recognized by an object detection solution, pallets with inventory come in a variety of shapes, sizes, and colours where the actual pallet itself is minimally visible. E.g. the inventory on the pallet can be uniform boxes, random components, large sheets of metal, etc.

We make the same assumption as was made in [1] that no pallet will move on their own, so there has to be a pallet moving asset (a pallet jack, stacker, forklift in our case) involved. Such approach simplifies the task to recognizing pallet movers and determining their state (whether carrying a pallet or not) and tracking the position of the asset over time. By combining the position and carrying status of an asset we can reconstruct the path of the pallet. For example, if a full forklift drove to position A, then became empty, we can assume that it left the pallet at location A. If another empty forklift gets full at position A, we can assume that it picked up the same pallet.

Our full tracking pipeline is described as follows:

1. Optical flow - Find if and where movement took place
2. Segmentation - Locate assets precisely
3. Projection - Estimate the location of an asset on the floor
4. Tracking
   (a) Camera combining - Filter and combine data from multiple cameras to get current position for each asset
   (b) Association - Associate items from previous frame with current ones.
   (c) Uncertainty computation - Decrease or increase uncertainty based on if information from cameras is similar or not

Our system uses multiple cameras to track a single object we need precise intrinsic (focal point, principal point, distortion) and extrinsic (position, rotation) camera parameters. Knowing these, we can map a pixel on one image to a pixel on another. Lack of precise camera parameters was one of the problems brought out in [1], which caused tracking inaccuracies. To acquire camera parameters, we use a Structure-from-Motion (SfM) [5] and Multi-View Stereo (MVS) [6] pipeline called Colmap[1]. This pipeline uses multiple images taken of an object or room to create a 3D point cloud and estimations of the relative position of the cameras, where the images were taken from. We use those positions to combine the data from different camera tracking feeds. In addition to the position, we can also use the produced point cloud to visualize our tracking results in a 3D environment.

This thesis focuses on one part of the tracking system - the instance segmenter[2]. The segmenter is an integral part of our system, allowing us to detect objects of interest and their exact shape. This is one of the major improvements compared to [1], which we hope will give us better results. In the paper they used bounding-box-based object detection[3] which gives a box around an object of interest. The box often covers more area than the actual object we are interested in, leading to inaccuracies.

Another area of concern we wanted to address was limitations to publishing the dataset used for training. In [1] they used only real-world material for training and because it contained intellectual property of others, they could not publish it. Also, real world data needs to manually be annotated which can be time consuming and costly so we want to use as little of it as possible. As a solution we will be building a digital twin of an actual warehouse and gathering synthetic training data which we will use to train a base model.

[1] `https://colmap.github.io/` Accessed 19.04.2024
[2] `https://encord.com/blog/instance-segmentation-guide-computer-vision/` Accessed 19.04.2024
[3] `https://blog.roboflow.com/object-detection/` Accessed 19.04.2024

The base model will be fine-tuned[4] on real world data for optimal performance. This allows us to publish our synthetic data and base model which everyone can fine-tune on their own data, and still keep intellectual property safe. Also, in case there is a change in the warehouse that would require a new model (warehouse moving to a new location, new piece of equipment, etc), we should be able to train one much more quickly.

[4] https://www.ibm.com/topics/fine-tuning

3. Synthetic Data for Industrial Environments

3.1 Choice of Simulation Environment

To generate synthetic data for our segmenter training we considered several options like ROS Gazebo, as was used for synthetic data generation in [4], Blender, NVIDIA Omniverse [7]. While Gazebo and Blender are featureful open source tools suitable for generating simulation environments, there are 2 the main functionalities that we seeked: the availability of relevant asset models and toolchain to create annotated data.

We opted for NVIDIA Omniverse as according to their website [7] : "NVIDIA Omniverse™ is a platform of APIs, SDKs, and services that enable developers to easily integrate Universal Scene Description (OpenUSD) and RTX rendering technologies into existing software tools and simulation workflows for building AI systems."

The decision to use Omniverse stemmed from its features:

- High visual fidelity - We needed the training data to look as realistic as possible.
- Pre-made assets - Working in a small team, we do not possess the time or resources to create our own assets, e.g. pallet jacks, forklifts, shelves, pallets etc. It was crucially important that the tool we used include pre-made assets that we needed.
- Automatic segmentation - The entire idea of using a simulation environment is to save time and resources on manual labelling. Hence we needed a tool that in addition to rendering synthetic images could also add ground truth data to them. This also means that the ground truth is as precise as it can be because it was created pixel perfectly by the software.
- Built-in randomization - We did not want to create each synthetic images by hand. Instead, we needed a tool that could randomize the scene for each image automatically.

We looked at other options like Gazebo[1] and Blender[2]. Both of these options would allow us to generate high fidelity synthetic data. Randomizing of scenes would also be possible through scripting. We decided on Omniverse because of the fact that it has pre-made high quality assets.

[1] `https://gazebosim.org/home` Accessed 19.04.2024
[2] `https://www.blender.org/` Accessed 19.04.2024

3.2 Warehouse replica

To give our segmenter the best chance of working in the real world, we need to have a good approximation of the real warehouse. To create a warehouse shell we used an extension called Warehouse Creator[8]. With this extension we can create a realistic warehouse of any grid-based shape and size. We simply had to mark the perimeter of the desired shell and the extension automatically created the warehouse.

Our goal is to track pallets in a single aisle of the warehouse. To mimic the area, we modified an existing shelve asset to match the shape and size of the shelves used in the real warehouse. These were then arranged to form an aisle. For further realism, each shelf got a set of randomized pallets that resemble the real-world counterparts.

Figure 1. *Rendered image of the simulated warehouse environment.*

3.3 Replicator

We need our segmenter to be able to recognise our objects of interest from almost any angle and in different situations. For this reason, we require input data that captures this. Omniverse has an extension called Replicator which is made for generating synthetic [9]. It works by taking in a description of how to randomise a scene and produces a graph in OmniGraph.

In Figure 2 we can see a simple Replicator OmniGraph graph. Its purpose is to randomize a cameras position on every frame of the synthetic data generation. *Group* node references all the objects being randomized - in this case a single camera. *Count* node outputs the number of objects in the group. *Uniform distribution* node outputs random numbers that follow a uniform distribution. This node is configured to output a 3D coordinate in a specific range - for example [0, 2.4, 1] The *Num Samples* tells the node how many of these coordinates to output. *Per axis pose* node modifies the position to be a correct type (float3[] -> double3[]). *Write prim attribute* changes each objects attribute, in this case the position, to some value. *Values* input takes the random coordinates and maps each to an object. This

process is controlled by the *On Frame* node which sends out an execution signal on each frame which triggers all the nodes its connected to. *Read Fabric Time* and *Rational Sync Gate* nodes are not particularly important and will not be explained. This all may seem complicated but it is all generated by Replicator. The actual code that generated it is only 5 lines of Python code (See Appendix 2).

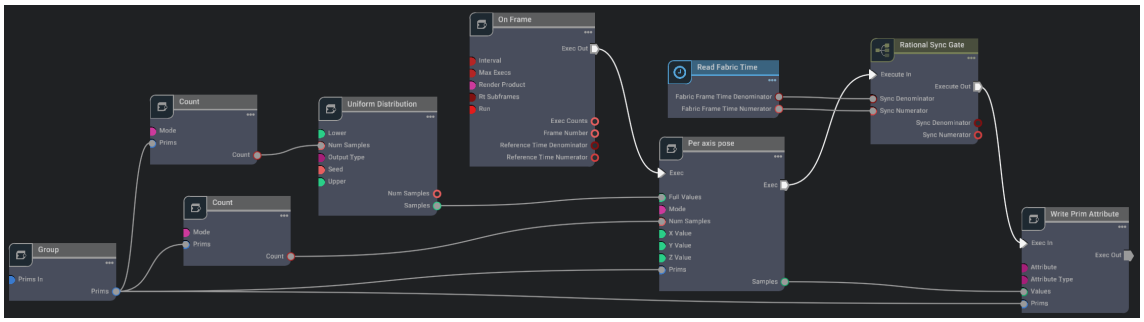


Figure 2. *Replicator graph example*

3.4 Randomized Data Creation

To provide our segmenter with as varied dataset as possible, we randomised the following:

- mover position/rotation
- pallets (shelf, movers)
- camera position/rotation
- clutter position/rotation
- mover lift height
- mover color
- mover pallets
- lighting
- humans

The randomizer starts by selecting a random sample of one to six movers. Next a random sample of ten clutter objects are spawned in. All the spawned objects get a random rotation applied and then scattered randomly in the isle. This allows us to show different combinations of objects in different positions and angles to the segmenter so that it can recognize them in any situation.

We also need to make sure that the segmenter does not learn that movers are always one color and their forks are near the floor. To avoid it we select a random color and change the fork height for each mover on each frame.

We hope that we can create an association between the presents of a pallet (and the lack

of visible forks) and the mover being assigned as full. To teach this, some movers are spawned with a pallet on its forks. This also creates a problem: if the pallet was always the same, the segmenter would try and learn the exact shape and composition of the pallet. To make it learn more generally about pallets we randomize the items on top of the pallets each frame.

Lighting conditions in warehouses can be different, both in terms of the brightness (and amount) of the artificial lights and natural light. Sunlight shining in through a window creates bright areas that the segmenter needs to be able to deal with. This is accomplished through setting the brightness of ceiling lights and altering the angle of the sunlight passing through ceiling windows.

Lastly, we noticed that in the real world there humans operating the movers often occlude them. Because of this, movers sometimes have a human operating it. These humans are only in one pose; hence they are not very useful for training it to recognize humans in general but we hope it does teach the segmenter to also look at small details behind humans for signs of movers.

Figure 3. *Replicator example*

3.5 Semantics

Replicator also provides a way to automatically output ground-truth data for rendered images (Figure 4). It supports many different types of data. The following list is not a complete list of options, only the most relevant for our task.

- Semantic segmentation
- Instance segmentation
- 2D loose bounding box

- 2D tight bounding box
- 3D bounding box

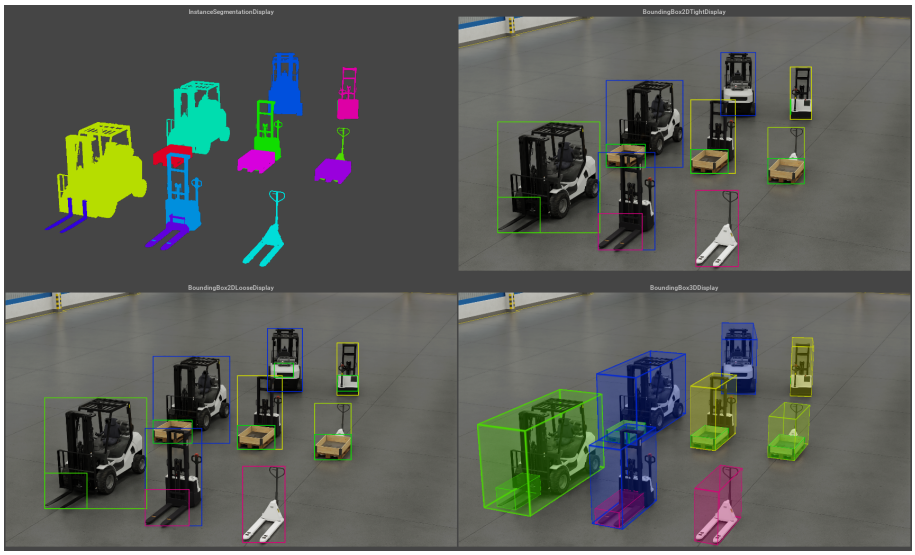


Figure 4. *Replicator semantics*

Additionally, the default semantic data writer in Replicator can be overwritten with a custom one to write in different formats or output additional data. For example, if we wanted to train a model to estimate the height of forklifts forks on an image, we could query the position of the forks on each frame and write them to a file.

3.6 Post-processing

We cannot use the raw data output by Replicator. The out is a mixture of images (RGB image, segmentation mask) and JSON files (instance segmentation category mapping, instance segmentation object mapping). This step is in the form of a script that takes this data and convert it to the correct format, YOLO format in our case.

Additionally, the script needs to alter some annotations. In Omniverse we have hard-coded the mover's occupancy status, meaning even if we cannot a pallet, we have an annotation that tells us that the mover is full. To automatically assign the unknown type to a mover we look at the segmentation mask and determine if we can see forks, in the case of empty movers, or a pallet, in the case of a full mover. If we cannot see them or the visible area is too small, we assign the unknown class.

3.7 Synthetic Data Generation Experience

We spent around 3 months learning this program and creating the environment and producing the synthetic images.

Was it worth it?

We do acknowledge that if this time was spent labelling images for our datasets we would have a massive one that in all likelihood would outperform anything that could be created with synthetic data. But most of that time was spent learning Omniverse, its many extensions and other tools we needed. Knowing what we know now, we estimate we could set up a similar data generator in about 15-20 hours. We will see the effects of using the simulated data in the model performance analysis chapter.

4. Warehouse Asset Dataset

In the current chapter we will explain the rationale in choosing which classes are labelled and how the dataset is composed.

4.1 Labels

Our dataset consists of 11 categories.

- forklift_empty
- stacker_empty
- pallet_jack_empty
- forklift_full
- stacker_full
- pallet_jack_full
- forklift
- stacker
- pallet_jack
- mover_pallet
- person

First 9 are movers in different states of occupation. *empty* suffix means that we can see that there is no cargo being carried by the mover. *full* suffix in case we can see that there is cargo. Class name without suffix is used in situations where we are unable to determine if the mover is carrying a pallet or not. This could be caused by objects blocking the view, the mover itself occluding the pallet, or part of the mover being outside of the frame. We chose to add the unknown case because it will be very useful in later parts of the MCHT pipeline. Mover pallet is assigned to pallets that are being carried by movers. The hope is that the segmenter will be able to make an association between mover pallet existing and mover type being full. The MCHT pipeline does not rely on the mover pallet to always be detected because the way the pallets look varies so much. But in case we are able to detect it, tracking becomes much simpler. Person is assigned to any human. It makes sense to label humans, because they are present in the scene when pallets are moved (the company currently does not use robotic forklifts).

4.2 Datasets

For our training process we put together two training datasets - one for base model training and one for fine-tuning. We also have a dataset for doing final model testing for performance comparison.

There are three sources of data that we used. Real data is manually annotated security camera footage from a real warehouse. Then we have synthetic data which was described in Chapter 3. Finally we took data from the COCO(Common Objects In Context) dataset[10][1]. Specifically, we filtered out images and annotations of humans. The reason for this is that we need our models to reliably detect humans and humans come in such variety, so we needed a larger sample of them.

In Table 1 we can see the compositions of our datasets. The largest set is the base set containing 14820 images in total. This set is designed to teach models what warehouse equipment might look like. This works as a foundations for the fine-tuning process. By training a already having trained a model on a large dataset we can train a situation specific model with less data and time. The dataset contains only synthetic and COCO person data. This is by design, to allow us to publish the data and base models without having to worry about the privacy of the people and protected intellectual property of the real company.

Then we have the fine-tuning dataset which consists of 1127 images. This set is a mix of real and synthetic data. Real data was split close to 70/30 which is very common datasets. Synthetic data was added to the training set after early tests showed signs of over fitting and bad performance. Subsequent tests showed signs of improvement which were later confirmed by experiments in Sections 5.2 and 6.3. The rational of how much synthetic data to add is explained in Section 5.3. We chose not to include any COCO person images because as we will see in Section 4.3, there are already examples of people.

The validation set in the fine-tuning dataset is very small - 98 images and they are all real images. Due to time constraints, we could not spend any more time annotating more images. But if we had so little real images then why did we not just add synthetic ones? At this stage of the training, we want the model to perform well in the real world. We did some early tests synthetic data in validation. This caused models keep optimizing for synthetic images which is not what we at this stage of training.

Finally, the testing set consists of 145 real world images. Both the fine-tuning and test real world data come from the same warehouse and the same set of cameras, but the data has

[1] `https://cocodataset.org/` Accessed 19.05.24

been collected at different times.

Table 1. *Dataset compositions*

Dataset	**Real**	**Synthetic**	**COCO person**	**Total**
	Base			
Train	0	9114	1300	10414
Validation	0	3906	500	4406
	fine-tuning			
Train	229	800	0	1029
Validation	98	0	0	98
	Testing			
Test	145	0	0	145

Table 2. *Dataset image resolutions*

Image type	**Resolution (pixels)**
Synthetic images	1920*1080
COCO person	300*300 - 640*640
Real images (training)	4096*2160
Real images (testing)	1920*2160

Another important aspect of our datasets is their image resolutions (Table 2). Different resolutions can influence model performance and inference times. We generated our synthetic images at 1920*1080 resolutions which is a good middle ground between generation times and image quality. Any larger and they would take considerably longer to generate, any smaller and we would start to lose detail in smaller and further away objects. Images taken from the COCO dataset were kept as is. The resolutions vary from image to image but are in the range given. We used two different resolutions for our real images. For training we used the cameras native 4096*2160 (4K) images. For testing images, we decided to cut away sides of the native image and keep a 1920 pixel wide section from the middle of the image. This was done because the cameras edges cover areas that are outside of interest.

4.3 Class distribution

To get a better overview of the distribution of different classes in our dataset, we can look at the graphs in Figure 5. Starting with the base dataset. The most numerous class by far is

person. This is because all mover objects in our synthetic dataset have a 50% chance of having a person with it. Plus, we included images from the COCO dataset that include people. We could have reduced the number of COCO images that we included but we wanted to have many humans in different situations. Next most numerous class is mover pallet. This makes sense because their amount in the dataset is the sum of all the full movers, which amounts to quite many. The last classes that stand out because of their large size are forklift and stacker. The reason for those is we used a script to post-process our synthetic data annotations, where we were looking for either the visibility of forks or a mover pallet. We had to set a threshold for the smallest size we consider visible. If they were not visible we considered the mover as unknown. Due to variable distance of movers from the camera the relative size of forks and mover pallets also changes. This caused some distant objects with clearly visible fork/pallet to be marked as unknown. In our opinion, this kind of imbalance of classes was fine for the base dataset because this sets purpose is not to create a model that we could deploy.

Because the fine-tuning set contains mostly synthetic data, all of the reasonings about synthetic data from base dataset apply here as well. Another reason for some of the imbalances in classes can be attributed to the composition of videos provided to us by the real warehouse.

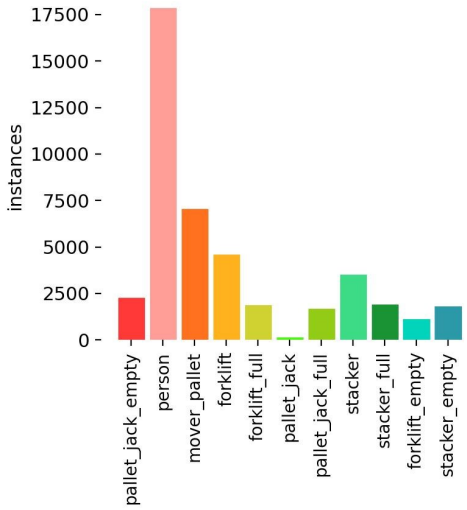
Finally for the test dataset the composition was completely dictated by videos that were provided to us.

As a final point we would like to mention the pallet_jack class. There is so little of this class because it is very rare to see a pallet jack and not the forks or pallet its carrying. The pallet jack cannot cover the load its carrying or the absence of it, as opposed to something like a forklift which is wide enough to do so.

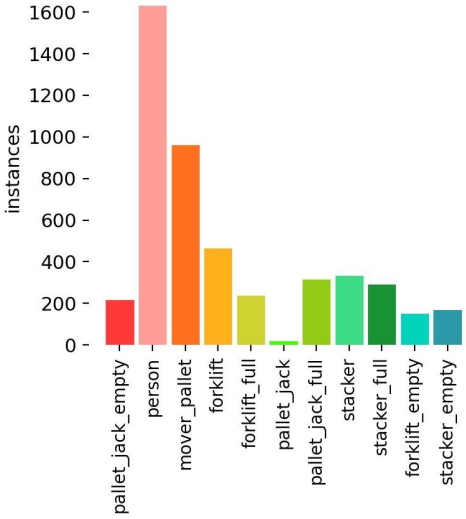
4.4 Manual annotation process

All our manual annotations were done in a program called CVAT[2] which is a tool for allowing teams of people to annotate images in parallel. It has functions creating and editing mask and polygon-based annotations. We chose this tool because of its integration with the Meta AI Research's[3] Segment Anything (SAM)[11] model. This model produces segmentation mask based on a several types of user inputs, which include points, text and boxes. In CVAT this model is used to allow for almost automatic segmentation of objects. All the user must do is click on the object they wish to segment, then refine the produced

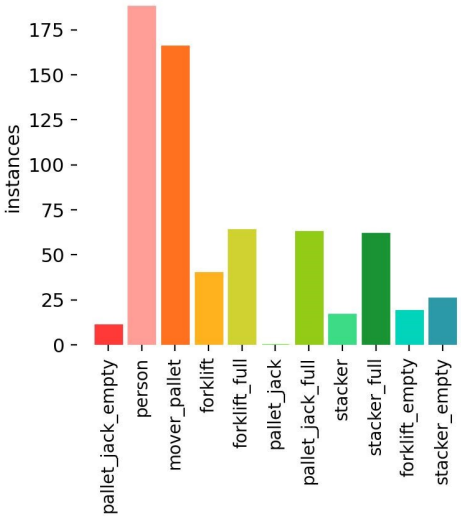
[2] `https://www.cvat.ai/` Accessed 21.05.24
[3] `https://ai.meta.com/meta-ai/` Accessed 21.05.24



(a) Base dataset class distribution



(b) fine-tuning dataset class distribution



(c) Test dataset class distribution

Figure 5. *Dataset class distributions*

segmentation with additional points if needed and they end up with a segmented object.

Before we get to how well this works, we will go over our work process. We started by uploading our videos to be processed into individual images to be segmented. When uploading we can set the desired interval at which frames are taken. This is useful because it is not worth the effort to segment every single image in video - close by frames are so similar that we can skip a few. All videos were split into jobs of ten images each. Each person segmenting assigned themselves a job that only they were working on thus avoiding problems caused by multiple people modifying a single image at once.

At this point people would start manually segmenting by drawing a polygon over the objects of interest which is a very time-consuming process. We tried to use the SAM model integration to semi automate the process. It worked well in situations where objects had well defined edges and was large enough. Unfortunately, this was not the reality in most cases. Extremely low bitrate of the security camera footage we were segmenting caused many compression artefacts and loss of details from the image which made the model quite inaccurate. The resulting segmentations would have wavy edges on straight lines, corners cut off and other errors causing us to do manual fixes which often meant correcting almost every part of the segmentation. In addition, the model took a long time to process an image - when starting work on a new image we had to wait 30 seconds before the model would respond. While this wait was only for the first segmentation of a new image, it still wasted a lot of time and was an annoyance for the people annotating. In the end, we rarely used the model, opting instead to do it manually.

We did not keep an accurate track of time spend segmenting, but we estimate that each ten image set took us around 30 minutes for a simple case and up to 1 hour for more difficult case to complete. A rough estimate for the time spent segmenting the 472 images is around 35 hours, assuming we spent 45 minutes on each set on average.

5. Experiments

To validate our hypothesis, we conducted preliminary experiments using a single instance segmentation model. We opted for YOLOv8 [12] due to its reported precision on their GitHub repository[1], high popularity (nearly 24,000 GitHub stars), and user-friendliness. As a commercial product, YOLOv8 prioritizes ease of use, making it a suitable choice for initial exploration.

All of the following experiments were performed with a 640*640 input image window size and tested on our test dataset. All models trained for these experiments use YOLOv8x-seg weights as to initialize the model. These weights are provided by the creators of YOLOv8 and come for a model that was trained on the COCO dataset. All of the models were left to train for as long as their validations showed signs of improvement.

5.1 Experiment Setup

In this section we describe the hardware that was available at the time of conducting experiments described in the thesis and what kind of metrics we use for model performance evaluation.

5.1.1 Hardware

For training we used two machines in TalTech's AI-Lab in parallel to speed up the process.

Table 3. *Hardware configuration*

Machine	**GPU**	**CPU**	**RAM**
ai-lab-03	NVIDIA RTX 6000 ADA (48 GB)	AMD Threadripper PRO 7975WX (32 cores / 64 threads)	384 GB
ai-lab-04	NVIDIA RTX 4090 (24 GB)	AMD Threadripper 3970X (32 cores / 64 threads)	128 GB

Performance wise, the node ai-lab-03 is faster than ai-lab-04. The main bottleneck in the experiments was the GPU, as most of the computations are using it. The CPU is used for

[1] `https://github.com/ultralytics/ultralytics`

job management and data loading. Most significant difference between the two available GPUs is the amount of VRAM available on the cards. The consumer oriented RTX4090 has 24 GB, while the workstation and AI computations oriented RTX6000 ADA has 48 GB. It is worth noting that both RTX6000 ADA [13] and RTX4090 [14] share the same NVidia silicon architecture AD102 code named Ada Lovelace, but there are differences in what features are enabled and how many CUDA cores are available (16384 in RTX4090 vs 18176 in RTX6000 ADA). While almost all the models were trainable on the RTX4090 machine, one of the models required more than 24 GB of VRAM thus requiring the use of the node with RTX6000 ADA. Because of the difference in training hardware, we will give rough estimates of training times in hours rather than with higher precision.

5.1.2 Evaluation metrics

For evaluation results to be comparable we need to test them all the same way. All of the predictions will be evaluated using cocoapi's [2] evaluator. This evaluator uses datasets in COCO format[3] as input and outputs different metrics. For models which output in a different format, a script will be used to convert it to COCO format.

The evaluator gives us values for precision and recall for different IoU (Intersection over Union) values[4]. We to evaluate and compare our models we will use the following metrics:

- Average precision @ IoU=50 (AP50)
- Average precision @ IoU=50-95 (AP50-95)
- Average recall @ IoU=50-95 (AR50-95)

To evaluate our segmenters performance, we use average precision (AP) with different IoU thresholds. AP50, for a 50% overlap between predicted and actual masks, gives a general idea of segmentation ability. The stricter AP50-95 focuses on more precise masks, indicating how well predictions match ground truth at a wider range of overlaps (50-95%). Comparing these APs reveals accuracy: similar values mean good overlap, while a lower AP50-95 suggests the model struggles with pinpointing objects exactly.

We also consider recall (AR50-95) to gauge how well the model finds all objects of interest. Analyzing both AP50-95 and AR50-95 together tells us a more complete story of the model: high values in both metrics indicate the model excels at both finding and precisely locating objects which is what we are aiming for.

[2] `https://github.com/cocodataset/cocoapi` Accessed 21.05.24
[3] `https://cocodataset.org/#format-data` Accessed 21.05.24
[4] `https://cocodataset.org/#detection-eval` Accessed 21.05.25

5.2 Inclusion of synthetic data

A crucial question early on was whether incorporating synthetic data would actually improve model performance. To address this, we trained two separate models: One model was trained with the full set, the other one had the synthetic data removed.

- One trained on the entire fine-tuning dataset (including synthetic data).
- Other trained on the fine-tuning dataset excluding synthetic data.

Table 4. *Synthetic data effect results*

Model	**AP50**	**AP50-95**	**AR50-95**
Real	0.101	0.070	0.094
Real + synthetic	0.268	0.172	0.218

From the results in Table 4 we can see the addition of synthetic data has a significant positive effect on models performance. All of the performance metrics were least doubled.

5.3 Balance between synthetic and real data

The amount of synthetic data in our fine tuning dataset might plays a part in how well the model will perform. To find the optimal amount we tested 8 models with varying amount of synthetic data. We chose to test a range of 200 to 3200 synthetic images. In addition, all datasets include the real world data from our fine tuning set.

Table 5. *Synthetic data concentration results*

Synthetic	**AP50**	**AP50-95**	**AR50-95**
200	0.060	0.040	0.047
600	0.140	0.091	0.107
1000	0.144	0.087	0.097
1400	0.164	0.104	0.121
2000	0.139	0.083	0.091
2400	0.210	0.136	0.168
2800	0.123	0.073	0.082
3200	0.118	0.069	0.083

At first look, the results in Table 5 show 2400 images of synthetic data as the clear winner. But at closer look we can see that a strange pattern, where the results improve in in the 200

- 1400 range, drops at 2000, reaches peek at 2400 and then drops again. We are unsure if best the results at 2400 are because the amount of synthetic data is optimal or because the model got lucky during training. When deciding the amount of synthetic data to include in our fine tuning set, we settled on 800 because according to these results, anything in the range from 600 - 2800 should give at least a decent result. This amount of was also chosen for faster training times and to not totally out number the 229 real images we had for fine tuning.

These results should be taken with a grain of salt. Results from one model may not reflect results from another. We would have loved to run this test on all of the models that we evaluated but this did not fit into the time frame of this project. While YOLO models presented here took around a few hours each to train, other segmenter models took closer to 10.

6. Model performance

We chose the following models for evaluation:

1. YOLO V8 [12]
2. EVA-02 [15]
3. Mask DINO [16]

For the rationale of picking YOLOv8 see Chapter 5.

The ranking of instance segmentation tools at Papers with Code website[1] at the time of writing the thesis is represented in Table 6. It is worth mentioning that the state of the art model in the ranking is EVA [17], but the model that we chose to test is EVA02 [15] for which the authors claim that the AP50-95 has been further improved to 55.8 from 55.5.

The FD-SwinV2-G [18] model we chose to omit because we could not find a pre-trained instance segmentation model. The Mask Frozen-DETR [19] seems to be very promising especially because the authors claim its training times are 10x less than for Mask DINO, but since there was no code available, we could not test it out with the given time budget. For BEiT-3 [20] we also did not find an instance segmentation model readily available. Also, since one of the main purposes of the model is to unify vision and language models, the focus of the approach is not very well aligned with our current task. Thus for the third model we chose Mask DINO.

For YOLO we chose to test two different window sizes to make sure that we do not

[1] `https://paperswithcode.com/sota/instance-segmentation-on-coco` Accessed 21.05.24

Table 6. *Top 5 instance segmentation tools on COCO test-dev dataset*

Model	**AP50-95**
EVA [17]	55.5
FD-SwinV2-G [18]	55.4
Mask Frozen-DETR [19]	55.3
BEiT-3 [20]	54.8
Mask DINO [16]	54.7

discriminate unfairly as both EVA-02 and Mask DINO models have larger windows (1536 and 1024 respectively).

All model training was started on a pretrained weights provided by the model authors. We picked the best performing weights that were trained on the COCO 2017 dataset.

Table 7. *Model base weights*

Model	**Weights**
YOLO V8	YOLOv8x-seg[2]
EVA-02	eva02_L_coco_seg_sys_o365[3]
Mask DINO	Mask DINO[4]

6.1 Synthetic base model performance

Our investigation into the synthetic base model fine tuning approach began with training the base models. These trainings took by far the longest due to the size of the base dataset.

We were interested to see how our base models would perform on real world data. Good results might mean that we do not need any real data, but we were not very hopeful. Also if a model was able to correctly segment real world data, being only trained on synthetic data, it would show a higher degree of generalization ability from the model.

Table 8. *Synthetic base model performance*

Model	**AP50**	**AP50-95**	**AR50-95**
YOLO (640px)	0.127	0.067	0.070
YOLO (1536px)	0.130	0.093	0.124
Mask DINO	0.392	0.282	0.445
EVA-02	0.449	0.364	0.530

From the results in Table 8 we make two conclusions. Firstly, both YOLO models are basically unable to learn anything from synthetic data that it could then apply to real world images. Secondly, the results of both Mask DINO and EVA-02 are surprisingly good. At

[2] `https://github.com/ultralytics/ultralytics?tab=readme-ov-file#models` Accessed 21.05.24

[3] `https://github.com/baaivision/EVA/tree/master/EVA-02/det#system-level-comparisons-w-o365-intermediate-fine-tuning` Accessed 21.05.24

[4] `https://github.com/IDEA-Research/MaskDINO?tab=readme-ov-file#coco-instance-segmentation-and-object-detection` Accessed 21.05.24

IoU 50, 32.2 and 44.9 percent of predictions, respectively, match ground which is quite impressive considering they have not seen any images from our real world warehouse. This would suggest that these models are able to learn and apply more general knowledge.

As impressive as these results are, they are still not good enough for us to use in our MCHT pipeline.

6.2 Fine tuned base model performance

Continuing with our test, we took all of our synthetic base models and continued training on them our fine tuning dataset.

Table 9. *Fine tuned synthetic base model performance*

Model	**AP50**	**AP50-95**	**AR50-95**
YOLO (640px)	0.294	0.195	0.230
YOLO (1536px)	0.247	0.189	0.242
Mask DINO	0.657	0.540	0.758
EVA-02	0.590	0.475	0.608

The results in Table 9 show that, as expected, results get significantly better if we perform a fine tuning with a dataset that includes real images, on a synthetic base model. We can see a large increases for all models across all metrics. Both YOLO models experienced an increase in performance of over 100% across the metrics but considering the very low results before, it should be quite easy for the model to improve. Both sizes of the model performed quite equally and poorly. Mask DINO showed an increase of $\tilde{7}0\%$ in AP50 and AR50-95. The biggest increase was in AP50-90 of 91% meaning it got significantly better at predicting accurate masks. The model that improved the least was EVA-02 31, 30 and 41 percent for the metrics. This could be due to its already best performance in the previous test - it was most difficult for it to find meaningful improvements. Overall, Mask DINO was the best performing model from this test.

6.3 Inclusion of synthetic data continuation

When testing our fine tuned synthetic base models in Sections 6.2 we noticed how close the performance was to one achieved without a synthetic base model in Section 5.2. This promoted us to run the tests described in Section 5.2 on our other models as well because up to this point we had only experimented with models that derived from our synthetic base models for models besides YOLO.

We chose not to include the 1536px YOLO model in this test because previous experiments show little to no difference between them (Tables 9, 8)

Table 10. *Synthetic data effect results 2*

Dataset	**AP50**	**AP50-95**	**AR50-95**
YOLO (640px) Real	0.101	0.070	0.094
YOLO (640px) Real + synthetic	0.268	0.172	0.218
Mask DINO Real	0.497	0.361	0.652
Mask DINO Real + synthetic	0.710	0.580	0.776
EVA-02 Real	0.469	0.363	0.503
EVA-02 Real + synthetic	0.627	0.496	0.636

The results can be seen in Table 10. For easier reference we included the YOLO 640px results from Table 4.

Starting our comparison with YOLO, we can see that even with synthetic data, it performed worse compared to the previous experiment. This might indicate that this model could benefit from a synthetic base model but because the results do not have a very large difference, we concluded that this would require further testing. Mask DINO showed a small improvement of 8, 7, 2 percent across metrics. EVA-02 showed similarly small improvements of 6, 4, 4 percent. This indicates that for Mask DINO and EVA-02 it is not worth training a synthetic base model first because not only does it not help performance but also can degrade it.

This test further proves the conclusion made in Section 5.2 that adding synthetic data significantly improves model performance.

6.4 Inference testing

Another aspect that we need to consider is how fast do these models process an image and what are its requirements in terms of hardware. Inference times were measured by feeding all of our test dataset images to the model one by one and measuring the time it takes to process. The resulting times were averaged. In terms of hardware limitations, the biggest problems arise from the amount of VRAM (Video RAM) a model requires to run on the GPU. Without sufficient VRAM the models would not run at all. VRAM requirements were gathered by taking the largest amount used as reported by nvidia-smi utility over a full run of inference on our test dataset.

For comparable results we ran all of our tests on the ai-lab-04 machine (see Table 3 for hardware).

Table 11. *Inference results*

Model	**Inference (ms)**	**VRAM usage**
YOLO (640px)	32.86	4.2 GB
YOLO (1536px)	67.41	21.6GB
Mask DINO	219.22	16.8 GB
EVA-02	383.40	4.2 GB

A large difference in inference times (Table 11) between YOLO and other models comes as no surprise. YOLO V8 a model architecture designed for fast inference and has been optimized for real time applications[5]. 32 milliseconds per frame for the 640px model is really fast. 1536px model has almost twice the input size and takes over twice as long to inference. At 67 milliseconds it is still a good result and considerably less time then the next best model. As for VRAM usage, the YOLO models acted differently from each other. 640px model reached a maximum of 4.2GB and stayed there. 1536px model used 21.6GB which is a very large increase from the smaller model. We are unable to provide a good explanation as to why it increased so much.

Looking at the other two models we can see that they require much more time to inference an image. Mask DINO with 219ms is around 6.6 times slower than 640px YOLO. In last place is EVA-02 with 383ms.

6.5 Visual model evaluation

Looking at test scores gives us a good overview of how well the models perform and how they compare to each other. This may not give us the whole picture as there could be some other interesting and noteworthy things to see. For this reason we visualized the model outputs and compared them.

Both images show here are taken from our test dataset. Order of sub images is the same on both images:

- top left - Mask DINO
- top right - EVA-02
- bottom left - unlabelled

[5] `https://docs.ultralytics.com/` Accessed 21.05.24

- bottom right - YOLO 640px

All results shown are created from inference on models described in Section 6.2

Both images Figures 6 and 7 show our models dealing with a challenging situation. Our real world training data contains very few instances where we see a forklift from the top. In addition, we have no training images from forklifts with extended masts, like we can see on the figures. The closest the models have seen are forks raised to the top of a non extended mast. Surprisingly both Mask DINO and EVA-02 are able to segment the machines very well. YOLO could not recognize the forklift. All three models on both images were able to correctly segment and classify the stacker in the fore and background along with the person and mover pallet. One difference between the models was that EVA-02 classified the background stacker as an unknown stacker, while rest of the models assigned stacker_full. We consider both of these labels as correct.

Looking closer at both sets individually, we can see on Figure 6 that Mask DINO could not segment the pallet being carried. The forklift was assigned a forklift_empty class. This comes as no surprise because our training data had mostly only standard size pallets and all pallets were carried so that the pallet was against the fork support. This most likely caused the models to label a mover as full only if no fork was visible. EVA-02 was able to recognize the pallet which is quite impressive but it struggles more with classifying the forklift. It was assigned stacker_empty. We think this could be because the extended mask of the forklift resembles the mast of a stacker.

On Figure 7 one observation we would like to point out is that all of the models struggled to identify the pallet jack, situated under the left side shelf. EVA-02 and Mask DINO were quite rarely able to identify it but usually with low confidence and incorrect class. This was the case if the pallet jack was there alone but as soon as there was a person near the models would be able to segment and classify it correctly most of the time. We do not know the exact reason behind it but suspect that the models made some associations between a person and pallet jack class.

Last thing we would like to point out is the difference between Mask DINO and EVA-02 segmentation mask quality. When looking at the edges of the forklift we can see that Mask DINO follows the contours really precisely while EVA-02 has more wavy edges that sometimes cover background or cut into the model.

Looking at the classes that models gave to segmentations on Figure 6

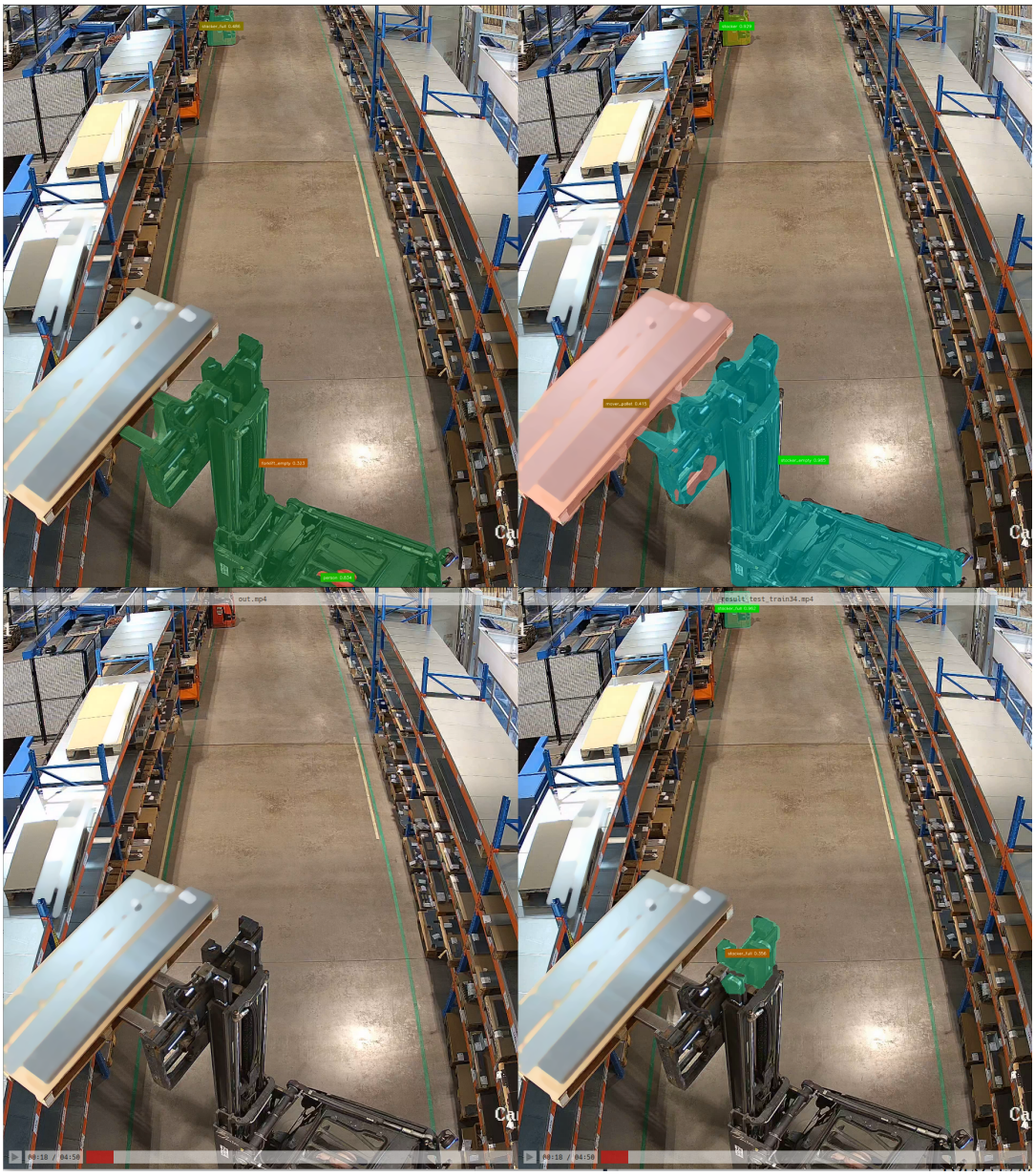


Figure 6. *Real world example 1*
Top - Mask DINO, EVA-02
Bottom - unlabelled, YOLO 640px

Figure 7. *Real world example 2*
Top - Mask DINO, EVA-02
Bottom - unlabelled, YOLO 640px

6.6 Evaluation

To actually compare the models we need to set some criteria:

- Performance
- Accuracy[6]

It is crucial that the segmenter we select is able work in real time or close to it. We can allow our processing to fall behind real time in situations of much activity but we need to be able to catch up. Also the model cannot require a large amount of computing resources for a single camera because MCHT will need to handle a large amount of cameras at once.

We need a segmenter that is accurate enough where we are able to interpret the correct state of the objects of interest over time. A few incorrect classifications or segmentations is acceptable because later in the MCHT pipeline we calculate uncertainty over time, which should filter out any occasional mistakes.

It is also worth noting that we can make compensate for a low result in one criteria with high result in the other. For example, if the model has low accuracy but high performance then we can run the model more often to get more results which over time should give us the correct states. On the other hand, if the model is highly accurate but slow to run then we can run it less often while being confident that we can get good detections on each frame.

Having set these criteria we started evaluating the models for our purpose. The first decision was the eliminate both YOLO models. While they have the large advantage of very quick inference, their accuracy is not up to par with what we need. We see more benefits from increased accuracy as opposed to more speed.

From the remaining models, EVA-02 and Mask DINO, we chose to go with Mask DINO. While it does have a very large VRAM requirement, the additional speed and especially accuracy outweigh the negatives.

[6]In this section, we use the term accuracy in a more general way to mean the ability to detect, segment and label correctly.

7. Related work

The larger goal of the work presented in the current thesis is related to asset tracking within the manufacturing environment. Typically such systems involve a combination of indoor and outdoor tracking systems, GNSS for outdoor and a combination of Wifi/Bluetooth or proprietary beacon systems indoors. Such systems are outlined e.g. in [21]. With reduced costs in mind we set out to solving the task of camera based inventory tracking where the inventory is on pallets. Pallets are moved by a finite list of assets: forklifts, pallet jacks and stackers. One of the options to achieve the tracking of pallets is to opt for a system that tracks assets. There exist such systems e.g. solution provided by ForkNAV[1] where a custom device is required for each forklift to perform the tracking. Alternatively, there are also various IoT based pallet tracking solutions where an IoT device is attached to each pallet[2]. There are several issues with such solutions: the cost of installing the tracking devices to thousands of pallets and also creating some kind of indoor navigation system to compensate that GNSS does not work well indoors. Another approach to pallet tracking is described in [22] where the authors electronically track a forklift that can detect loaded pallets with RFID readers. While this approach is shown to work quite well, it suffers from the problems of manually having to label pallets with RFID tags and extra equipment needed to be installed on forklifts.

Much of the research in computer vision focuses on tracking known objects or very similar looking objects for which it is possible to train object detectors. In pallet tracking applications there are some attempts, e.g. for locating pallets using forklift mounted cameras [23] or apparently promising tracking pallets with boxes[3]. In the latter application it may appear that similar problem is solved compared to the one in the current thesis, but on closer look it appears that the authors say:

> One primary application is the determination of warehouse zone occupancy. By accurately tracking the movement of objects in different zones, real-time occupancy data could be easily garnered, which optimizes inventory management and streamlines logistics.

Meaning that that they mention Yolo5 [24] for object detection and DeepSORT [25] for

[1] `https://www.forknav.eu`
[2] `https://www.digitalmatter.com/applications/gps-pallet-tracking/`
[3] `https://www.mdpi.com/2076-3417/13/17/9895`. Accessed 20.04.2024.

tracking, but the system has not actually been developed to perform tracking but rather warehouse zone occupancy determination.

Multi object tracking (MOT) has received a large amount of attention over recent years and available solutions work quite well when it is possible to train an object detector for particular type of objects. Many of the solutions are based on the Simple Online and Realtime Tracking (SORT) algorithm [26] that internally uses a Kalman filter and the Hungarian algorithm and has quite good performance despite being fairly simple. One of the main problems with SORT is frequent identity switches that has been addressed by more recent work, e.g. DeepSORT [25] that adds a deep association metric to SORT to improve identity preservation.

ByteTrack [27] uses an alternative method to utilizing low confidence detections to improve tracking accuracy. Bot-Sort [28] combines the approaches of SORT and ByteTrack and achieves better quality metrics.

The authors of observation centric SORT (OC-SORT) [29] algorithm improve the performance of SORT-like Kalman filter based tracking in the cases of occlusion and non-linear movement. StrongSORT [30] approach revisits the DeepSORT approach and in addition introduces a Gaussian-smoothed interpolation (GSI) based on Gaussian process regression to relieve the missing detection problem in the StrongSORT++ tracker. Hybrid-Sort [31] uses weak cues like confdence state, height state and velocity direction in addition to strong cues to improve tracking performance under occlusion. The algorithm does not require training for particular application.

It is worth mentioning that there are various built in trackers also in computer vision libraries such as e.g. OpenCV [32] that by the time of writing the thesis has reached version 4.9[4]. The pre-deep learning era built-in trackers in OpenCV [33] are able to track patches of images, but they lack semantic information of what is being tracked.

In the current task we can track assets which we can train object detectors or segmenters for. Initial effort towards achieving the goal was made in [1] where the author demonstrated that using multiple security cameras, without any markers or tags, it is possible to track pallet jacks in simpler situations. The key observation was that instead of tracking the inventory which changes all the time and for which training an object detector or segmenter is very hard, the author opted for tracking pallet jacks and their status. It was noted that the implemented system works but has problems - mainly it struggles with occlusion and tracker id jumping. The suggestions for fixing the problems were suggested in future

[4] `https://github.com/opencv/opencv`

work: to increase the number of cameras watching each area and to strive for better camera position and calibration parameters.

As a precursor to the current work we solved the camera calibration task using structure from motion approach. We devised a solution where the area of interest is photographed and a point cloud model is constructed using Colmap [5, 6]. The floor is detected and the coordinate system is rearranged so that the xy axis is aligned with the floor. The locations of security cameras are then inferred using Colmap and we are able to project every pixel from the security cameras to the corresponding coordinate on the floor.

Since we solved the extrinsic camera calibration problem we tried various filtering solutions to filter the detections of the model developed in [1], but since the introduction of new equipment (stackers) that are very frequently used, the old model was no longer accurate enough.

For asset tracking we had to decide between using object detection and instance segmentation solutions. We opted for instance segmentation because bounding boxes offer too much overlap in the scenes in our application and in addition instance segmentation offers better insight into what features are recognized by which class, i.e. it is easier to debug the model and suggest what data needs to be added to fix certain problems. Thus with segmentation we get more precise indication of where the assets are in our scene and it is easier to position them precisely. Also, once we have the annotated data, it is easy to infer object detection annotations from segmentation annotations while the reverse is impossible.

We set out to solving the inventory tracking with security cameras. There are solutions in the literature where cameras are used for inventory tracking. In [34] an application used in Daimler is described where computer vision techniques are used to read bar codes available on inventory equipment. In our solution the cameras are generally placed too far to be able to read bar codes. We rather aim to achieve to record the movements of pallets. In [35] the authors propose the use of QR codes to track pallets in a warehouse. This solution eliminates the unknown object tracking problem. However, the need to add tracking markers to the pallets makes it not suitable for our needs and purposes.

A very recent attempt to track inventory in the setting of a mobile robot with depth cameras has been described in [36]. Again such approach requires autonomous robots roaming around to detect if something has changed position. We in turn detect actual movements of inventory by recording the moment the pallet starts moving with a warehouse asset until it is left in a new position either in a shelf or on the floor.

8. Conclusion

In this thesis we set out to train an instance segmenter for warehouse equipment. One of the main objectives was to investigate the usefulness of synthetic data in the training process to reduce the need for manually labelled images which are time consuming to create.

We started by creating the synthetic data that we need. We used NVidia Omniverse to create a replica of our real warehouse, populate it with randomized pallets. A random scene generator was added that placed different movers in different configurations, in addition to random clutter. From these totally random scenes we gathered our data that was automatically annotated, almost ready to be used. What remained was a little post processing and we had our data.

Various experiments and tests were carried out to assess the effects of synthetic data in the model training process. We compared models that were trained with and without synthetic data and found it increased their goodness significantly. Another experiment tested our hypothesise that firstly training a base model on synthetic data and then fine tuning it would result in a better model as opposed to training one directly on the fine tuning dataset. This gave us mixed results, where one model did show improvements, while other two achieved worse results. Different segmenters were compared to gather info about their inference speed and VRAM requirements to give us a fuller picture when deciding on the best segmenter for our tracking project.

As a result of these model trainings and tests we found the most suitable segmenter, MaskDINO, have a model with very good accuracy and recall. Our best model reached AP50 of 0.710, AP50-95 of 0.580 and AR50-95 of 0.776. This fulfilled our last goal of training a model to detect different warehouse equipment, their states and pallets they are carrying.

8.1 Further work

We see that more work needs to be done to assess and optimize the training of transformer based models. Due to time constraints we were unable test for example test how much synthetic data is optimal for these types of models. Maybe these types of models could deliver even better quality with larger amounts on synthetic data? It is worth while noting

that one of the top performing models, Mask Frozen-DETR [19], that we were not able to test due to the implementation not being available, claims to have up to 10 times faster training times with comparable quality performance with Mask DINO.

Another possible future work involves creating more true to life synthetic data. This contains multiple aspects. Firstly the look of the synthetic images - they are too perfect and clean. Current images have a very vibrant look, do not have any compression and artefacts associated with it, wear and tear on equipment and the environment. The aim would be to more closely match the image that would come out a real security camera in a real warehouse.

Another aspect is the inclusion of more situations and more variety of the equipment. For example half length pallets, pallets positioned on ends of forks, forklifts and stackers with masts extended, etc.

The goal of this work would be to see if these improvements would result in better models. Perhaps a good enough models could be created solely on synthetic data.

References

[1] Ivar Vipper. *Inventory movement tracking using video surveillance system*. 2020. URL: https://digikogu.taltech.ee/et/Item/41b80a70-73fd-4c08-997a-c6bea3375c94.

[2] Juraj Fulir et al. "Synthetic Data for Defect Segmentation on Complex Metal Surfaces". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023 - Workshops, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 4424–4434. DOI: 10.1109/CVPRW59228.2023.00465. URL: https://doi.org/10.1109/CVPRW59228.2023.00465.

[3] Deokhwan Park et al. "Deep Learning based Food Instance Segmentation using Synthetic Data". In: *18th International Conference on Ubiquitous Robots, UR 2021, Gangneung, South Korea), July 12-14, 2021*. IEEE, 2021, pp. 499–505. DOI: 10.1109/UR52253.2021.9494704. URL: https://doi.org/10.1109/UR52253.2021.9494704.

[4] Danilo G. Schneider and Marcelo Ricardo Stemmer. "Synthetic Data Generation on Dynamic Industrial Environment for Object Detection, Tracking, and Segmentation CNNs". In: *Technological Innovation for Connected Cyber Physical Spaces - 14th IFIP WG 5.5/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2023, Caparica, Portugal, July 5-7, 2023, Proceedings*. Ed. by Luis M. Camarinha-Matos and Filipa Ferrada. Vol. 678. IFIP Advances in Information and Communication Technology. Springer, 2023, pp. 135–146. DOI: 10.1007/978-3-031-36007-7_10. URL: https://doi.org/10.1007/978-3-031-36007-7%5C_10.

[5] Johannes L. Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 4104–4113. DOI: 10.1109/CVPR.2016.445. URL: https://doi.org/10.1109/CVPR.2016.445.

[6] Johannes L. Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III*. Ed. by Bastian Leibe et al. Vol. 9907. Lecture Notes in Computer Science. Springer, 2016, pp. 501–518. DOI: 10.1007/978-3-319-46487-9_31. URL: https://doi.org/10.1007/978-3-319-46487-9%5C_31.

[7] NVIDIA Corporation. *NVIDIA Omniverse*. 2024. URL: https://www.nvidia.com/en-eu/omniverse.

[8] NVIDIA Corporation. *Omniverse Warehouse Creator*. 2024. URL: https://docs.omniverse.nvidia.com/isaacsim/latest/features/warehouse_logistics/ext_omni_warehouse_creator.html.

[9] NVIDIA Corporation. *Omniverse Replicator*. 2024. URL: https://docs.omniverse.nvidia.com/extensions/latest/ext_replicator.html.

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

[11] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV].

[12] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[13] NVIDIA Corporation. *NVidia Ada Lovelace professional GPU architecture*. 2023. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/technologies/NVIDIA-ADA-GPU-PROVIZ-Architecture-Whitepaper_1.1.pdf.

[14] NVIDIA Corporation. *NVidia Ada GPU Architecture*. 2023. URL: https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf.

[15] Yuxin Fang et al. *EVA-02: A Visual Representation for Neon Genesis*. 2023. arXiv: 2303.11331 [cs.CV].

[16] Feng Li et al. "Mask DINO: Towards A Unified Transformer-based Framework for Object Detection and Segmentation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 3041–3050. DOI: 10.1109/CVPR52729.2023.00297. URL: https://doi.org/10.1109/CVPR52729.2023.00297.

[17] Yuxin Fang et al. "EVA: Exploring the Limits of Masked Visual Representation Learning at Scale". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 19358–19369. DOI: 10.1109/CVPR52729.2023.01855. URL: https://doi.org/10.1109/CVPR52729.2023.01855.

[18] Yixuan Wei et al. "Contrastive Learning Rivals Masked Image Modeling in Fine-tuning via Feature Distillation". In: *CoRR* abs/2205.14141 (2022). DOI: 10.48550/ARXIV.2205.14141. arXiv: 2205.14141. URL: https://doi.org/10.48550/arXiv.2205.14141.

[19] Zhanhao Liang and Yuhui Yuan. “Mask Frozen-DETR: High Quality Instance Segmentation with One GPU”. In: *CoRR* abs/2308.03747 (2023). DOI: 10.48550/ARXIV.2308.03747. arXiv: 2308.03747. URL: https://doi.org/10.48550/arXiv.2308.03747.

[20] Wenhui Wang et al. “Image as a Foreign Language: BEiT Pretraining for All Vision and Vision-Language Tasks”. In: *CoRR* abs/2208.10442 (2022). DOI: 10.48550/ARXIV.2208.10442. arXiv: 2208.10442. URL: https://doi.org/10.48550/arXiv.2208.10442.

[21] Fahim Ahmed et al. “Comparative Study of Seamless Asset Location and Tracking Technologies”. In: *Procedia Manufacturing* 51 (2020). 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021), pp. 1138–1145. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2020.10.160. URL: https://www.sciencedirect.com/science/article/pii/S2351978920320175.

[22] Andrea Motroni, Alice Buffi, and Paolo Nepa. “Forklift tracking: Industry 4.0 implementation in large-scale warehouses through uwb sensor fusion”. In: *Applied Sciences* 11.22 (2021), p. 10607.

[23] Tianjian Li et al. “Application of convolution neural network object detection algorithm in logistics warehouse”. In: *the Journal of Engineering* 2019.23 (2019), pp. 9053–9058. DOI: https://doi.org/10.1049/joe.2018.9180.

[24] Glenn Jocher et al. *ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation*. Version v7.0. Nov. 2022. DOI: 10.5281/zenodo.7347926. URL: https://doi.org/10.5281/zenodo.7347926.

[25] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. “Simple online and realtime tracking with a deep association metric”. In: *2017 IEEE International Conference on Image Processing, ICIP 2017, Beijing, China, September 17-20, 2017*. IEEE, 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962. URL: https://doi.org/10.1109/ICIP.2017.8296962.

[26] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing, ICIP 2016, Phoenix, AZ, USA, September 25-28, 2016*. IEEE, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003. URL: https://doi.org/10.1109/ICIP.2016.7533003.

[27] Yifu Zhang et al. “ByteTrack: Multi-object Tracking by Associating Every Detection Box”. In: *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXII*. Ed. by Shai Avidan et al. Vol. 13682. Lecture Notes in Computer Science. Springer, 2022, pp. 1–21. DOI:

10.1007/978-3-031-20047-2_1. URL: https://doi.org/10.1007/978-3-031-20047-2%5C_1.

[28] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. *BoT-SORT: Robust Associations Multi-Pedestrian Tracking*. 2022. arXiv: 2206.14651 [cs.CV].

[29] Jinkun Cao et al. "Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 9686–9696. DOI: 10.1109/CVPR52729.2023.00934. URL: https://doi.org/10.1109/CVPR52729.2023.00934.

[30] Yunhao Du et al. "StrongSORT: Make DeepSORT Great Again". In: *IEEE Trans. Multim.* 25 (2023), pp. 8725–8737. DOI: 10.1109/TMM.2023.3240881. URL: https://doi.org/10.1109/TMM.2023.3240881.

[31] Mingzhan Yang et al. "Hybrid-SORT: Weak Cues Matter for Online Multi-Object Tracking". In: *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. Ed. by Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan. AAAI Press, 2024, pp. 6504–6512. DOI: 10.1609/AAAI.V38I7.28471. URL: https://doi.org/10.1609/aaai.v38i7.28471.

[32] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[33] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. "Online Object Tracking: A Benchmark". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*. IEEE Computer Society, 2013, pp. 2411–2418. DOI: 10.1109/CVPR.2013.312. URL: https://doi.org/10.1109/CVPR.2013.312.

[34] Stephen Gregory et al. "A computer vision pipeline for automatic large-scale inventory tracking". In: *ACM SE '21: 2021 ACM Southeast Conference, Virtual Event, USA, April 15-17, 2021*. Ed. by Kazi Rahman and Eric Gamess. ACM, 2021, pp. 100–107. DOI: 10.1145/3409334.3452063. URL: https://doi.org/10.1145/3409334.3452063.

[35] A Vukićević et al. "A smart Warehouse 4.0 approach for the pallet management using machine vision and Internet of Things (IoT): A real industrial case study". In: *Advances in Production Engineering & Management* 16.3 (2021), pp. 297–306.

[36] Adam Rashid et al. “Lifelong LERF: Local 3D Semantic Inventory Monitoring Using FogROS2”. In: *CoRR* abs/2403.10494 (2024). DOI: 10.48550/ARXIV.2403.10494. arXiv: 2403.10494. URL: https://doi.org/10.48550/arXiv.2403.10494.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Marek Lahk

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Warehouse Equipment Instance Segmenter Training for Digital Twin Using Synthetic Data and Fine Tuning", supervised by Juhan-Peep Ernits
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

24.05.2024

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Replicator code example

```
with rep.trigger.on_frame(max_execs=100):
    with camera:
        rep.modify.pose(
            position=rep.distribution.uniform(
                (-6, -33, 4.5), (6, 33, 7.5)))
```