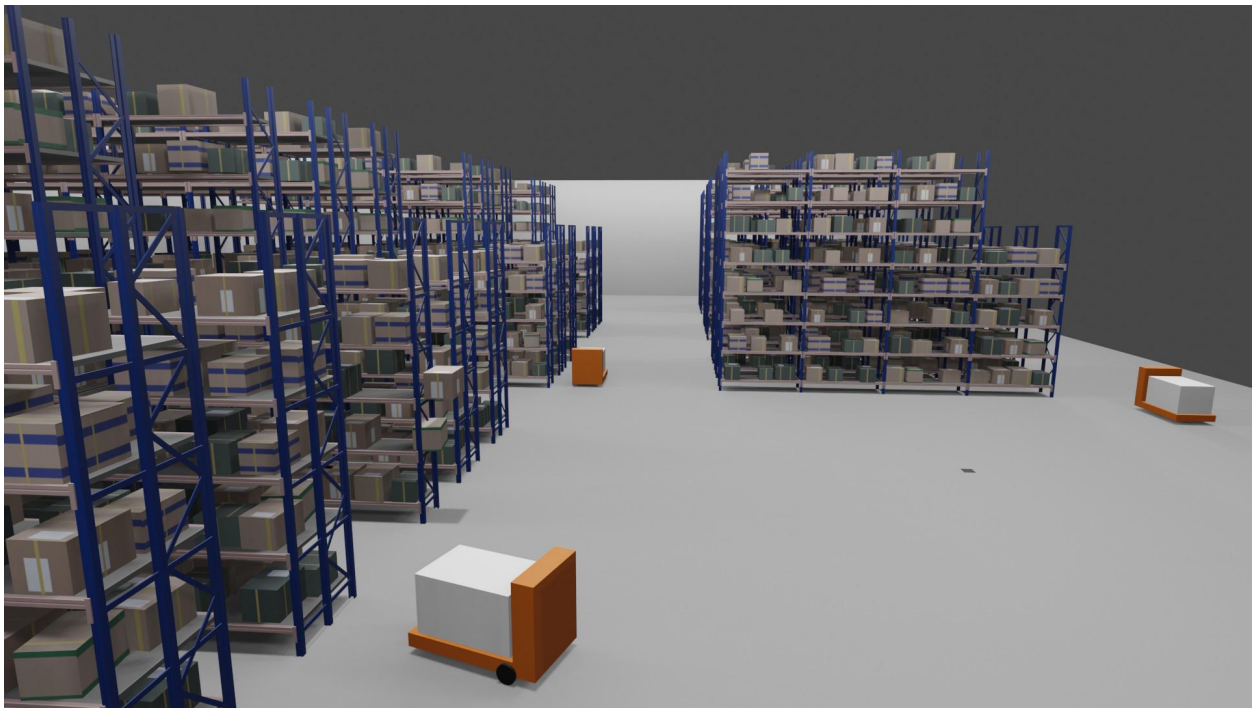
CHALMERS
UNIVERSITY OF TECHNOLOGY



Indoor tracking using a central camera system

Use computer vision to track multiple objects in a simulated warehouse environment

Master's thesis in Systems, Control and Mechatronics

Jonas Lindberg
Sara Roth

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Indoor tracking using a central camera system

Use computer vision to track multiple objects in a simulated warehouse environment

JONAS LINDBERG
SARA ROTH

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Systems and control
Automation
Chalmers University of Technology
Gothenburg, Sweden 2021

Indoor tracking using a central camera system
Use computer vision to track multiple objects in a simulated warehouse environment
JONAS LINDBERG
SARA ROTH

Supervisor: Jonas Tillström, Sigma Technology Insights AB
Supervisor: Martin Dahl, Department of Electrical Engineering
Examiner: Petter Falkman, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Systems and control
Automation
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Simulated warehouse with Automatic Guided Vehicles

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Indoor tracking using a central camera system
Use computer vision to track multiple objects in a simulated warehouse environment
JONAS LINDBERG
SARA ROTH
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Modern Automatic Guided Vehicles, AGVs, use different types of sensors for positioning themselves within an operating scene, such as a warehouse. These sensors are typically mounted onboard the vehicles which can result in a limitation in the field of view and also make for a fragile system sensitive to bumps and collisions.

This thesis aims at position and track AGVs using static cameras mounted in the ceiling of a simulated warehouse. The thesis further seeks to reduce the physical sensitivity of vehicles by substituting onboard sensor equipment for the developed system and furthermore provide a larger field of view. Using an object detection neural network, AGVs and humans were detected and projected onto a shared plane using homographies. Seven different cameras were used in parallel which generated bounding boxes from different views by the object detection algorithm. The intersection between these boxes was determined and the center point of the intersection was assumed to be the detected objects center point. A non-linear Kalman Filter was implemented to track objects troughout frames and provide a smoothed position estimate.

The algorithm was able to track AGVs and humans throughout the environment with two identity switches in a video with six objects in 300 frames. The mean positioning error of an AGV of size 1.4×0.8 meters was found to be 77 millimeters.

A discussion on the possibility to substitute present day sensor equipment for the developed system was carried out and found that the developed system could indeed be valuable in a similar real-life warehouse environment for improved collision avoidance. More research would have to be conducted on whether the system could actually replace onboard sensor equipment for control of AGVs.

Keywords: Computer vision, Sensor fusion, Machine learning, Image analysis.

Sammanfattning

Moderna automatiskt styrda fordon, AGVer, använder olika typer av sensorer för att positionera sig i en operativ miljö, så som ett lager. Sensorerna är typiskt sett monterade ombord på fordonet vilket kan leda till begränsningar i fordonets synfält och ömtåliga system som är känsliga för tillstötningar och kollisioner.

Avhandlingen syftar till att, med hjälp av statiska kameror monterade i taket i ett simulerat lager, positionera och följa AGVer i det simmulerade lagret. Avhandlingen syftar vidare till att minska den fysiska känsligheten för fordonen genom att ersätta sensorutrustning ombord mot det utvecklade systemet samt att erbjuda ett större synfält. Ett neuralt nätverk för objektdetektering implementerades för att hitta AGVer och människor. Dessa projicerades sedan ner till ett gemensamt plan med hjälp av homografier. Sju olika kameror användes parallellt och skapade avgränsningslådor (bounding boxar) med hjälp av objektdetekteringsalgoritmen. Överlappningen mellan de skapade bounding boxarna fastställdes och mittpunkten av överlappningen antogs vara det detekterade objektets mittpunkt. Ett icke-linjärt Kalmanfilter implementerades för att spåra objekt genom bildrutor och ge ett utjämnat estimat.

Algoritmen kunde spåra AGVer och människor genom miljön med två identitetsbyten i en video med sex objekt i 300 bildrutor. Medelvärdet av positioneringsfelet av en AGV med måtten 1.4×0.8 meter blev 77 millimeter.

En diskussion kring möjligheterna att byta ut dagens sensorutrustning mot det framtagna systemet gjordes och visade att det framtagna systemet mycket väl kan användas i en liknande lagermiljö i verkligheten för att undvika kollisioner. Det konstaterades att mer forskning behövs för att kunna avgöra om systemet helt kan ersätta sensorer monterade ombord på AGVer.

Nyckelord: Datorseende, Sensor fusion, Maskininlärning, Bildanalys.

Acknowledgements

We would like to thank our Chalmers supervisor Martin Dahl for your support and feedback throughout the project. Martin introduced us to Ze Zhang who have given us great feedback on our challenges throughout the project. Thank you both for taking the time to help us out. Your input, questions and guidance have really helped us to produce a work we are very proud of.

Thank you to our examiner Petter Falkman for your help in making this thesis project happen and for taking on our project with your great enthusiasm.

We would like to express our gratitude to Sigma Technology Insights for offering us office spaces and introduced us to your network. A special thank you to our Sigma supervisor Jonas Tillström for your support and feedback trough out the project. A great thank you to Ulrika Stålhammar and Robert Åberg at Sigma for the great support you've given us, your interest in our project and the opportunities you've given us.

Thank you.

Jonas Lindberg & Sara Roth, Gothenburg, Maj 2021

Contents

List of Figures

List of Tables

Acronyms

AGV Automatic Guided Vehicle.
AP Average Precision.

BoF Bag of freebies.
BoS Bag of specials.

CNN Convolutional Neural Network.
CPU Central Processing Unit.
CSPNet Cross Stage Partial Network.

FN False Negative.
FP False Positive.
FPN Feature Pyramid Networks.
FPS Frames per second.

GT Ground Truth.

IDSW Identity Switches.
IMU Inertial Measurement Unit.
IoU Intersection over Union.

LiDAR Light Detection and Ranging.
LTI Linear Time Invariant.

mAP Mean Average Precision.
MOTA Multiple Object Tracking Accuracy.

NMS Non Maximum Suppression.

PAN Path Aggregation Network.

QR Quick Response.

RANSAC Random Sample Consensus.
ReLU Rectified Linear Unit.
RGB Red Green Blue.

SLAM Simultaneous Localization and Mapping.
SNR Signal to Noise Ratio.
Sonar Sound Navigation and Ranging.
SSD Single Shot Multi-Box Detector.

TN True Negative.
TP True Positive.

UKF Unscented Kalman filter.

VGV Vision Guided Vehicles.

YOLO You Only Look Once.
YOLOv2 You Only Look Once 2nd version.
YOLOv3 You Only Look Once 3rd version.
YOLOv4 You Only Look Once 4th version.
YOLOv5 You Only Look Once 5th version.

1
Introduction

Computer vision, the field of making computers understand what they see and make decisions upon this, is growing and is being implemented in more and more branches of industry. One field where computer vision has gained popularity is the task of observing moving objects, more commonly known as tracking. Examples of this could include tracking customer walk patterns in shopping malls for analyzing consumer behavior [1] or tracking hockey players on a field to evaluate the game in every time instance [2]. This development is driven by the development of deep convolutional neural networks which has proven to have great results in object detection.

This thesis will investigate how computer vision can be used for solving tracking cases. More specifically the thesis will focus on using external static cameras mounted in the ceiling or walls inside of a building to track Automatic Guided Vehicles, AGVs, which are autonomous vehicles mainly used for material transport. A common use of these vehicles is in warehouses and factories where they autonomously deliver material from point to point. AGVs exist in different sizes, optimized for their specific task or specific freight. Normally, AGVs carry different types of sensors on board to determine necessary parameters for control, such as position and velocity. These parameters are then used to control the AGV toward its destination.

AGVs carry sensitive sensor equipment while in some cases being exposed to a rough environment with the risk of collision with other vehicles, debris, shelves and more. Techniques allowing for sensor equipment to be moved to a less exposed place could lead to less consumption of sensors, relating to Goal 12 in United Nations Agenda for Sustainable Development list 17 goals [3]. Furthermore, providing an accurate overview of the operating environment could lead to more streamlined logistics, fulfilling goal 9 of the United Nations Agenda for Sustainable Development.

1.1 Research questions

The thesis should provide answers to the following research questions:

- How well can a central camera based tracking system predict the position of tracked objects based on measurements of the deviation from true position and the deviation from true velocity?

The system needs to be able to accurately estimate the position of the tracked objects. This is crucial for a well-functioning tracking system. The deviation is simply the distance between the estimated position for the object and the true position.

- How are AGVs tracked today and how can a central camera based system be used to replace or support these current systems?

In order to motivate the performance of the system and select desired output parameters a literature study should be conducted on existing solutions. A deeper knowledge of existing systems and technologies plays a large role in developing a system that can either replace or support existing systems.

- Can a visual system provide measures to avoid collisions in a warehouse environment?

Rather than allowing sensors onboard the vehicle to detect and measure the distance to possible obstacles, it should be investigated if this feature is possible to conduct using only cameras. The cameras will not be placed onboard the vehicles.

1.2 Aim

This thesis aims at providing position and velocity parameters for a target at a given time using only external cameras and without any input from the target being positioned. Furthermore, the thesis aims at tracking the targets throughout a video feed. Tracking will be evaluated on the performance of the developed system to detect objects and keep an assigned identification troughout a video feed.

The thesis will not focus on a single type or brand of AGVs but rather show a proof of concept on tracking based on external cameras. Development and evaluation will be conducted on simulated data rather than real life data. Object detection algorithms will be trained and evaluated on virtual models of AGVs. This thesis does not make any effort in actually controlling AGVs.

1.3 Ethics

Object detection technology allows computers to recognize objects in an image. This is used in a variety of modern products such as advanced assistance systems in cars to detect driving lanes and pedestrians, facial recognition, marketing applications and video surveillance [4] [5]. In the automotive industry, object detection could both be used for driver verification in form of facial recognition to make sure no

unauthorized drivers use the vehicle as well as driver assistance to detect lanes and warn the driver if the vehicle drifts off road. Monitoring and object tracking systems are also a growing technology with a variety of uses. These include surveillance and traffic control to provide information about traffic conditions, for example, velocity distribution and density of vehicles, to detect possible traffic congestion. It can also help identify vehicles that violate traffic rules by for example drive in a forbidden crossroad or cross a red light [6].

As tracking systems become more affordable, with more efficient models and a smaller computational footprint, they become a more common feature in society. These techniques provide a lot of benefits, for example, real time tracking of busses provides relevant information to passengers waiting at a bus stop. However, the technology also allows for tracking people very easily.

Even though detection and tracking technology is developed to ease peoples lives it also brings growing concerns about privacy, data protection and integrity breaches. How the information is used and transparency in the usage are important questions to support the big developments in the area and ensure systems are used for the right purposes. An example of unethical tracking could be to monitor how much time workers spend away from their workstations, while a more ethical example could be to monitor workers to assure they don't cross the path of an autonomous vehicle.

Information and education on the topic are important. Warning signs, similar to warning signs for surveillance cameras, could be implemented. When tracking is used for purposes that do not require information about peoples identity algorithms to mask faces could be used. It is important to consider ethics and human rights during this development. As the technology becomes more advanced, the privacy protection becomes increasingly harder.

2

Theory

To be able to track and control an AGV, knowledge about the physical surroundings and the vehicles position is needed. Furthermore, knowledge about velocity and heading is needed to determine future states of position to avoid collisions, ensure the AGV arrives at the correct spot and takes the correct path to get there. There are different techniques for deriving these parameters and determining the physical surroundings. This chapter gives a background on existing techniques based on different sensors as well as background theory on how to detect, classify and track objects in frames.

2.1 Current AGV positioning methods

An AGV have a need to navigate through its surroundings to perform its tasks. There are two types of systems that are required in an AGV navigation system, trajectory creation and position measurement. The vehicles are autonomous and in order for a vehicle to be called autonomous, the vehicle has to be able to go from point A to point B without human intervention.

The vehicle should be able to follow a path and depending on how the path is created it can be divided into two different subgroups, fixed path and free path. The fixed path has the trajectory established from the start and the AGV should simply follow the path. One example is magnetic floor tape. The free path is when the AGV itself decides the trajectory and can avoid obstacles. One example of a free path is Simultaneous Localization and Mapping, SLAM, which uses sensor fusion to position itself and simultaneously maps the environment [7].

The AGV also has to calculate its current position, a task that can be divided into absolute position and relative position. For absolute position, the vehicle knows its position in the environment at all times. One example is positioning with laser. In relative position, the vehicle calculates its current position relative to a previous position [8]. One example is the odometric systems which is a system to estimate the change of the position in time [9], for example rotary encoders in wheels. The different parts of AGV navigation can be seen in Figure 2.1.

Existing navigation method for AGV navigation can roughly be divided into two main groups, namely track guided and free navigation.

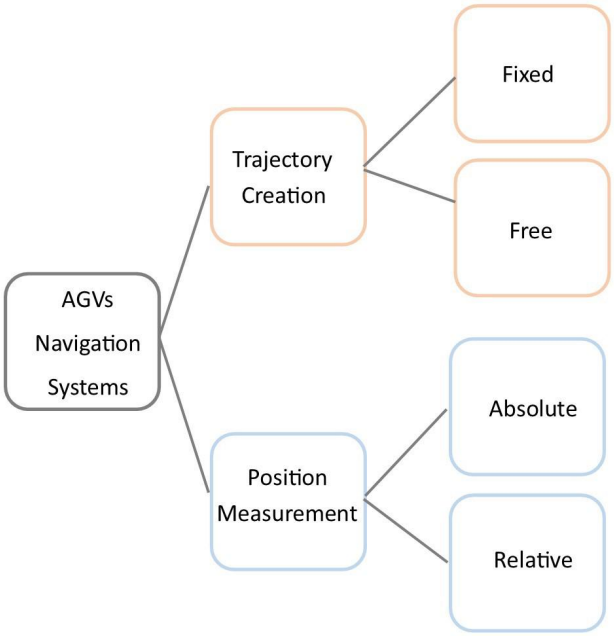


Figure 2.1: Visualization of the different parts in a AGV navigation system

2.1.1 Track guided navigation

The technique uses physical markers in the environment for navigation. These markers make up a physical track for the vehicle to follow. Markers can be of different types and utilize different technologies. Two examples are rigid track and triangulation.

2.1.1.1 Rigid track guidance

AGV vehicles used in production are in some cases navigated by magnetic fields in the floor allowing for a magnetic field sensor to guide the vehicle along the path [10], see Figure 2.2. To make this type of navigation system work, the travel path has to be completely known ahead and there is not much space for changes since that means that the rigid track has to be rebuilt. Since the AGV is running on previously designed trajectories that are known ahead and the AGV is not able to deviate from the track, this navigation method is therefore fixed. The position measurement is relative since its position is calculated based on its previous position. To be able to track and navigate vehicles free without a guided track all the sensor data needs to be processed in real time.

2.1.1.2 Triangulation methods

Triangulation can be used with different types of sensors to estimate the position of the vehicle. One example is triangulation with laser in Figure 2.3.
The time between emission and reflection of a laser beam is measured and yields a distance measurement. Combining distance measurements from at least three

Figure 2.2: Visualization of rigid track guidance for AGVs

reflectors allows for a position calculation using triangulation. This makes laser based navigation a free and absolute method. The laser rotates 360 degrees and defines a radial coordinate reference that allows for calculating the X,Y coordinates of the reflectors [11]. In methods with reflectors, the AGV relies on the localization of the landmarks installed for this specific purpose. This means that the planning of the reflectors has to be done carefully to sufficiently cover the entire environment to enable accurate localization everywhere. Furthermore the reflectors needs to be positioned at a joint height. It also has to avoid symmetries to be able to make unique associations of the observed reflectors and the created map. If changes are made in the facility the reflectors might have to be reinstalled or moved, leading to both economical disadvantages and time consumption.

2.1.2 Free navigation

In free navigation, the vehicles navigate by mapping the surrounding area, see Figure 2.4. Instead of reflections of the surrounding area, all sensors are located on the vehicle and does not need any sensor specific equipment in the surrounding area. Identification of objects in the surrounding area is performed and the vehicle navigates by calculating the distance to the different objects [12]. This type of navigation enable high flexibility but can be complicated in varying and messy environments since the navigation system needs to be able to distinguish between static and moving objects. For this reason natural free navigation is suitable in environments with fixed walls and well defined profiles.

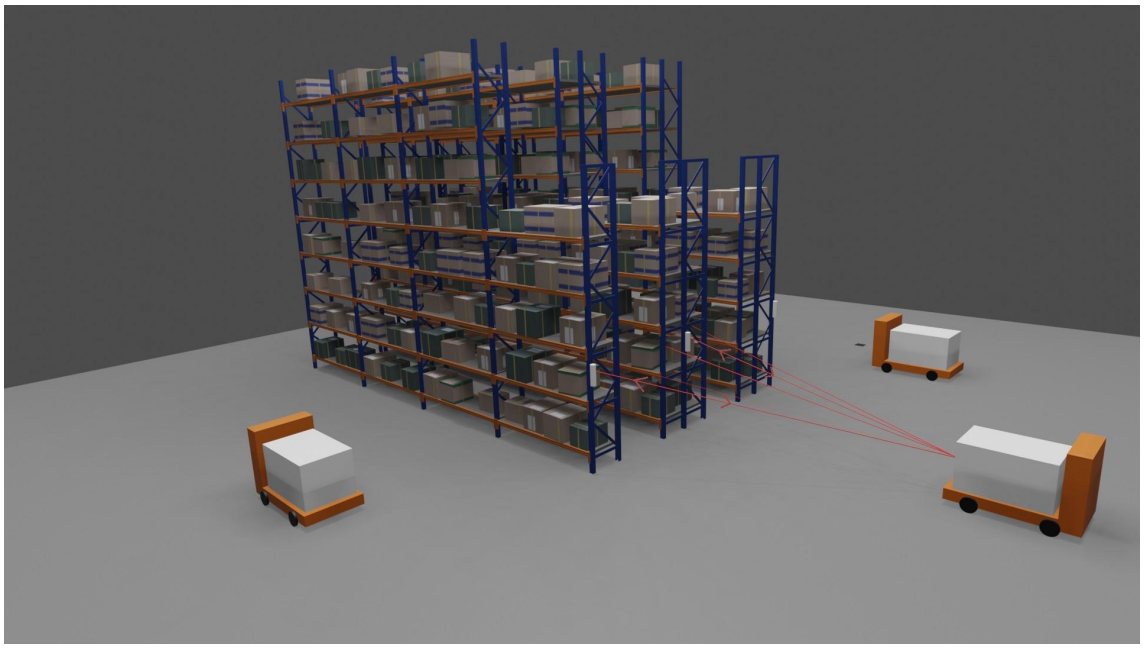


Figure 2.3: Visualization of triangulation positioning for AGV

2.1.2.1 Sound Navigation and Ranging

Sonar, short for Sound Navigation and Ranging, is a technique in which sound waves are used for measuring distances. These systems can either emit sound and listen for the echo or passively listen for sounds emitted by other objects [13]. In Ray *et al.*[14] an AGV using emitting sonar is designed. The sonar is used for finding obstacles along the calculated straight path towards the goal. If an obstacle is found, the AGV will deflect from the path and a new straight line is calculated towards the target. This type of navigation does not require prior knowledge about the surroundings but multiple obstacles could make the final overly complicated. This system is therefore also free and absolute. The system will treat all obstacles as static and hence do not take into consideration that an obstacle could in fact be another AGV crossing the calculated path.

2.1.2.2 Light Detection and Ranging

Another type of free navigation is navigation using Light Detection and Ranging, LiDAR. The vehicle measures the distance to a target by sending laser lights and measure the reflection with a sensor. The result can be computed as a point cloud in both two- and three-dimensions. The the-dimensional case is covered in Young and Simic [15] where the sensor is used for long range detection of obstacles at a certain height. In the covered case the LiDAR sensor is fused with readings from a camera. With a three-dimensional representation, point cloud, different objects needs to be segmented out and classified for it to be useful. In Rozsa and Sziranyi a so called two and a half- dimensional case is investigated [16]. The half dimension is achieved by tilting the LiDAR sensor and results in fewer points in the point cloud but the algorithm can reasonably well classify the objects detected. Since there is

no need for either guided tracks or previous position, LiDAR is a free and absolute method.

2.1.2.3 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping, SLAM, is a sensor fusion approach in which localization of the vehicle and mapping of the environment is achieved simultaneously [7]. Localization is achieved using internal sensors such as an odometer to estimate the change of the position in time [9], or an Inertial Measurement Unit, IMU, which can determine position using an accelerometer and a gyroscope [17]. Mapping of the surroundings can be made using LiDAR and cameras. Since localization and mapping are closely related (i.e. the surroundings appear different based on the localization of the vehicle) the data can be fused to give an accurate estimate for navigating the AGV.

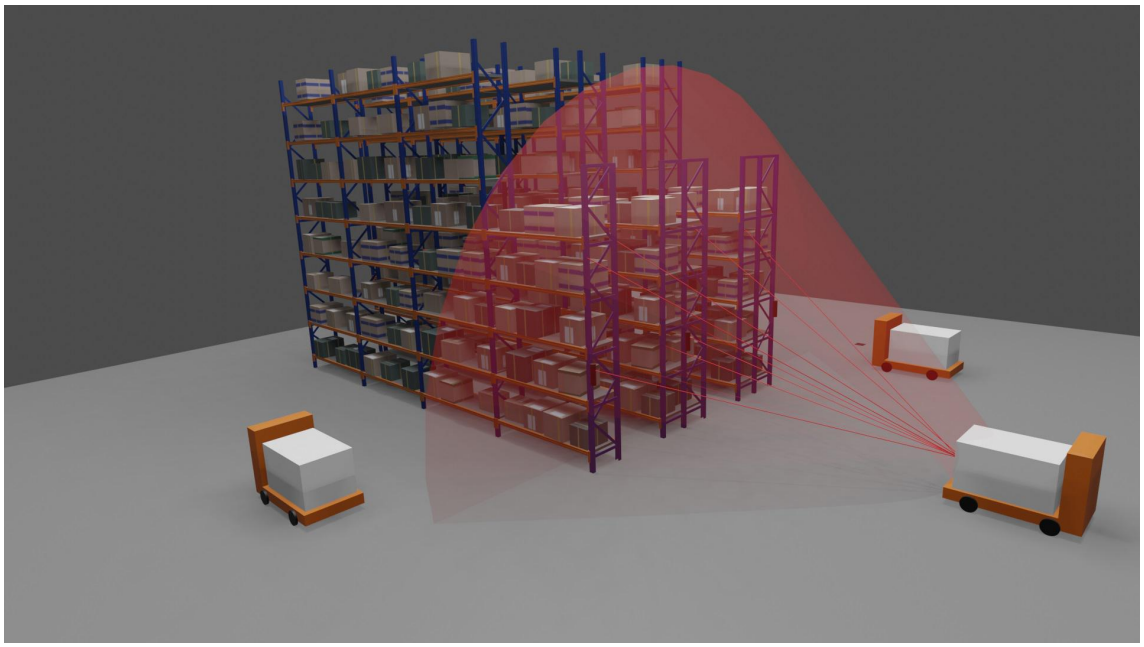


Figure 2.4: Visualization of natural free navigation for AGVs with both LiDAR and camera sensors

2.1.2.4 Optical navigation

Another type of positioning system is Vision Guided Vehicles, VGV, which use cameras to determine the characteristic of the environment and make decisions, see Figure 2.5. Vision guided vehicles can be divided into different subgroups. AGVs with Quick Response, QR, recognition is a fixed and absolute method that knows its absolute position by scanning a code located on the walls or on the floor in the environment. Guidance on the floor in form of strips painted in color is also a fixed and relative method while the third option, AGVs with three-dimensional cameras is a free and relative method. AGV with three-dimensional cameras uses multiple

stereo cameras attached to the vehicle to continuously capture and build a three-dimensional and computer-generated view of the surrounding. Since the AGV build a 3-dimensional map of its surrounding only based on the vision from the cameras, there is no need for static barriers or modification of the infrastructure, for example with magnets or reflectors [18]. Most of the optical systems are combined with inertial or odometric systems.

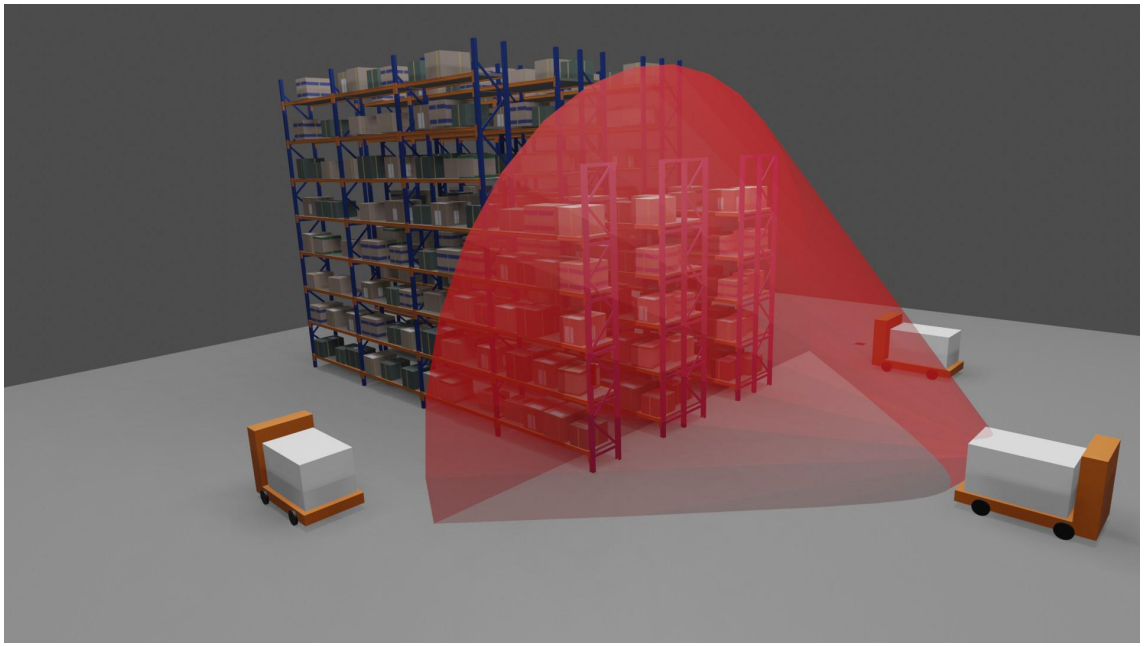


Figure 2.5: Visualization of optical navigation for AGVs

2.1.3 Current AGV control methods

In order to determine if the developed system can replace or supplement current systems, it is to be decided on what parameters are needed for AGV controlling. The thesis refers to existing path following techniques and the parameters needed rather than derive control systems equations from scratch. It is assumed that the physical properties of the vehicle are known and that the vehicle has sensors for describing the angle of turn on all steering wheels.

The task of controlling AGVs is covered in multiple articles, covering both two-wheel and four-wheel steering vehicles [19][20][21]. Controlling a path following AGV would require knowledge of the velocity, v, the angular velocity, $\dot{\theta}$ and the position, p, of the vehicle.

2.2 Object detection in computer vision

Object detection is a computer vision technique for locating occurrences of objects and their corresponding classes in an image. The detection algorithms typically use some kind of deep learning algorithm to produce results. Object detection is a key technology in different branches of technology, for example, advanced assistance systems in cars to detect driving lanes and pedestrians and in video surveillance. A variety of different techniques can be used for object detection.

Popular deep learning approaches include the use of Convolutional Neural Networks, CNNs, that can be trained to detect objects in images. The neural network can either be trained on large general datasets or on specific datasets for special implementations. Specialized datasets require tedious work to annotate ground truth labels but enable the algorithm to detect very specific objects, such as parts of a car engine. Deep learning approaches to object detection can be divided into two categories depending on the factors prioritized during training, one-staged and two-staged approaches.

You Only Look Once, YOLO, is an example of a one-stage detection algorithm together with RetinaNet and Single Shot Multi-Box Detector, SSD, while Mask Region Based-CNN and Faster Region Based-CNN are two examples of two-staged detection approaches. A two-stage detector uses a Region Proposal Network to first find the regions of interest, subsets of the image that might contain an object, and then use the cropped region for object classification and bounding-box regression. These models reach high accuracy and have good localization and precision. Two-staged detection algorithms take time to compute due to the complex pipelines and the difficulty in optimizing since each individual component has to be trained separately. Therefore, two-stages approaches should not be used when computational speed is of priority.

One-stage detectors perform prediction across the entire image using anchor boxes, predefined bounding boxes of set width and height, which are decoded to generate the final bounding boxes. Advantages of anchor boxes include the possibility to perform all predictions at once, eliminating the need to divide the image into regions. The approach allows for faster computation and execution, making the algorithm suitable for real-time object detection. The approach of treating object detection as a regression problem, like in one-staged detectors, is faster than two-staged but result in lower accuracy [22]. Finding an optimal trade off between accuracy and speed is not an easy task, but in general one can say that one-stage detectors are preferred in less complex images.

2.2.1 Deep learning

This chapter introduces deep neural networks, a key concept in deep learning. The objective of a neural network is to estimate a function

$$y = f(\mathbf{x}, \theta) \tag{2.1}$$

by learning the mapping θ from samples of data known as training data. In supervised learning the data, x, is bundled with labels of the correct output y. In the case of an image classifier the training data would consist of images labeled with the correct classification of the image. Below follows a brief introduction of the general feed forward neural network as well as the convolutional neural network, which is to be used in this thesis. The following introduction is based on Goodfellow et al. [23] and Nielsen [24].

2.2.2 Feedforward Deep Neural Network

Feedforward neural networks are the basis of deep learning and a crucial foundation for other types of deep learning networks.

The feed forward process refers to data flowing through the algorithm while the network refers to the representation of the algorithm. That is, multiple functions are coupled after one another to form a network. The neuron is a function, loosely inspired by neuroscience, that takes multiple inputs to compute its own value. Neurons are gathered in layers. Typically the layers would be denoted input, hidden and output layer. The layers between the input and output layers are the hidden layers. Where the layers for input and output often have a specified dimension depending on the problem the hidden layers can be assigned more freely. More neurons in more hidden layers will lead to increased complexity, which is not always guaranteed to yield better accuracy. The number of layers defines the depth of the network, which is also where the term deep in Deep Neural Network stems from.

The neurons compute the product between the input value, x, and the weight, w and adds a bias, b. This value is passed as the result through an activation function, $\sigma(\cdot)$. The choice of activation function differs depending on the implementation. Two widely used activation functions are the Rectified Linear Unit, ReLU, which strips the output of negative values and the sigmoid function which scales the output between zero and one.

For an input x into a neuron, the intermediate value z is computed and the output of the neuron is y as:

$$z = w \cdot x + b \tag{2.2}$$

$$y = \sigma(z) \tag{2.3}$$

Where w and b are the so called *learnable* parameters. In practical applications one neuron often has multiple inputs and the output is then the summation:

$$y = \sigma\Big(\sum_i z_i\Big) = \sigma\Big(\sum_i w_i x_i + b_i\Big) \tag{2.4}$$

Performing the neuron computations from an input, through the network, to the output from the networks is called *forward propagation*. A simple feed forward neural network is visualized in Figure 2.6.

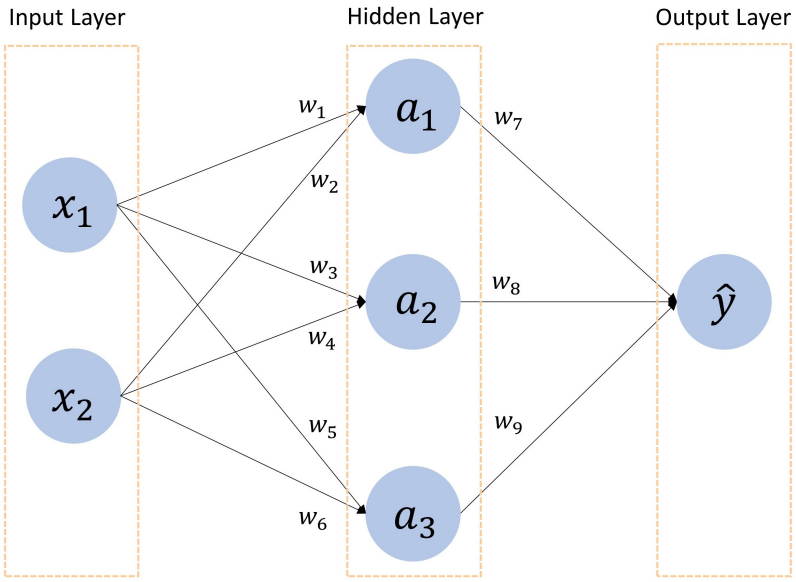


Figure 2.6: A simple feed forward neural network with two inputs, one hidden layer with three neurons and one output

Where a_i is the output from the ith neuron in the hidden layer of the neural network. Using Eq. 2.4 the forward propagation through the network is computed as:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \sigma_1((x_1 w_1 + b_1) + \sigma_1(x_2 w_2 + b_1)) \\ \sigma_2((x_1 w_3 + b_2) + \sigma_2(x_2 w_4 + b_2)) \\ \sigma_3((x_1 w_5 + b_3) + \sigma_3(x_2 w_6 + b_3)) \end{bmatrix} \tag{2.5}$$

$$\hat{y} = \sigma(a_1 \cdot w_7 + a_2 \cdot w_8 + a_3 \cdot w_9 + b)) \tag{2.6}$$

2.2.2.1 Gradient Descent Learning and Back-Propagation

The network is said to learn the mapping θ in Eq. 2.1. One way of achieving this is through gradient descent. With a cost function $J(\cdot)$, such as the cross correlation product, the optimal mapping θ^* is the mapping that satisfies

$$\theta^* = \arg\min_{\theta} J(\theta) \tag{2.7}$$

The iterative approach of gradient descent can be applied to find θ^*.

$$\theta_{k+1} = \theta_k - \alpha \frac{\partial J}{\partial \theta_k} \tag{2.8}$$

Where α is a small positive number, known as the learning rate. Given the neural network in Figure 2.6, back-propagation is applied to find the partial derivative $\frac{\partial J}{\partial \theta}$

and the mapping θ is each individual weight. Using the quadratic cost function:

$$J = \frac{1}{2n} \sum_{x} ||y(x) - \hat{y}(x)||^2 \tag{2.9}$$

Where $y(x)$ is the correct expected output, $\hat{y}(x)$ is the output from the neural network and n is the total number of samples. Back-propagation allows for calculating the partial derivative:

$$\frac{\partial J}{\partial w_k} \tag{2.10}$$

for any weight w_k. Setting the input from a neuron k^{th} to a proceeding neuron j^{th} as:

$$z_j = w_k x_k + b_k \tag{2.11}$$

And the output from the j^{th} neuron as:

$$a_j = \sigma(z_j) \tag{2.12}$$

The chain rule yields:

$$\frac{\partial J}{\partial w_k} = \frac{\partial z_j}{\partial w_k} \cdot \frac{\partial a_k}{\partial z_j} \cdot \frac{\partial J}{\partial a_k} \tag{2.13}$$

Hence, by using the chain rule, the partial derivative $\frac{\partial J}{\partial w_k}$ can be computed for use in the gradient descent learning. Depending on the activation functions used the chain rule can yield problems. Scaling outputs between zero and one, using the sigmoid function, can in a large network yield small gradients in which the problem with vanishing gradients can arise. Similarly can large gradients lead to the opposite problem, exploding gradients, where large gradients are multiplied together.

2.2.2.2 Convolutional Neural Network

Modifying the feed forward neural network to involve convolution rather than multiplication yields the convolutional neural network. The convolution operation allows for processing data of grid-like structure and with high spatial correlation, such as an image. The convolutional neural network has proven to be successful in image classification challenges such as the ImageNet Large Scale Visual Recognition Challenge [25]. The discrete time convolution operation between a real valued number f and a probabilistic density function g:

$$(f * g)(t) = \sum_{-\infty}^{\infty} f(a)g(a - t) \tag{2.14}$$

In an application where convolution is applied on an image Eq. 2.14 is expanded to include two axes, assuming the image black and white and therefore is two-dimensional.

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \tag{2.15}$$

Where I is a two dimensional matrix with i number of rows and j number of columns, such as an image, and K is the kernel applied in a sliding window manner and with a $m \times n$ kernel size. In a convolutional neural network the kernel would be the trainable weights of the network. Note that a color image, RGB image, would require an additional axis.

2.2.3 You Only Look Once

YOLO is a deep learning based approach to object detection in images utilizing the one-staged approach in a single neural network to predict the bounding boxes and the corresponding classes, taking the full image as an input. The basic concept of YOLO is to apply a grid cell of size $S \times S$ onto an image, as can be seen in Figure 2.7. If the center of an object falls into one grid cell this cell is responsible for detecting that object. For each grid cell YOLO predicts B bounding boxes and each of the bounding boxes has 5 + C attributes. The attributes describe the center coordinates (x and y), dimensions (height and width), objectness score and C class confidences for each bounding box. The dataset with humans and AGVs has 2 classes and when using this dataset, C is equal to 2.

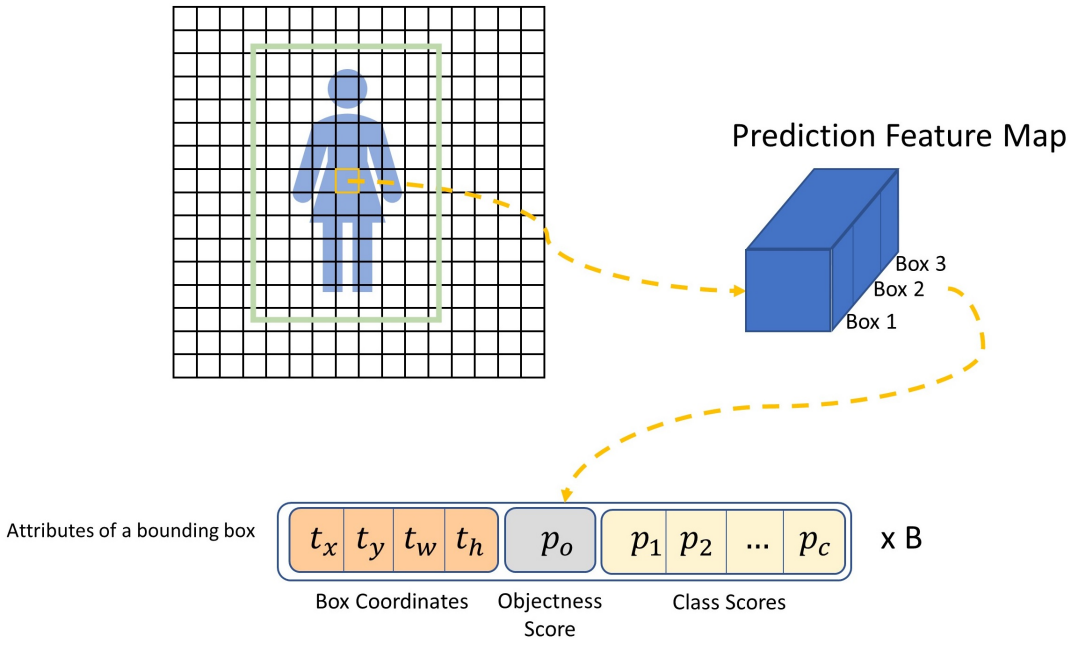


Figure 2.7: Visualization of the three bounding boxes that is predicted for the center cell and it's attributes

The four coordinates that the network predicts for each bounding box are

$$\text{Coordinates:} \begin{cases} \text{Center coordinates of bounding box: } t_x, t_y \\ \text{Width coordinate of bounding box: } t_w \\ \text{Height coordinate of bounding box: } t_h \end{cases} \tag{2.16}$$

And the objectness score p_o which is the probability between zero and one that the bounding box contains an object along with the confidence score for every class as can be seen in Eq. 2.17.

$$\text{Box confidence score for each class: } \begin{cases} p_1 \\ p_2 \\ \vdots \\ p_c \end{cases} \tag{2.17}$$

The sizes of the bounding boxes are determined by predicting the offset from the anchor boxes, see Figure 2.8. The YOLO algorithm uses three anchor boxes each of three different scales. K-means clustering is performed with IoU values as the distance measure between anchor boxes and the ground truth. Cluster centroids and the center coordinates are propagated through a sigmoid function $\sigma(\cdot)$ scaling the input between zero and one.

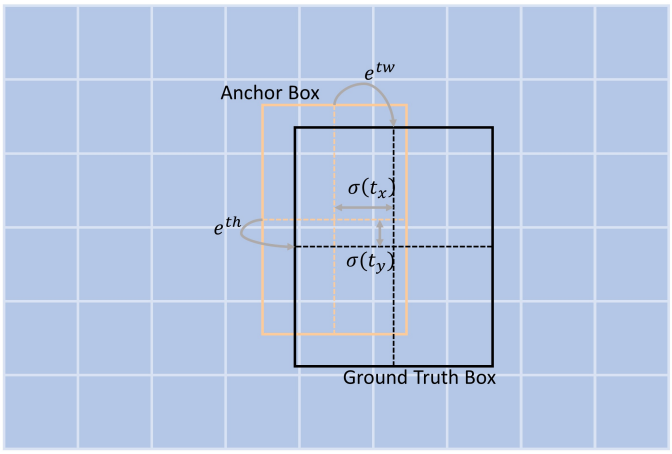


Figure 2.8: Visualization of how the bounding boxes are calculated as an offset from the anchor box in YOLO

The predicted bounding boxes are described by their center points, width and height. The parameters are calculated using the networks output.

$$\begin{bmatrix} b_x \\ b_y \\ b_w \\ b_h \end{bmatrix} = \begin{bmatrix} \sigma(t_x) + c_x \\ \sigma(t_y) + c_y \\ p_w e^{t_w} \\ p_h e^{t_h} \end{bmatrix} \tag{2.18}$$

Where

$$\begin{cases} b_x, b_y, b_w, b_h \text{ are coordinates of the predicted bounding box} \\ t_x, t_y, t_w, t_h \text{ are outputs from the network after training} \\ c_x, c_y \text{ are the cells top left corner of the anchor box} \\ p_w, p_h \text{ are the anchor boxes width and height} \end{cases} \tag{2.19}$$

Confidence score for each bounding box is calculated as

$$\text{Confidence score} = p_o \cdot IoU_{\text{pred}}^{\text{truth}} \tag{2.20}$$

Where $IoU_{pred}^{\text{truth}}$ is the intersection over union, described in Figure 2.9, of the predicted box and the ground truth box. A low confidence score indicates that there is no object in the cell. As the last step, YOLO applies Non Maximum Suppres-

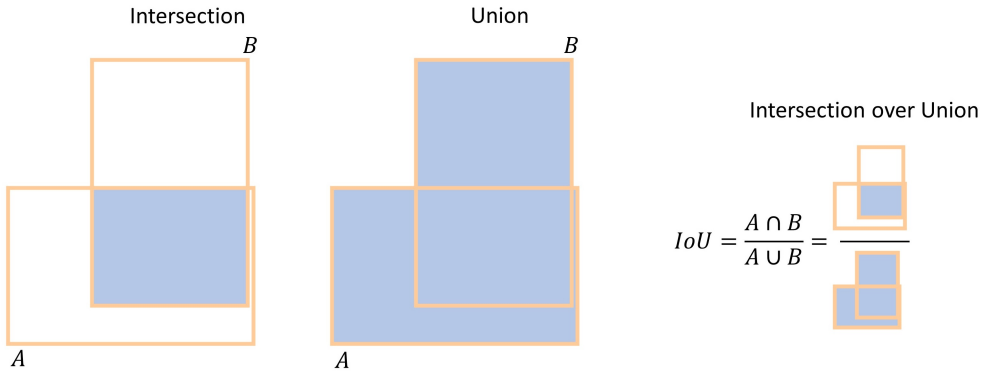


Figure 2.9: Visualisation of the calculation for Intersection over Union

sion, NMS, to get rid of overlapping bounding boxes and bounding boxes that do not contain any objects [26]. Overlapping bounding boxes for the same object is reduced to the single bounding box with the highest confidence score.

The output from the YOLO algorithm is then a list of bounding boxes for the detected objects along with the predicted classes and confidence score.

2.2.3.1 History of YOLO

There have been multiple versions of the YOLO algorithm since it was first released in 2016 [27]. In 2017 YOLOv2 was released [28] and the next year, 2018, YOLOv3 was released [29]. All with increased performance than its predecessor. In April 2020 the 4th version of YOLO, YOLOv4, was released [30]. This version use "bag of freebies", BoF, to improve the accuracy without increasing computational cost during implementation. Instead the training is heavier to compute. The purpose is to increase the variability of the input images using distortion, which increases the robustness of the model. Two commonly used distortions are photometric distortions, such as changing brightness or contrast, and geometric distortions which could be random scaling or rotation. YOLOv4 also use several "bag of specials", BoS, that only increase the hardware cost by a small amount but significantly improve the detection accuracy. These are post processing methods and plug in methods that give a trade off between the inference cost and the detection accuracy. One that is commonly used is Non Maximum Suppression, NMS [31].

2.2.3.2 You Only Look Once fifth version

In this thesis the most recent YOLO version is the 5th, released in June 2020. This version of YOLO is written in PyTorch instead of the Darknet framework and was released by Ultralytics [32]. YOLOv5 comes in four different sizes, small, medium, large and extra large. The small version, YOLOv5s, will be used in this thesis.

YOLOv5s is made up of three parts named the model backbone, neck and head. Parts of the network are visualized in Figure 2.12. The backbone is the base classification model that the object detection is based on. The backbone is based on DenseNet that was designed to connect layers in convolutional neural networks to reduce the vanishing gradient problem, described in Section 2.2.2.1, when doing back-propagation. Here a CNN form image features from the input image. Cross Stage Partial Networks, CSPNet, is used as a backbone that can achieve a richer gradient combination while reducing the amount of computation by splitting the feature map of the base layer into two parts. This makes the gradient flow propagate through different network parts and reduce memory cost since the feature map is split and only a part of the feature map is propagated through the heavy DenseNet. The feature map is then concatenated [33]. Figure 2.10 is a visualization of the Cross Stage Partial Network.

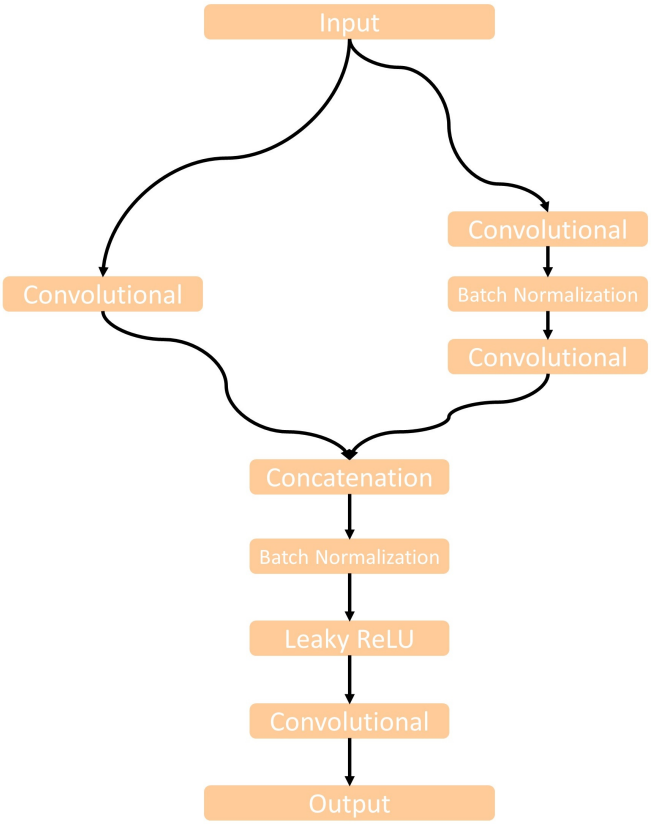


Figure 2.10: Visualisation of the split in Cross Stage Partial Network in the model backbone to reduce the vanishing gradient problem

The neck is the part of the object detection model that forms features from the backbone. It is a series of layers in the network to pass combined image features forward to prediction. This is used to help the model identify the same objects in different scales by generating feature pyramids. YOLOv5 uses Path Aggregation Network, PANet, as a neck to get Feature Pyramid Networks, FPN, that can be seen in Figure 2.11. FPN composes of a bottom-top pathway like the convolutional network feature extraction where the spatial resolution, the number of pixels utilized in the construction of a digital image, decreases higher up in the layers. FPN also provides a top-down pathway to construct high resolution layers from low spatial resolution layers [34]. Due to all the downsampling and upsampling between the layers, the locations of the objects may diverge in the reconstructed layers. To avoid this problem lateral connections between the reconstructed layers are added. This helps the detector predict the correct locations of the objects.

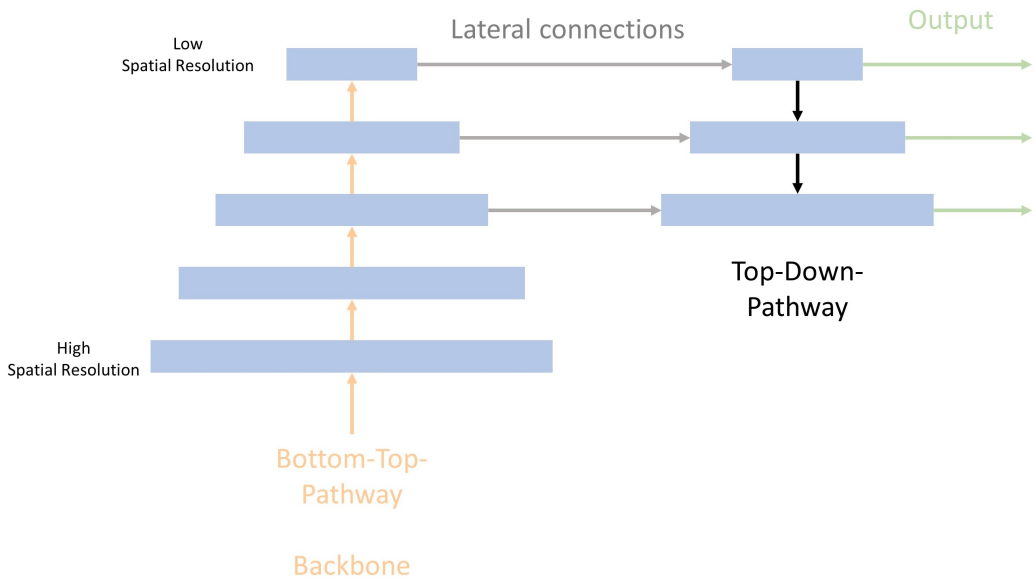


Figure 2.11: Visualisation of Feature Pyramid Network in the neck of YOLOv5s

Model Head is the part in the detector that has been customized with anchor boxes based on the dataset. It consumes features from the neck and applies anchor boxes of the features. At last, it creates the bounding boxes and predicts the class of the detected object.

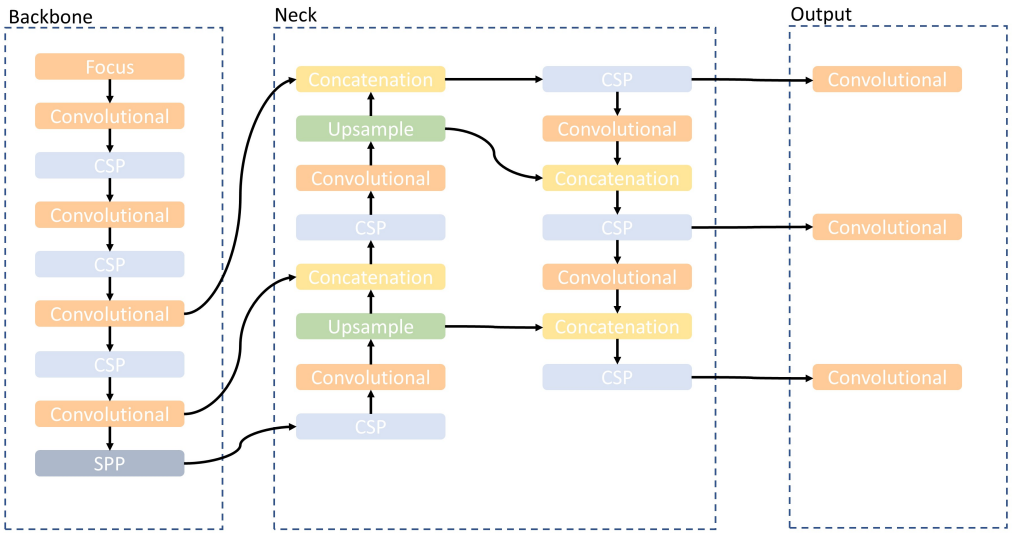


Figure 2.12: YOLOv5 network architecture visualisation

2.3 Object tracking

Object tracking aims at detecting relevant objects, assign identification to each object and maintain both the detection and the identification of the objects through out a video stream, thus capturing the track of the object. The technique of object tracking has a variety of uses such as in surveillance and traffic control, among others.

This thesis dealt with tracking of multiple objects which can be a complicated task. There are common methods related to multi object tracking such as Joint Probabilistic Data Association which is a method for, in each time frame, associate detected measurements with existing objects using a joint probabilistic score [35]. Another approach to multi object tracking is Global Nearest Neighbor (GNN). The basic GNN approach attempts to find and to propagate the single most likely hypothesis at each scan. In a cluttered environment, the received measurements may not all arise from the real targets [36].

Rather than considering more complicated scenarios this thesis will use the Kalman filter as a single object tracker and expand it to multi object tracking by instantiating a filter for each measure object.

Typically sensor measurements are used to determine the parameters of objects to be tracked. These parameters include position but can be expanded to velocity, type of object and identity depending on the implementation case. The parameters in this thesis are the state of position.

Typically a motion model is used to describes how the target object might change given estimated states, such as position and velocity. Filtering and data association involves prior information about the objects, object dynamics and evaluation of the different hypotheses. These methods allow tracking of objects and paths that are occluded for some frames. Kalman filter is one example of a filter for linear or nonlinear motions that uses a series of noisy measurements observed over time and produces an estimate of the systems states. The association between detected objects and tracks can increase in complexity when objects move fast, when there are multiple objects to track or when the tracked objects change orientation over time.

2.3.1 Linear Time Invariant systems

A Linear Time Invariant, LTI, System is a system that produces an output signal from an input signal subject to the constraints of linearity and time-invariance. The response $y(t)$ of an arbitrary input $x(t)$ can be found using convolution in Eq. 2.21.

$$y(t) = x(t) * h(t) \tag{2.21}$$

Where $h(t)$ is called the systems impulse response, in other words, the reaction of any dynamic system in response to some external change. An LTI system is linear which means that if the input $x(t)$ is scaled by a value α then the output $y(t)$ is also

scaled with the same value α. A linear system satisfies the superposition principle which means that if the input is a sum of two signals, the output signal will be the sum of the two original output signals. Since the system is time invariant it is ensured that if the input signal is shifted in time, the output signal will be shifted in time [37].

2.3.1.1 Kalman filter

The Kalman filter is an algorithm used to estimate the state of an LTI system based on previous states. The usage of the Kalman filter in this project suits two purposes, smoothing out the estimated position and associate measurements through time by comparing the distance between the predicted state and the measured state. The association provides the possibility to associate objects through multiple frames. If an object is given an identification upon detection, the identification should remain the same when the object moves. The following introduction on Kalman filtering is from Simo Särkää's book on Bayesian filtering and smoothing [38].

$$\text{Prior distribution: } p(x_{k-1}|y_{1:k-1}) \tag{2.22}$$

$$\text{Prediction distribution: } p(x_k|y_{1:k-1}) \tag{2.23}$$

$$\text{Posterior distribution: } p(x_k|y_{1:k}) \tag{2.24}$$

The linear discrete time Kalman filter estimates the state of a system described by equations 2.25 - 2.31. For state vector $\mathbf{x}_k$ and observation $\mathbf{y}_k$:

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{q}_{k-1} \tag{2.25}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{r}_k \tag{2.26}$$

Where $\mathbf{A}$ is the transition matrix and $\mathbf{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the process noise normally distributed with state noise covariance matrix $\mathbf{Q}$. $\mathbf{H}$ is the measurement matrix, $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise normally distributed with measurement noise covariance matrix $\mathbf{R}$. The choice of states, $\mathbf{x}$, differ depending on the filtering case.

The Kalman filter predicts the posterior state and corrects the prediction using the measurement update, recursively. The prediction step consists of computing the predicted state estimate, $\hat{\mathbf{x}}$, and the predicted covariance matrix distribution, $\mathbf{P}$, as:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_{k-1}\hat{\mathbf{x}}_{k-1|k-1} \tag{2.27}$$

$$\mathbf{P}_{k|k-1} = \mathbf{A}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \tag{2.28}$$

The update step corrects the state and covariance estimate using measurements:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \tag{2.29}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T \tag{2.30}$$

Where

$$\text{Filter components} = \begin{cases} \text{Kalman gain: } \mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \\ \text{Innovation: } \mathbf{v}_k = \mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \\ \text{Innovation covariance: } \mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \end{cases} \tag{2.31}$$

A key aspect in tuning the Kalman filter is to select the Signal to Noise Ratio, SNR.

$$\text{SNR} = \frac{\|\mathbf{Q}\|}{\|\mathbf{R}\|} \tag{2.32}$$

A large $\mathbf{Q}$ results in a large SNR $\rightarrow$ a quickly adapting filter that relies more on the new data than the predictions
A small $\mathbf{Q}$ results in a low SNR $\rightarrow$ the measurements are less important and the model relies more on the predictions.

2.3.1.2 Unscented Kalman Filter

The equations described in Section 2.3.1.1 are defined under the assumption that the system is linear. This is not always the case and in this thesis the paths of humans and AGVs can not be assumed to be linear at all times. When the state transitions and observation models are nonlinear the prediction and update functions are nonlinear and therefore a standard linear Kalman filter gives an inaccurate result. Therefore, a non-linear Kalman filter is to be implemented. The following brief introduction on non-linear kalman filtering is based on Särkkä [38] and Julier and Uhlmann [39] with a few adjustments.

The Unscented Kalman filter, UKF, is a method for calculating statics of a random variable $\mathbf{x}$ with dimension n that undergoes a nonlinear transform. The sampling technique, unscented transform, is to pick a set of sample points, sigma points, around the mean. A set of sigma points are chosen so their sample mean is $\bar{\mathbf{x}}$ and sample covariance is $\mathbf{P}_{xx}$.
The sigma points are then propagated through the nonlinear function $\mathbf{y} = g(\mathbf{x})$ and each point turns to transformed points in a cloud with a new mean and covariance $\bar{\mathbf{y}}$ and $\mathbf{P}_{yy}$.
A vector χ of 2n + 1 with sigma vectors χ_i and weights W_i is formed to calculate the statistics of $\mathbf{y}$. The sigma points are formed as:

$$\chi_0 = \bar{\mathbf{x}} \tag{2.33}$$

$$\chi_i = \bar{\mathbf{x}} + \left(\sqrt{(n+\lambda)\mathbf{P}_{xx}}\right)_i \quad \text{i = 1,...,n} \tag{2.34}$$

$$\chi_i = \bar{\mathbf{x}} - \left(\sqrt{(n+\lambda)\mathbf{P}_{xx}}\right)_{i-n} \quad \text{i = n+1,...,2n} \tag{2.35}$$

κ	$\geq$	0
α	$\in$	(0,1]
β	$=$	2

Table 2.1: Parameter suggestions in Unscented Transform

With

$$\lambda = \alpha^2(n + \kappa) - n \tag{2.36}$$

The parameter λ is a scaling parameter that determines the distance between the mean and the sigma points. Where α, κ and β are the picked to spread the sigma points. Typical choices of parameters are shown in table 2.1.
The sigma points are propagated through a dynamic model yielding $\hat{\chi}$.

$$\hat{\chi} = g(\chi) \tag{2.37}$$

The parameters for calculating the filter components are then defined as:

$$W_m^{[0]} = \frac{\lambda}{(n + \lambda)} \tag{2.38}$$

$$W_c^{[0]} = W_m^{[0]} + (1 - \alpha^2 + \beta) \tag{2.39}$$

$$W_m^{[i]} = W_c^{[i]} = \frac{1}{2(n + \lambda)} \quad \text{i = 1,...,2n} \tag{2.40}$$

The parameters α, β and λ are picked and chosen to give a good estimate of the Gaussian distribution. With sigma points and weights the transformed mean and covariance can be computed.

$$\bar{x} = \sum_{i=0}^{2n} W_m^{[i]} \hat{\chi}_i \tag{2.41}$$

$$\boldsymbol{P} = \sum_{i=0}^{2n} W_c^{[i]} (\hat{\chi}_i - \bar{x})(\hat{\chi}_i - \bar{x})^T + \boldsymbol{R} \tag{2.42}$$

With parameters for states, $\bar{x}$, state covariance, $\boldsymbol{P}$ and
Eq. 2.29-2.31 can be used for the update step.

2.4 Mapping

Mapping, in this case, refers to the process of projecting points from one view into another view. An example could be a car in a parking lot seen from a surveillance camera. Mapping in this case could be to project the cars position onto a 2-dimensional map of the parking lot. To achieve this, the relationship between the arbitrary view from the camera and the top-down view of the map needs to be established, this relationship is called the homography [40].

2.4.1 Homographies

The homography is the matrix that describes the linear relationship between points in planes. Typically, these points are mutual keypoints found in two or more images. Points in the first view $\boldsymbol{p}$ are related to points in the second view $\boldsymbol{p}'$ by the homography $\boldsymbol{H}$ [41].

$$\boldsymbol{p} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \text{ and } \boldsymbol{p}' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{2.43}$$

$$\boldsymbol{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tag{2.44}$$

$$c\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.45}$$

The homography matrix $\mathbf{H}$ can be estimated with random sample consensus, RANSAC. With a bounding box and a known homography the projected bounding box can be computed by mapping each corner point of the bounding box using the homography, see Figure 2.13.

2.4.1.1 RANSAC

Random Sample Consensus, RANSAC, is a robust homography estimation algorithm. Robust in the sense that it can handle outliers, keypoints that are falsely detected or matched. The algorithm is defined in A, Agarwal et al. [42] and is described below, slightly rewritten.

Algorithm 1 RANSAC

Result: Homography matrix

1. Sample a subset of points from the matches
2. Compute the homography for the subset
3. Compute the number of inliers for the complete set of matches
4. Repeat until the number of inliers are satisfyingly high

An inlier is found if the euclidean distance, the length of a line segment between two points $d(\boldsymbol{H}p, p')$, is below a given threshold, t.

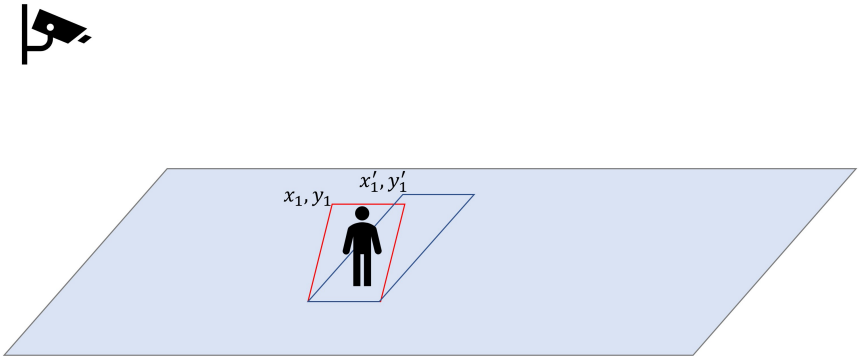


Figure 2.13: Mapping of a bounding box. Captured bounding box from camera in red and projected bounding box on the plane in blue

3
Data Gathering

The thesis required data in the form of synchronized videos from cameras in a warehouse. The thesis had a research approach and a need to change camera angles and positions based on experiments. Considering this it was found infeasible to work towards acquiring real life data from an actual physical warehouse and simulation of data was decided on.

3.1 Simulation

The simulation was carried out in Blender [43] by modeling mock up scenes of a warehouse. Two scenes were created, one large (50m x 30m) to evaluate the tracking performance with multiple AGVs and humans to detect, and one small (26m x 26m) to evaluate the position performance with a single AGV to detect. Renders of the warehouse scenes are visualized in Figure 3.1 - 3.2.

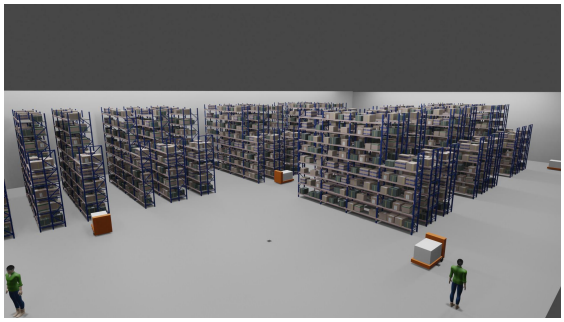


Figure 3.1: Large warehouse simulation environment

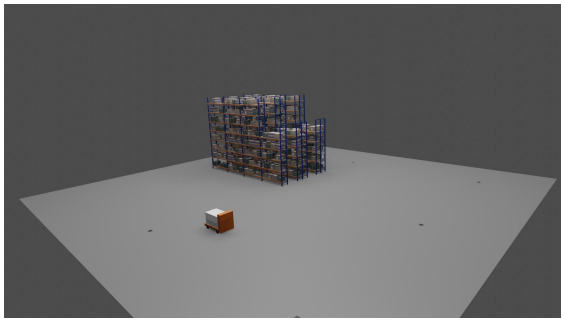


Figure 3.2: Small warehouse simulation environment

Both environments used third party models for shelves and humans [44][45]. The AGVs were modelled by the thesis group as well as floor and walls. The scenes were lit up by artificial light racks placed in the ceiling to resemble the lighting of an actual warehouse. The corresponding overview maps for the environments are visualized in Figure 3.3 - 3.4. The AGVs and humans were animated to have paths within the scene. Cameras were positioned in the roof of the simulated warehouse and renders of videos were created to be used as data in the system.

The dynamical simulation was performed by specifying points for each moving objects to travel to. The point, together with a timestamp on when the object should be present at the different points made up the path for the object. These paths were all made with the simulation software in Blender and yielded a scene with moving objects that could be captured through the cameras in a render. The animated paths gave the ground truth for evaluation use and the videos composed the actual data.

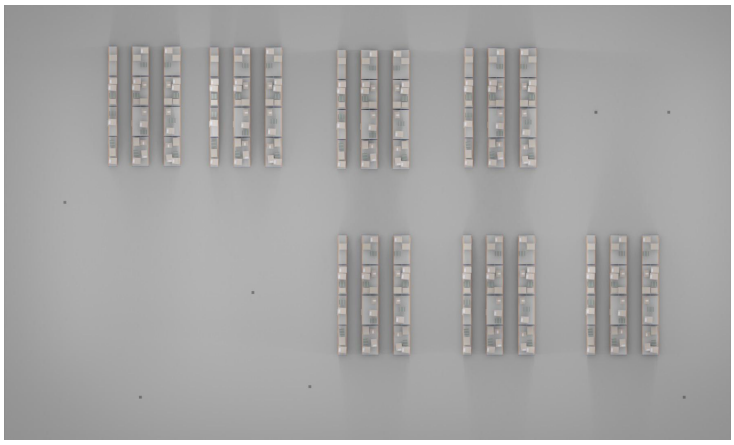


Figure 3.3: Top-down overview of the large warehouse environment

Figure 3.4: Top-down overview of the small warehouse environment

3.2 Data labelling

Training the object detection algorithm requires annotated ground truth data. The data needs to be annotated with correct bounding boxes and correct class belonging. The annotation was performed manually using the software LabelImg [46]. Multiple rendered videos from the simulation software Blender were collected and split frame-by-frame. The videos, and in turn the data, captured different views of the scene to assure robustness in the trained model. The annotations were made manually using two classes, *AGV* and *human*, Figure 3.5 is an example of an annotation of an image in the software. This resulted in a text file associated with every image containing information on how many ground truth objects were in the image, as well as the bounding box coordinates for these objects. The accuracy and robustness of the neural network are closely linked to the amount of training data. However, manually annotating thousands of images would be a tedious work. 900 images were annotated to this thesis which resulted in a network with enough accuracy of the detections in the produced videos to be used in the thesis.

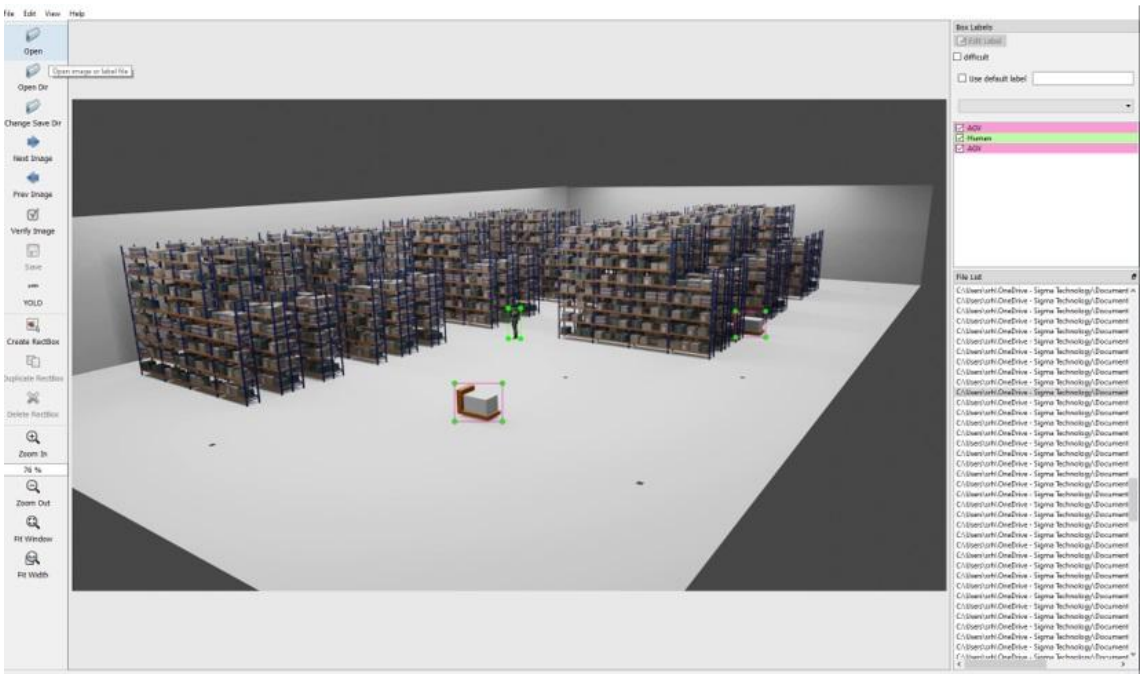


Figure 3.5: Example labelling of one image in Labelimg Software

4
Positioning

The system should be able to estimate the position of objects in a scene. The method for retrieving the position of a detected object was broken down into:

- Transform the bounding boxes according to the respective homography
- Find intersections between transformed bounding boxes
- Compute center point of the resulting polygons from bounding box intersection

Below follows a more in-depth description of the different steps.

4.1 Homography projection

Homographies were computed using OpenCV [47] in Python with manually annotated key points. The annotation of key points was made using visual matching and saving corresponding pixel-space points in vectors. An example of the annotation and matching is shown in Figure 4.1.

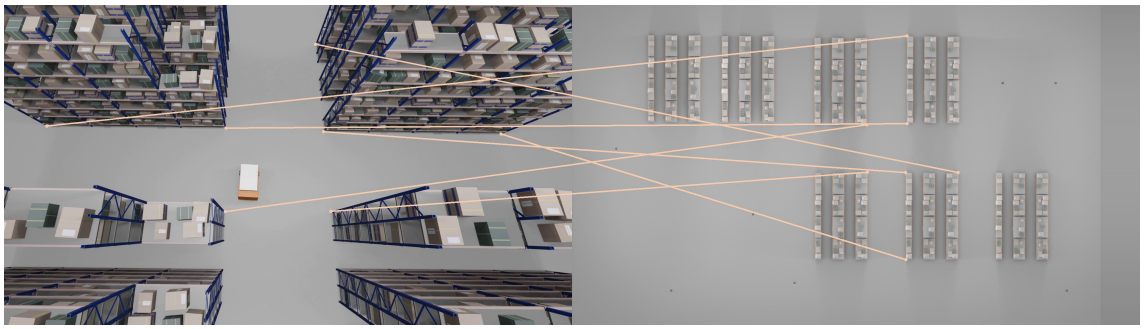


Figure 4.1: Keypoint matches between raw camera view and overview

The matched keypoints were used, together with RANSAC, to find the homography matrix using OpenCV findHomography function. With the computed homography an input image can be warped to fit onto the overview map in Figure 3.3 or Figure 3.4 depending on which warehouse was used. A visualization of the homography warping can be seen in Figure 4.2 and Figure 4.3.

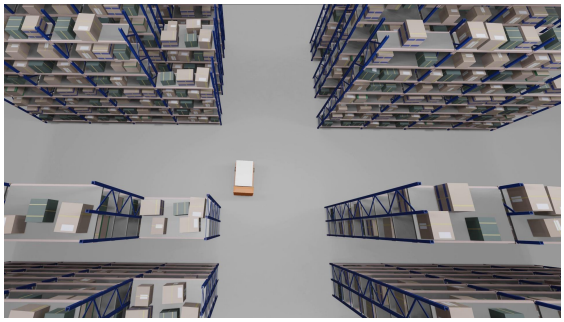


Figure 4.2: Raw view from camera

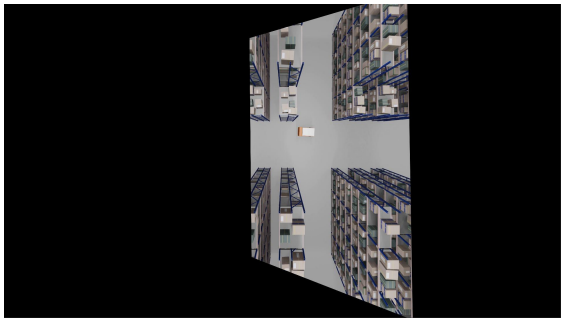


Figure 4.3: Camera view warped according to homography

4.2 Intersecting shadow-triangulation

The thesis dealt with positioning objects in an arbitrary known environment, such as a warehouse. The center point coordinates of an object of interest were the natural point to try to estimate. However, mapping the bounding box of an observed object will produce a shadow like bounding box on the map view and the center point will naturally have an offset depending on the cameras angle. To get an accurate estimate of the center point the thesis proposed an "Intersecting Shadow-triangulation" approach. The intersection between the shadows from multiple views would estimate the actual footprint of the object on the ground and the center point of this footprint would be close to the actual center point of the object. See Figure 4.4 and Figure 4.5. To achieve this, it was necessary to compute which bounding boxes intersect.

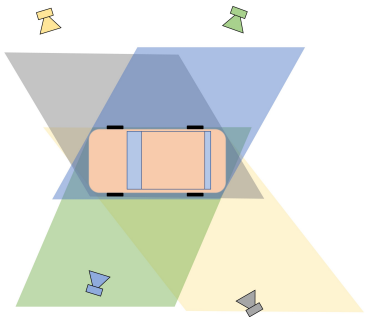


Figure 4.4: Multiple bounding boxes projected onto map

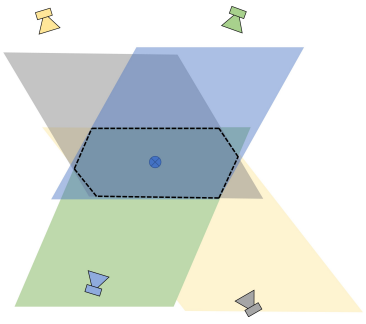


Figure 4.5: Intersection area and center point

The method requires bounding boxes to fully, or almost fully, enclose the object. The objection detection algorithm can detect objects without seeing the entire object, e.g. when the object moves out of the camera, resulting in a smaller bounding box that only partially covers the detected object. Naturally, computing the center point of such a bounding box would offset the object center point heavily. To account for this, a bounding box area threshold was implemented stating that bounding boxes of consideration would need to have a sufficient area. The threshold was set as the approximate size of the object. For an object detected and classified as class *Human*, the threshold was set to 250 and for an object detected and classified as class AGV, the threshold was set to 2500.

4.2.1 Multiple polygon intersection

An algorithm for finding intersections between multiple bounding boxes was developed and implemented for the case where multiple cameras detect the same object. The algorithm uses an intersection ratio matrix to search for multiple intersections. The IoU gives the ratio of the overlap between two bounding boxes. Hence, the IoU matrix is simply the IoU values between all given bounding boxes. A set of N bounding boxes yields a $N \times N$ IoU-matrix. In Algorithm 2 a shorter version of the find intersections algorithm is presented. A longer version can be found in Appendix A 1.

Algorithm 2 Find intersections short version

```
1: for index = (1:nBBoxes) do
2:   if index ∈ available bboxes then
3:     add  index to intersected bbox
4:     remove  index from available bboxes
5:     switcher = row
6:
7:     while available bboxes is not empty do
8:       current index = index
9:       if switcher = row then
10:        while intersecting bboxes exist do
11:          iou_value = max(sorted(IoUMatrix(row)))
12:          index = index of iou_value
13:
14:          if iou_value = 0 then
15:            No overlapping bboxes
16:          end if
17:
18:          if length of index > 1, more that one box overlaps equally then
19:            if iou_value > 0 then
20:              Make one of the iou_values smaller
21:              index = index of iou_value
22:            end if
23:          else
24:            if index in available bboxes then
25:              add index to intersected bbox
26:              remove index from available bboxes
27:            end if
28:          end if
29:        end while
30:        switcher = col
```

```
31:         else if switcher = col then
32:           while intersecting bboxes exist do
33:             iou_value = max(sorted(IoUMatrix(col)))
34:             index = index of iou_value
35:
36:             if iou_value = 0 then
37:               No overlapping bboxes
38:             end if
39:
40:             if length of index > 1, more that one box overlaps equally then
41:               if iou_value > 0 then
42:                 Make one of the overlaps smaller
43:                 index = index of iou_value
44:               end if
45:             else
46:               if index in available bboxes then
47:                 add index to intersected bbox
48:                 remove index from available bboxes
49:               end if
50:             end if
51:           end while
52:         end if
53:       end while
54:     end if
55: end for
```

Given an IoU-matrix the bounding boxes intersecting was determined. The algorithm searches for matches row- and columnwise respectively. If a matching bounding box is found in a row, i.e. the IoU value is greater than zero, the index of this bounding box becomes the starting point in a column search. If a match is found in the column search, it is first determined if the match also intersects with the previously picked bounding boxes before adding it to the list of intersecting bounding boxes. The search continues until no additional bounding boxes can be added to a set of intersected boxes.

In Figure 4.6 an example of the algorithm is shown:

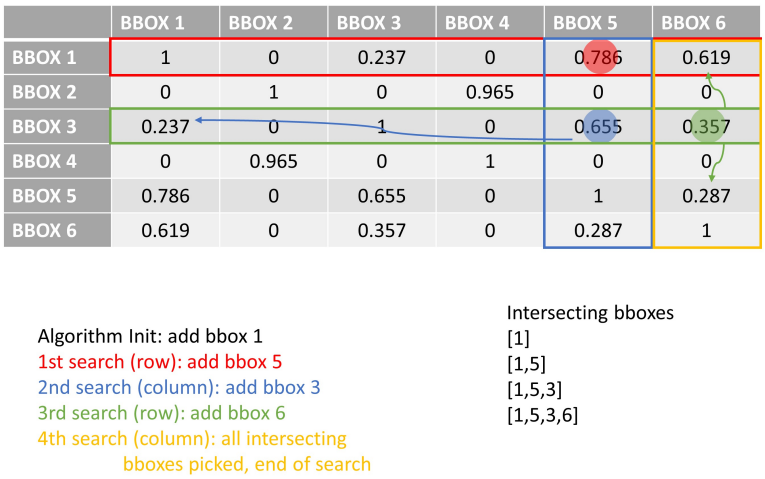


Figure 4.6: Intersection search algorithm example

In the initialization step, the first bounding box in the list would be added as "intersected". The first search is performed along the first row, and the highest IoU value is picked along with the index of this bounding box, in this case the fifth bounding box. The second search is performed column-wise along the fifth column, resulting in picking bounding box number three after checking that bounding box three indeed intersects all previously selected bounding boxes. The algorithm then switches to row search along row three and continues until no further intersecting bounding boxes can be found. When no more intersected bounding boxes are found, one can take the IoU between the found bounding boxes which results in the final bounding box for the detected object.

5
Tracking

Humans can take one look at an image and instantly know which objects are in the image, the positions and how they move. The visual system of a human is fast and accurate. To be able to make a computer act like a human visual system require a lot of work but would allow AGVs to drive without other sensors than cameras and unlock the potential for responsive robotic systems. In this chapter, the project is broken down into different sub tasks to view the workflow from start to a system that detects and tracks AGVs in a warehouse environment only using cameras. The first step is to detect and track interesting objects.

5.1 Parallel computing

In order to achieve real-time running capabilities, information from multiple cameras has to be processed simultaneously, or in parallel. Multithreading is a technique to achieve parallelism in computing. This technique allows concurrent execution of different tasks on a single Central Processing Unit, CPU, by dividing tasks into threads and assigning portions of the processing resources to each thread [48].

5.2 Object detection

To detect the interesting objects, YOLOv5s was used as a detection algorithm. The YOLO-family of detection algorithms was considered to be a well established algorithm with good performance, documentation and a large community for troubleshooting. The latest model, YOLOv5, has the best performance and the small version was considered to be a good trade off between accuracy and computational complexity which vouched for good real time execution performance. Implementation was based on Patty Wu's GitHub repository [49] but with the DeepSORT tracker algorithm removed. In short, DeepSORT tracker uses a neural network to improve the re-identification of tracked objects. The neural network is trained on pedestrians and given that the simulated AGVs were all exactly alike it was considered retraining the network would not lead to a usable tracker. Apart from this, the algorithm was intended to only output measurement in the form of bounding box coordinates and class of the detected objects. The classes used in the object detection part were *AGV* and *Human* and comes from the labeled data described in section 3.2.

5.2.1 Training the object detector

The object detection algorithm was trained on relevant objects, in this case pictures containing the simulated AGVs and humans from Blender, labeled as described in Section 3.2, to make it able to better spot and classify the objects. Different angles and scales of the AGVs and humans in the training pictures were used to make the detector robust against scales and rotations in the videos. The actual training process was carried out using a training notebook, a predefined training environment [50]. The training environment uses the code supplied with YOLOv5 to train the object detector. Training was performed on Cloud GPU and with image size 416 pixels, batch size 16 and hyper parameters as defined in Table 5.1. The training was performed over 1000 epochs.

lr0	0.01
lrf	0.2
momentum	0.937
weight decay	0.0005
warmup epochs	3
warmup bias lr	0.1
box	0.05
cls	0.5
cls pw	1.0
obj	1.0
obj pw	1.0
iou t	0.2
anchor t	4.0
f1 gamma	0.0
hsv h	0.015
hsv s	0.7
hsv v	0.4
degrees	0.0
translate	1.0
scale	0.5
shear	0.0
perspective	0.0
flipud	0.0
fliplr	0.5
mosaic	1.0
mixup	0.0

Table 5.1: Hyperparameters used in training of the YOLOv5 object detection algorithm

During training the Adams optimizer was used with the Binary Cross Entropy loss was to calculate the loss for both bounding box and classification.

5.3 Multi object tracking

Object detection and classification was performed in each camera and the results were fed into the system. A flowchart of the processes in the system at a given time step k could be described as follows:

- Transform the bounding boxes according to the respective homography.
- Find intersections between transformed bounding boxes
- Compute center point of the resulting polygons from bounding box intersection
- Feed center points as measurements into Kalman filter
- Draw filtered positions on map

Below follows a more in depth description of the different steps.

5.3.1 Kalman tracking

The Unscented Kalman Filter was used for tracking and smoothing out estimated states. Individual filters were initialized for all tracked objects and measurements were associated with predicted filter states using the euclidean distance.

A block scheme of the Kalman tracker is visualized in Figure 5.1.

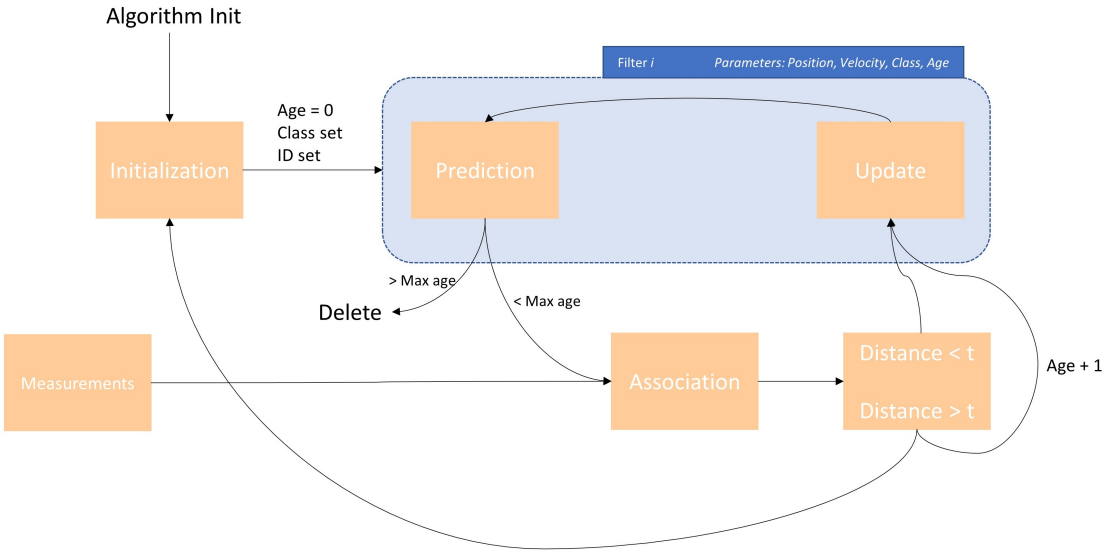


Figure 5.1: Block scheme visualization of Kalman based tracker algorithm with age being a parameter for discarding objects that were lost for a certain time

Measurements were divided into associated and non-associated, where an associated measurement would have to belong to a filter with a predicted state close to the measurement. Non-associated measurements would typically appear when a new target was introduced, resulting in the initialization of a new filter. In a similar way, a filter needs to be deleted in the case a target leaves the scene of sight. To deal with this an age parameter was added to the filter. When a filter was updated without any measurement the age parameter was increased. If the age reaches a threshold the filter was deleted.

5.3.2 Unscented Kalman Filter

The system was described by the discrete state space model in Eq. 5.1 - 5.2.

$$\mathbf{x}_k = F(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1} \tag{5.1}$$

$$\mathbf{y}_k = H\mathbf{x}_k + \mathbf{r}_k \tag{5.2}$$

Parameters of the Kalman filter were set according to Eq. 5.3 - 5.6

$$\underbrace{\begin{bmatrix} x_k \\ y_k \\ v_k \\ \phi_k \\ \omega_k \end{bmatrix}}_{\text{States}}, \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta t\sigma_v^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Delta t\sigma_\omega^2 \end{bmatrix}}_{\text{State noise covariance}} \tag{5.3}$$

Where x_k, y_k are center point coordinates, v_k is velocity, ϕ_k is heading and ω_k is turn rate.

The measurement provided a measured center point for an object. The measurement matrix, converting a state into a measurement, was hence:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.4}$$

Yielding,

$$\mathbf{y}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}_k + \mathbf{r}_k \tag{5.5}$$

The humans and AGVs were assumed to be moving with a constant velocity, producing the motion model:

$$\mathbf{x}_k = F(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1} = \begin{bmatrix} x_{k-1} + 2\frac{v_{k-1}}{\omega_{k-1}} \sin\left(\frac{\omega_{k-1}\Delta t}{2}\right) \cos\left(\phi_{k-1} + \frac{\omega_{k-1}\Delta t}{2}\right) \\ y_{k-1} + 2\frac{v_{k-1}}{\omega_{k-1}} \sin\left(\frac{\omega_{k-1}\Delta t}{2}\right) \sin\left(\phi_{k-1} + \frac{\omega_{k-1}\Delta t}{2}\right) \\ v_{k-1} \\ \phi_{k-1} + \Delta t\omega_{k-1} \\ \omega_{k-1} \end{bmatrix} + \mathbf{q}_{k-1} \tag{5.6}$$

Where Δt was set as the update frequency of the cameras, or the inverse of the rendered frames per second, FPS, 1/24.

On initialization, one filter per measurement was set up. For each new set of measurements, the measurement with the closest euclidean distance was picked for the update step. If no measurement could be associated with a certain tracker, the update step was to run without measurement input and an addition to the age parameter was made. The unassociated measurement yielded the initialization of new trackers. Implementation of the Kalman filter was achieved using Filterpy [51] with modification to fit the thesis, including the addition of ID, class and age parameters.

5.4 Safety functionalities

Trajectory prediction refers to techniques for computing a predicted trajectory. This was used with two different approaches in this project. By computing safety zones that depend on the direction and the velocity of the object and a heatmap that changes with increasing and decreasing movement in the area. Those two approaches were created and tested but since the AGVs are only simulated in pre-defined trajectories and not able to change direction or speed in the simulation, no evaluation of the trajectory prediction was made.

5.4.1 Velocity calculation

The velocity was, together with the position, one of the desired output parameters of the system according to Section 2.1.3. Equation 5.7 was used for velocity calculation. The algorithm stored the last observed position and corresponding frame number and calculated the difference in relation to the current position and frame number. In a scene with perfect coverage, where no occlusion occurs, the frame numbers are always trailing. If there was occlusion, keeping the last observed frame number allows for a linear estimation of the velocity during occlusion.

This results in a linear velocity estimation if the target is occluded for a number of frames.

$$\text{Pixels per Frame} = \frac{(x,y)_k - (x,y)_j}{\text{frame}_k - \text{frame}_j} \tag{5.7}$$

Where k is the current time step and j is the last known time-step.

5.4.2 Safety zones

A safety zone concept was implemented for crash avoidance. This technique was inspired by a system from SICK AG [52] which uses laser scanners for distance measuring and applies threshold for vehicle interaction when an obstacle is found. Within an outer set of thresholds the vehicle should slow down if an obstacle is detected within the inner set of the threshold. See Figure 5.2 The thresholds could either be set statically or dynamically, in which a greater velocity yield a larger safety zone. The dynamic change in safety zone size would better represent the longer braking distance for a vehicle travelling with increased velocity.

The implementation was made dependant on the velocity in x- and y-direction determining the major direction of the vehicle and in turn the safety zone. Conditional statements defined eight cases, straight traveling either forward or back and in left and right direction as well as 45° travelling in four directions. Two safety boxes were drawn around the estimated center point for the inner zone, red, and an outer zone, orange. In addition, the velocity of the vehicle resulted in a larger zone in front of the vehicle during travel.

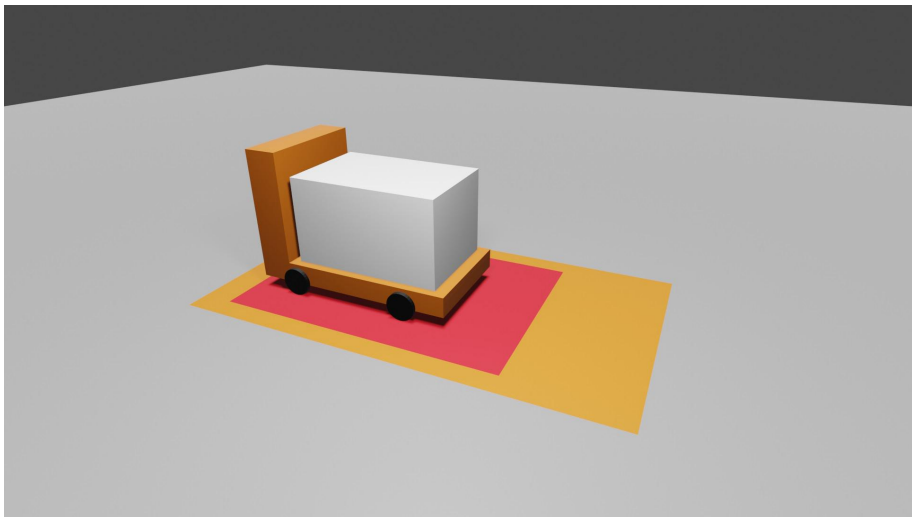


Figure 5.2: Simulated AGV with two level safety zone

5.4.3 Heatmap

A heatmap was implemented. Implementation was based on saving snapshots of the scene for a set number of time steps, removing the oldest snapshot as time progressed. Positions of vehicles and humans were represented using an intensity distribution, to account for uncertainty in the absolute position. The algorithm took as input arrays of position and class belonging and the intensity distribution could hence be dependant on the type of object, where an AGV rendered a larger distribution than a human.

The heatmap catches and registers movement by AGVs and humans in the scene and saves these a certain number of time steps. The heatmap serves as the main source of information regarding what dynamical obstacles surround a certain AGV. This would substitute the use of e.g. LiDAR onboard the vehicle for determining awareness of the surroundings. An example of the heat map can be seen in Figure 5.3-5.4.

The heat map is supposed to function as a safety measure. The size of the safety zones was made dependant on the heat map in the trafficked area would be assigned a greater safety zone.

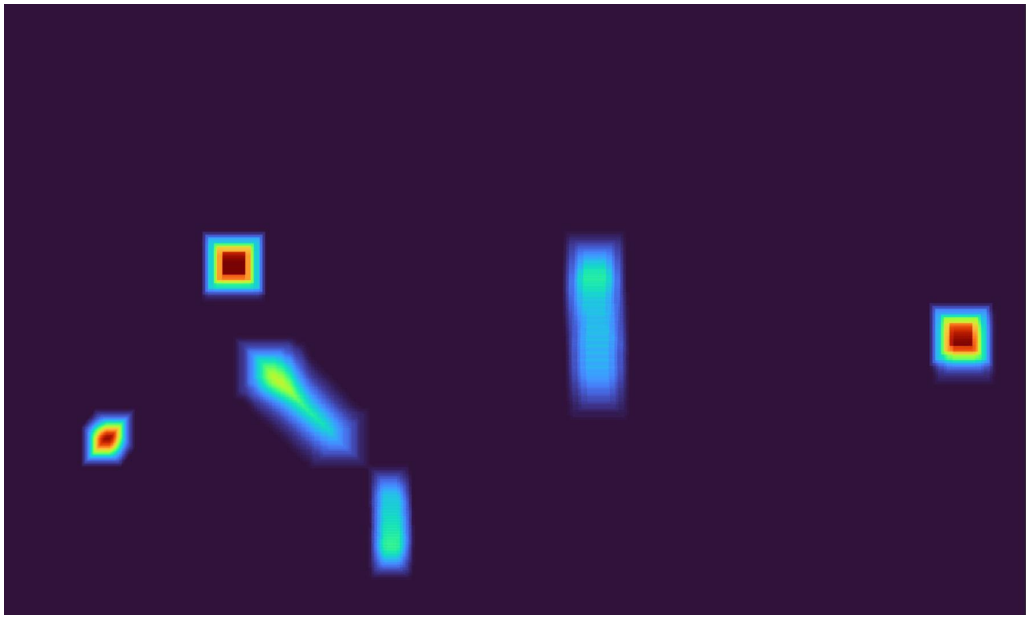


Figure 5.3: Heat map view at time instance k

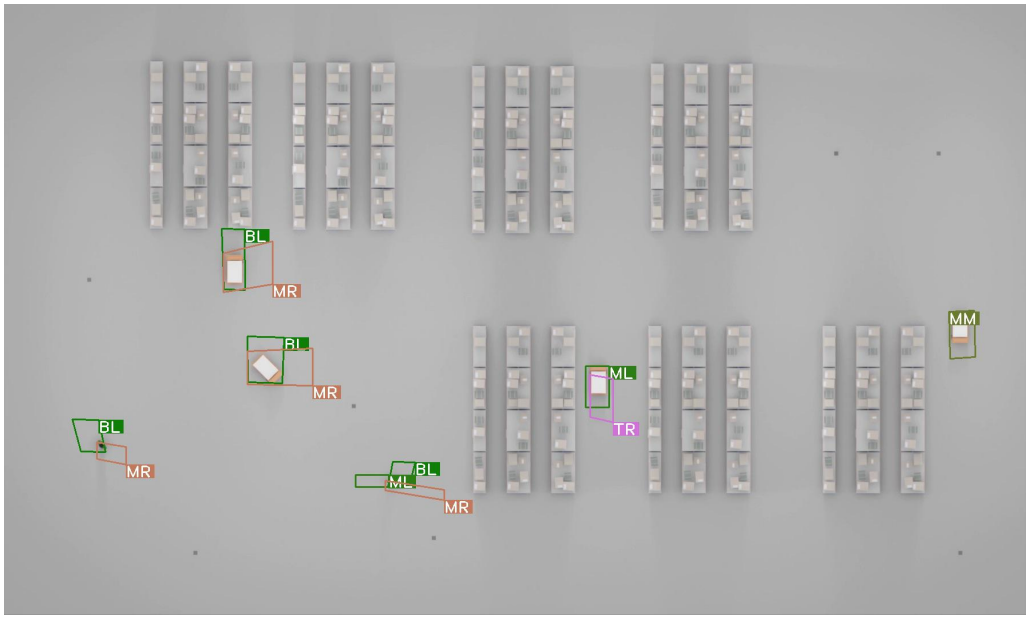


Figure 5.4: Corresponding overview time instance k with mapped bounding boxes from different camera angels

6
Results

Evaluation and testing of the different subsystems were conducted to motivate the methods used and support the proof of concept claim. The system should provide positioning and tracking with minimal identity switches, correct detection and high positioning accuracy.

6.1 System verification and validation

The results were gathered for the systems ability to perform two main tasks, namely positioning and tracking. The positioning performance was evaluated compared to the ground truth position from the simulation and the tracking performance was evaluated in the form of the number of correct detections and the number of identity switches.

6.1.1 Positioning evaluation

Evaluation of the systems positioning ability was based on the ground truth acquired from the simulation part in Section 3.1. An evaluation track was created, see Figure 6.1. The evaluation simulation included a single AGV with cameras strategically positioned to achieve a good positioning performance. By comparing the mapped position and time instance t with the true position at the same time instance a value of the positioning error at time instance t for object k could be calculated as:

$$e_t^k = \sqrt{(x_{true,t}^k - x_{pred,t}^k)^2 + (y_{true,t}^k - y_{pred,t}^k)^2} \tag{6.1}$$

The total mean average error for AGV k is then:

$$e^k = \frac{\sum_{t=1}^{N} \sqrt{(x_{true,t}^k - x_{pred,t}^k)^2 + (y_{true,t}^k - y_{pred,t}^k)^2}}{N} \tag{6.2}$$

Here $x_{pred,t}^k$ and $y_{pred,t}^k$ are the x- and y-position of the center point in the predicted bounding box k, and $x_{true,t}^k$ and $y_{true,t}^k$ are the x- and y-position of the simulated k^{th} object which also is the ground truth.

The AGV made two turns around its center point, one 90° turn between frame 120 and frame 140. The second turn, 180°, was performed between frame 200 and frame 230. The result showed that the algorithm could position the AGV with a mean average error of 0.077 meters and a median error of 0.075 meters.

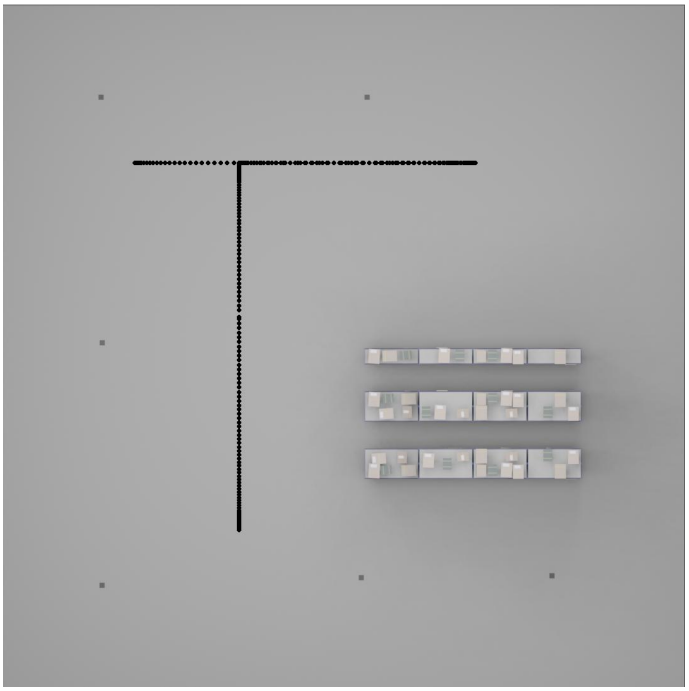


Figure 6.1: Track for single AGV for evaluating positioning of the tracker

The position was calculated and evaluated against the ground truth to see how accurate the camera based tracking predict the position of the tracked objects. A comparison between the true position and the tracked estimated position can be seen in Figure 6.2 and Appendix A.2.

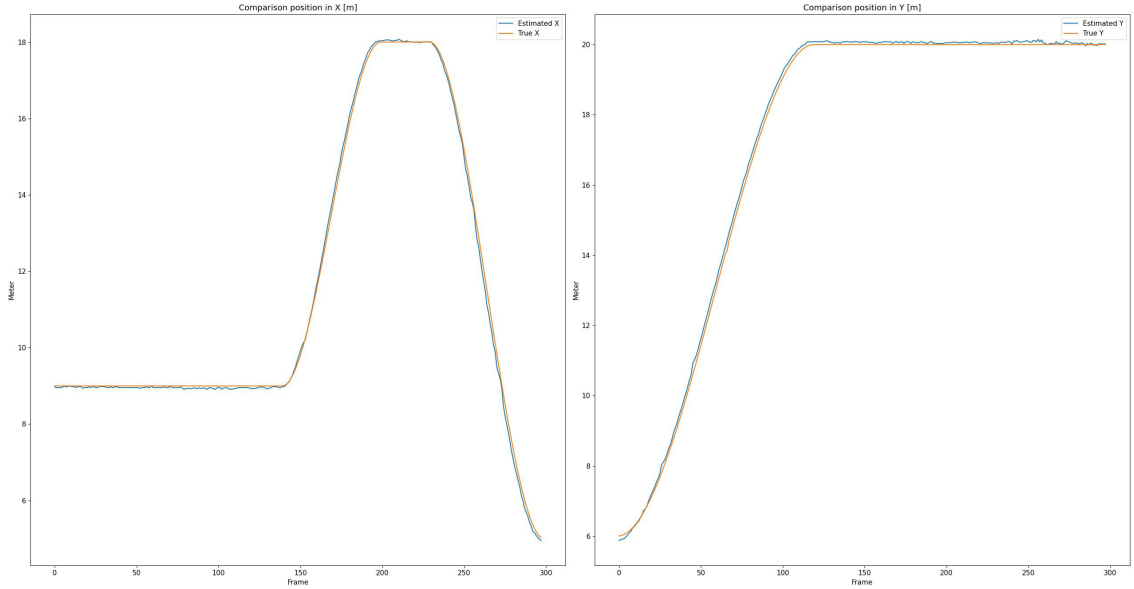


Figure 6.2: Comparison between true state and tracked estimate of single AGV

To evaluate the overall performance of the algorithm with respect to the ground truth the positioning error at every time step was plotted in Figure 6.3 and Appendix A.3.

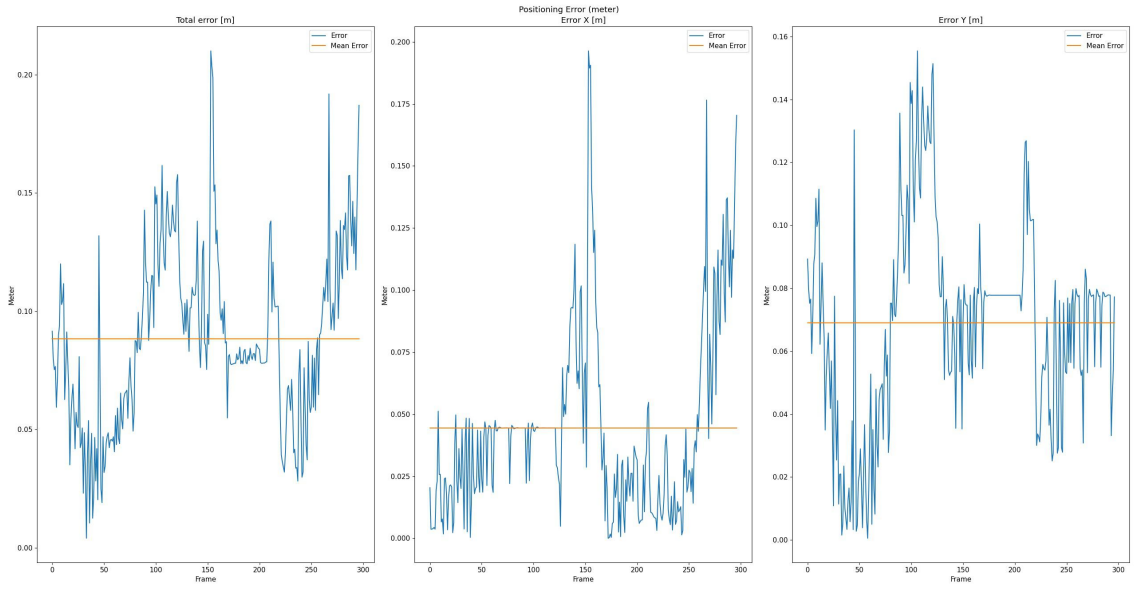


Figure 6.3: Positioning error between true state and tracked estimate of single AGV

6.1.2 Tracking evaluation

Evaluation of the tracker refers to its ability to correctly detect relevant objects, assign identification to these and keep the identification throughout multiple frames. An example of a time instance of the tracker is visualized in Figure 6.4. This part was evaluated on the large warehouse with two humans and four AGVs as can be seen in Figure 6.4. The object detection algorithm was evaluated on mean average precision, mAP, the description of this precision measurement is based on Ren Jie Tans article [53]. To identify true positive, TP, true negative, TN, and false positive, FP, detections the ratio between the intersection of a detected bounding box and the ground truth bounding box is calculated and divided with the union of these two bounding boxes. This ratio is the IoU:

$$\text{IoU} = \frac{\text{Intersection between detection and ground truth}}{\text{Union between detection and ground truth}} \tag{6.3}$$

TP, TN, and FP are now defined as:

$$\begin{bmatrix} \text{TP} \\ \text{TN} \\ \text{FN} \end{bmatrix} = \begin{bmatrix} \text{IoU} > 0.5 \\ \text{IoU} < 0.5 \\ \text{No detection} \end{bmatrix} \tag{6.4}$$

Using this, the precision and loss were calculated and a curve between these was plotted in Figure 6.5 and Figure 6.6. Interpolating at every recall level by taking the maximum precision gives the interpolated precision, integrating this yields the average precision, AP. Taking the mean of the AP for all classes yields the mAP. The object detection network was trained for 400 epochs and yielded a mAP of 74.89% as can be seen in Figure 6.5.

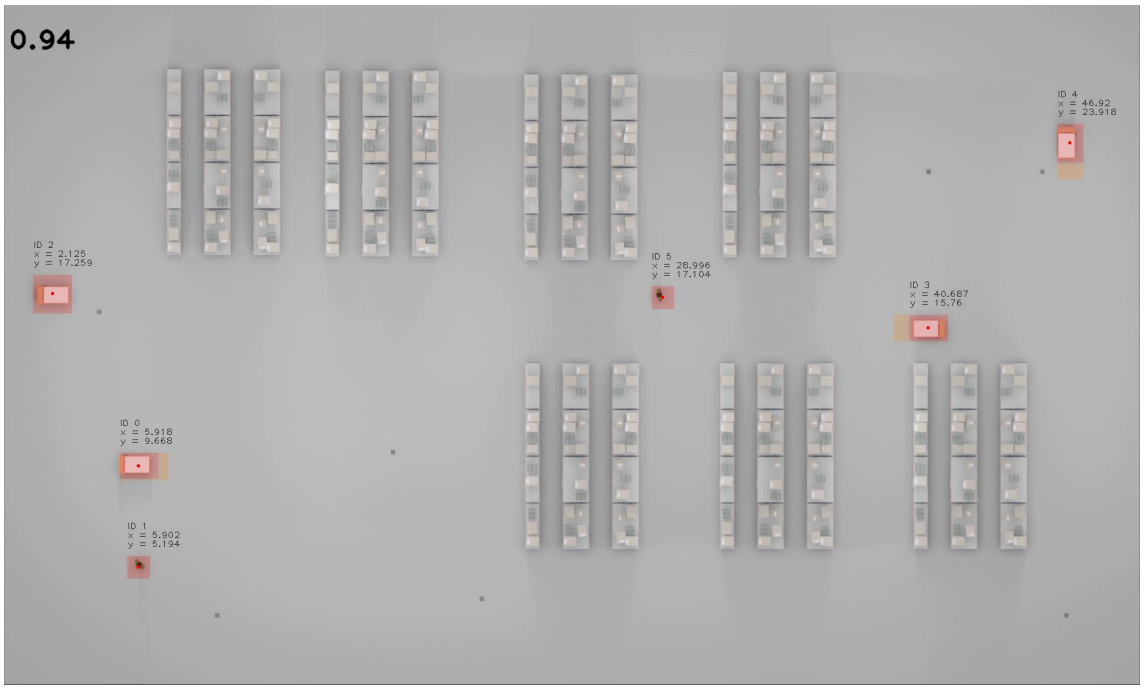


Figure 6.4: Example output from algorithm with ID assigned to AGVs and position information

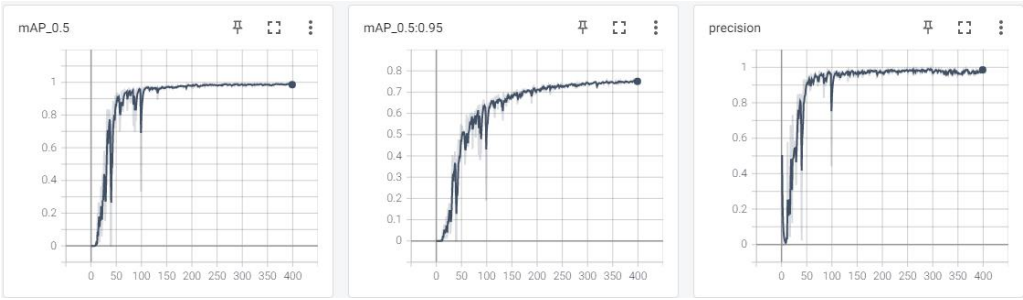


Figure 6.5: Validation mean average precision of the trained model

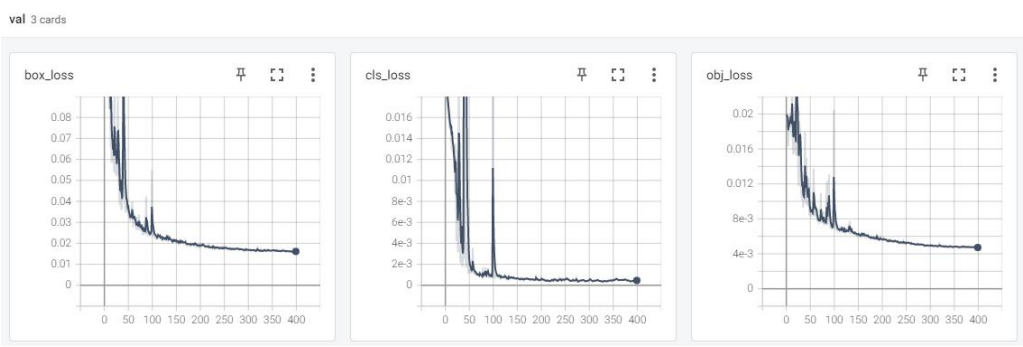


Figure 6.6: Validation loss of the trained model

Furthermore, a general performance measure on multi object tracking systems is defined in the Multi Object Tracking Challenge, MOT, and specifically the Multiple

Object Tracking Accuracy, MOTA value. Evaluation of the tracker was performed using inspiration from the MOT challenge [54] and the work of Bernardin *et al.* [55]. This part of the thesis was evaluated on the precision of the MOTA. MOTA evaluates how many mistakes the tracker makes and indicates how accurate the tracked ID is with respect to the false negatives, FN, FP and identity switches, IDSW, see Eq 6.5. With the simulation of data, ground truth, GT, the number of ground truth objects, could be computed in both exact position and exact velocity. From the ground truth, the error between the estimated position, velocity and heading was computed.

The MOTA value for the system was computed as:

$$\begin{aligned} MOTA &= 1 - \frac{\sum_t (FN_t + PN_t + IDSW_t)}{\sum_t GT_t} = \\ &= 1 - \frac{12 + 1 + 2}{287 \cdot 6 + 13 \cdot 5} = 1 - \frac{14}{1787} = 0.9916 \end{aligned} \tag{6.5}$$

MOTA has been calculated for a dataset consisting of six objects and 300 frames where six of the objects are visible for the cameras in the first 287 frames and then one AGV leaves the remaining five objects are visible in the last 13 frames.

6.1.2.1 Kalman Filter tuning

State noise covariance, Q was set as:

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \Delta t \sigma_v^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Delta t \sigma_\omega^2 \end{bmatrix}}_{\text{State noise covariance}} \tag{6.6}$$

Experiments found different characteristics on the results depending on the filter tuning.

High σ_v:
When σ_v was increased the filtered position became noisy. The speed changed quickly.

Low σ_v:
When σ_v was decreased the changes in velocity was slow.

High σ_ω:
σ_ω affect the yaw rate of the vehicle. A higher σ_ω allowed the yaw rate to be changed more easily and the vehicle would initiate turns on small measurement changes.

Low σ_ω:
When decreasing σ_ω the vehicle had a slow adaption to turns.

Since the velocity of the objects can be assumed to be almost constant, σ_v was set to a small number. Since it is unlikely that the tracked objects turn on straight roads, the turn rate σ_ω was also set to a low value. After experimental testing, the parameters were found according to Table 6.1.

σ_v	0.1
σ_ω	$\frac{0.915}{180}\pi$

Table 6.1: Tuned process noise covariance parameters

Measurement noise covariance was tuned to:

$$R = diag(1\mathrm{e}^{-7}, 1\mathrm{e}^{-7}) \tag{6.7}$$

The SNR is then:

$$\mathrm{SNR} = \frac{||Q||}{||R||} = \frac{||diag(0, 0, \Delta t\sigma_v^2, 0, \Delta t\sigma_\omega^2)||}{||diag(1\mathrm{e}^{-7}, 1\mathrm{e}^{-7})||} \approx 2947 \tag{6.8}$$

The parameters of the sigma point calculations were set as:

κ	=	0
α	=	0.001
β	=	2

Table 6.2: Chosen parameters in Unscented Transform

The velocity was calculated and evaluated against the ground truth to evaluate how accurate the camera based tracking predict the velocity of the tracked objects. A comparison between the true and tracked estimate velocity can be seen in Figure 6.7. The average velocity error was 0,45 m/s.

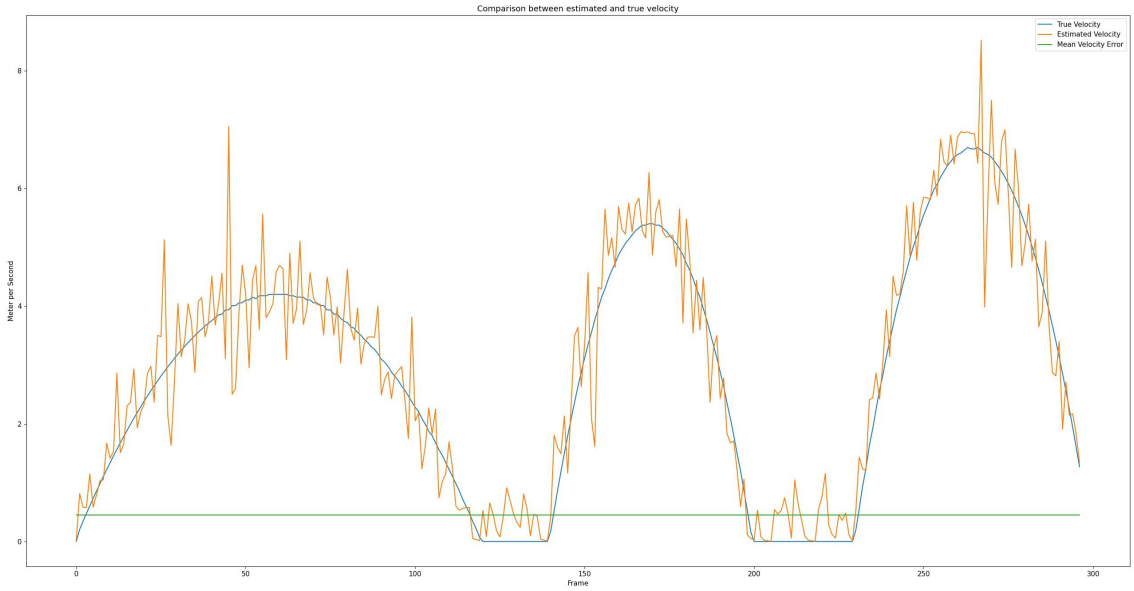


Figure 6.7: Comparison between true velocity and tracked estimate velocity of single AGV

7
Conclusion

The developed system is proven to be able to extract position and velocity parameters of the tracked object with a mean average error of 0,077 meters in position and mean average error in velocity of 0,45 m/s. The different AGV positioning and localization methods could, as mentioned in Section 2.1 be divided into fixed or free trajectory creation, and absolute or relative position measurement. The developed multi camera system could be described as a free and absolute method since the AGVs do not have any path restrictions as long as the objects are visible in at least one of the cameras and can be updated at every time step.

Since the system is tested in a simulated environment, other tracking systems were not evaluated in the same testing environment as the developed systems. Due to this, a fair comparison between systems is not provided. Further studies in other environments could provide this measure. From a cost perspective, a solution with only cameras could be a cheaper option than other advanced sensors, such as LiDAR. The main difference between the developed system and the researched existing systems is that the developed system does not need any onboard sensors on the vehicles for tracking and positioning. This creates flexibility in that the new vehicles and other dynamic objects easily can be introduced without having to equip them with sensors. The AGV will not carry any sensitive sensor equipment, which could result in less consumption of sensors since they do not risk getting damaged to the same extent as if they were located on the driving vehicle. This, as stated in the introduction, relates to Goal 12 concerning consumption in United Nations Agenda for Sustainable Development [3]. Neither are any landmarks in the warehouse required to position the detected objects since the cameras determine the position based on homographies. The neural network can be trained on more classes than only AGVs and humans which would give the AGVs the possibility to avoid other objects, e.g. chairs, tables, boxes or pallets. Static objects like walls and shelves has to be preprogrammed to make the system aware of the objects. Because of this, the warehouse environment can't be changed without reprogramming the static environment. One solution to make it possible to change the static objects would be to train the network to detect static objects. In an environment with multiple static objects or difficult camera angles, this method would probably result in a bad estimation since the bounding boxes could be hard to position satisfactorily.

The system makes it possible to visualize the entire warehouse and get an overview of all tracked objects at once. This could, in future development, make it possible to control all AGVs from one central unit that has the exact position of all objects in the entire warehouse. Instead of the current AGV methods where every AGV controls itself depending on the small environment they see around them.

Having a complete bird-eye view of a warehouse, like the one the developed algorithm provides, would enable collision avoidance on another level than what is currently possible with traditional onboard sensors. Collision avoidance could be applied for cases where other objects are occluded, such as behind corners. Efficiency could be increased as well since an AGV could be instructed to slow down when approaching a corner to allow for another AGV to pass in front of it, rather than arriving at the corner and both AGVs performing a hard stop. By having the equipment placed in the roof instead of on the vehicles creates possibilities to adjust and rebuild the AGVs without having to take into account how to place the sensors on the AGVs, resulting in more optimal shapes of the vehicles. Humans can also enter and leave the warehouse without picking up any sensors or other equipment to be visible for the sensors since the object detector captures humans entering the scene and keeps track of the class and movement until it leaves the warehouse.

This makes it easy to integrate both AGVs and humans and potentially also fix robots in a single warehouse since the system can determine the class of the detected objects and change the size of the safety zone depending on which objects the AGVs are close to. To introduce objects with new appearances the only change that has to be made is to re-train the neural network to make the detection algorithm recognize the new classes. Further development could focus on using the system to control the AGVs with multiple different objects to be avoided.

Bibliography

[1] Harikrishna Rai, Kishore Jonna, and Pisipati Radha Krishna. Video analytics solution for tracking customer locations in retail shopping malls. pages 773–776, 08 2011.

[2] Janez Pers and Stanislav Kovačič. Computer vision system for tracking players in sports games. pages 177 – 182, 02 2000.

[3] UN General Assembly. Transforming our world : the 2030 agenda for sustainable development. Oct. 2015.

[4] J. Wei, J. He, Y. Zhou, K. Chen, Z. Tang, and Z. Xiong. Enhanced object detection with deep convolutional neural networks for advanced driving assistance. *IEEE Transactions on Intelligent Transportation Systems*, 21(4):1572–1583, 2020. doi: 10.1109/TITS.2019.2910643.

[5] A. Raghunandan, Mohana, P. Raghav, and H. V. R. Aradhya. Object detection algorithms for video surveillance applications. Apr. 2018. doi: 10.1109/ICCSP.2018.8524461.

[6] B. Tian, Q. Yao, Y. Gu, K. Wang, and Y. Li. Video processing techniques for traffic flow monitoring: A survey. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1103–1108, 2011. doi: 10.1109/ITSC.2011.6083125.

[7] Y. Chen, Y. Wu, and H. Xing. A complete solution for agv slam integrated with navigation in modern warehouse environment. In *2017 Chinese Automation Congress (CAC)*, pages 6418–6423, 2017. doi:10.1109/CAC.2017.8243934.

[8] JL. Martinez. Agvs: navigation systems. https://medium.com/@jlmartinez.es/agvs-navigation-systems-354b3469e6c3, Apr. 2020. [Online; accessed April-2021].

[9] P. Corke. *Robotics, Vision and Control*. Springer, Berlin, 2013.

[10] AGV network. Automated guided vehicles are equipped with magnetic sensors and follow a defined track made by a magnetic tape track. https://www.agvnetwork.com/index.php/17-agv-components/yuanben/magnetic-guidance-sensors/27-digital-agv-magnetic-guiding-sensor-16-bits#magnetic. [Online; accessed April-2021].

[11] AGV network. What are the laser guided vehicles? lgv advantages and disadvantages. https://www.agvnetwork.com/what-is-a-laser-guided-vehicle-lgv. [Online; accessed April-2021].

[12] AGV network. Natural navigation automated guided vehicles. https://www.agvnetwork.com/natural-navigation-automated-guided-vehicles. [Online; accessed April-2021].

[13] National Oceanic and US Department of Commerce Atmospheric Administration. What is sonar? https://oceanservice.noaa.gov/facts/sonar.html. [Online; accessed January-2021].

[14] A. K. Ray, M. Gupta, L. Behera, and M. Jamshidi. Sonar based autonomous automatic guided vehicle (agv) navigation. In *2008 IEEE International Conference on System of Systems Engineering*, pages 1–6, 2008. doi: 10.1109/SYSOSE.2008.4724179.

[15] J. Young and M. Simic. Lidar and monocular based overhanging obstacle detection. *Procedia Computer Science*, 60:1423 – 1432, 2015. Knowledge-Based and Intelligent Information and Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings. doi:10.1016/j.procs.2015.08.218.

[16] Z. Rozsa and T. Sziranyi. Obstacle prediction for automated guided vehicles based on point clouds measured by a tilted lidar sensor. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2708–2720, Aug. 2018. doi:10.1109/TITS.2018.2790264.

[17] M. Kok, J. D. Hol, and T. B. Schön. Using inertial sensors for position and orientation estimation. In *Foundations and Trends in Signal Processing*, 2017. doi:10.1561/2000000094.

[18] L. Li, Y. Liu, M. Fang, Z. Zheng, and H. Tang. Vision-based intelligent forklift automatic guided vehicle (agv). In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 264–265, 2015. doi:10.1109/CoASE.2015.7294072.

[19] Y. Wu, L. Wang, J. Zhang, and F. Li. Path following control of autonomous ground vehicle based on nonsingular terminal sliding mode and active disturbance rejection control. *IEEE Transactions on Vehicular Technology*, 68(7):6379–6390, 2019. doi: 10.1109/TVT.2019.2916982.

[20] C. Hu, H. Jing, R. Wang, F. Yan, and M. Chadli. Robust output-feedback control for path following of autonomous ground vehicles. *Mechanical Systems and Signal Processing*, 70-71:414 – 427, 2016.

[21] M. V. Gomes, L. A. Bássora, O. Morandin, and K. C. T. Vivaldini. Pid control applied on a line-follower agv using a rgb camera. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 194–198, 2016. doi:10.1109/ITSC.2016.7795553.

[22] P. Soviany and R. T. Ionescu. Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction. *arXiv*, Sep. 2018. doi:10.1109/SYNASC.2018.00041.

[23] Y. Bengio I. Goodfellow and A. Courville. Deep learning. chapter 6 and 9. MIT Press, 2016. http://www.deeplearningbook.org.

[24] M. Nielsen. Neural networks and deep learning. chapter 2. 2019. http://neuralnetworksanddeeplearning.com/.

[25] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

[26] D. Thuan. Evolution of yolo algorithm and yolov5: The state-of-the-art object detection algorithm. Bachelor's thesis, Information Technology Oulu University of Applied Sciences, 2021. http://urn.fi/URN:NBN:fi:amk-202103042892.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv*, 2016. https://arxiv.org/pdf/1506.02640.pdf.

[28] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv*, 2016. https://arxiv.org/pdf/1612.08242.pdf.

[29] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018. https://arxiv.org/pdf/1804.02767.pdf.

[30] H.Y. M. Liao A. Bochkovskiy, C.Y. Wang. Yolov4: Optimal speed and accuracy of object detection. *arXiv*, 2020. https://arxiv.org/pdf/2004.10934.pdf.

[31] Rasmus Rothe, Matthieu Guillaumin, and Luc Van Gool. Non-maximum suppression for object detection by passing messages between windows. volume 9003, 04 2015.

[32] G. Jocher. yolov5. https://github.com/ultralytics/yolov5, 2021. [Online; accessed April-2021].

[33] C.Y. Wang, H.Y. M. Liao, I.H. Yeh, Y.H. Wu, P.Y. Chen, and J.W. Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn. 11 2019. https://arxiv.org/pdf/1911.11929.pdf.

[34] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. 9 2018. https://arxiv.org/pdf/1803.01534.pdf.

[35] Seyed Hamid Rezatofighi, Anton Milan, Zhen Zhang, Qinfeng Shi, Anthony Dick, and Ian Reid. Joint probabilistic data association revisited. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3047–3055, 2015.

[36] Pavlina Konstantinova, Alexander Udvarev, and Tzvetan Semerdjiev. A study of a target tracking algorithm using global nearest neighbor approach 1. 06 2003.

[37] C. L. Phillips and J. M. Parr. *Signals, Systems, and Transforms*, chapter 3. Pearson Education, Inc., USA, 2008.

[38] S. Särkkä. *Bayesian Filtering and Smoothing*, volume 3. Cambridge University Press, 2013.

[39] S. J. Julier and J. K. Uhlmann. A new extension of the kalman filter to nonlinear systems. https://www.cs.unc.edu/~welch/kalman/media/pdf/Julier1997_SPIE_KF.pdf.

[40] K. Okuma, J. Little, and D. Lowe. Automatic rectification of long image sequences. Jan. 2003.

[41] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004.

[42] C. V. Jawahar A. Agarl and P. J. Narayanan. A survey of planar homography estimation techniques. 2005.

[43] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[44] Pujiyanto. Storage rack, blender 2.8x, May. 2020. Blender 2.8.

[45] GeorgeTheMaker. Feng, Dec. 2020. Blender 2.9.

[46] Tzutalin. Labelimg. https://github.com/tzutalin/labelImg, 2015.

[47] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
[48] Intel Corporation. *Intel®Hyper-Threading Technology, Technical User's Guide*, January 2003.
[49] P. Wu. Multi-class yolov5 + deep sort with pytorch. `https://github.com/WuPedin/Multi-class_Yolov5_DeepSort_Pytorch`, 2020.
[50] J. Solawetz and J. Nelson. How to train yolov5 on a custom dataset. `https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/`, May. 2020. [Online; accessed May-2021].
[51] R. Labbe. Filterpy. `https://github.com/rlabbe/filterpy`, 2015. [Online; accessed April-2021].
[52] SICK AG. Safe motion: Non-stop safe processes. `https://www.sick.com/de/en/safe-productivity-safe-motion/w/safe-motion/`. [Online; accessed April-2021].
[53] R. J. Tan. Breaking down mean average precision (map). `https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#f9ce`, Mar. 2019. [Online; accessed April-2021].
[54] P. Dendorfer, A. Ošep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth, and L. Leal-Taixé. Motchallenge: A benchmark for single-camera multiple target tracking, 2020.
[55] K. Bernardin, A. Elbs, and R. Stiefelhagen. Multiple object tracking performance metrics and evaluation in a smart room environment.

A

Appendix 1

A.1 Algorithm Find Intersections

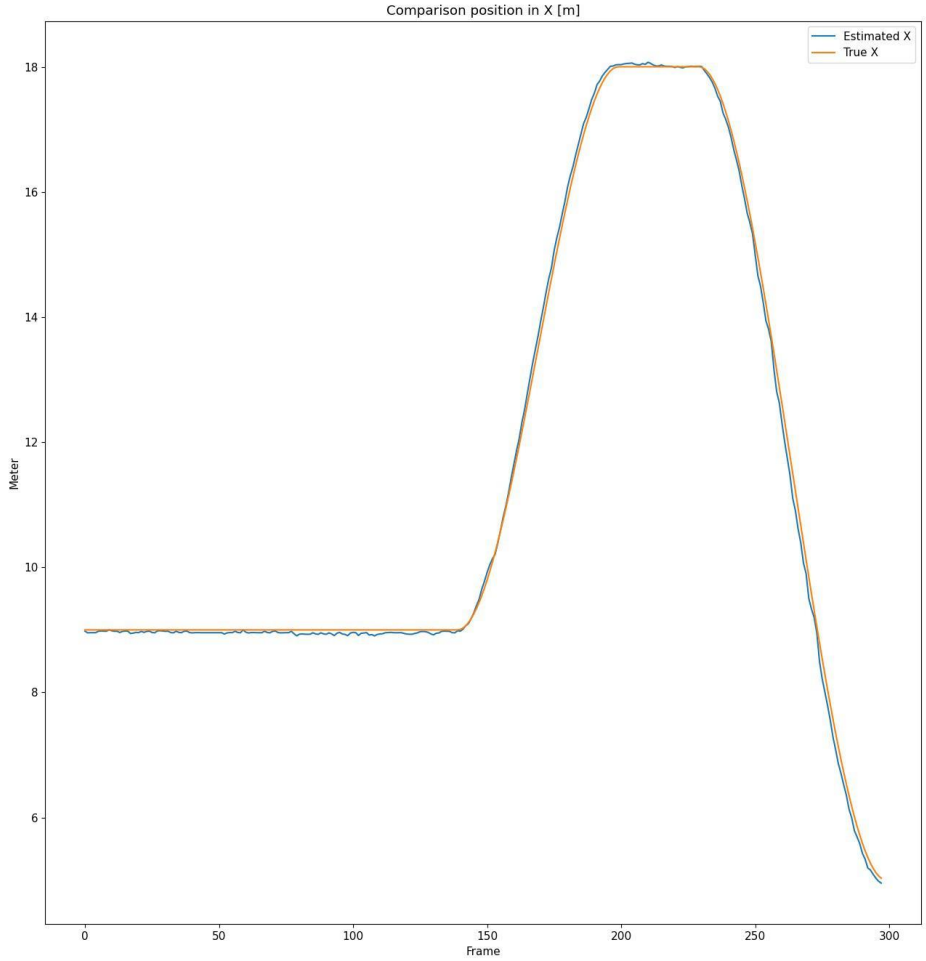
Algorithm 3 Find intersections

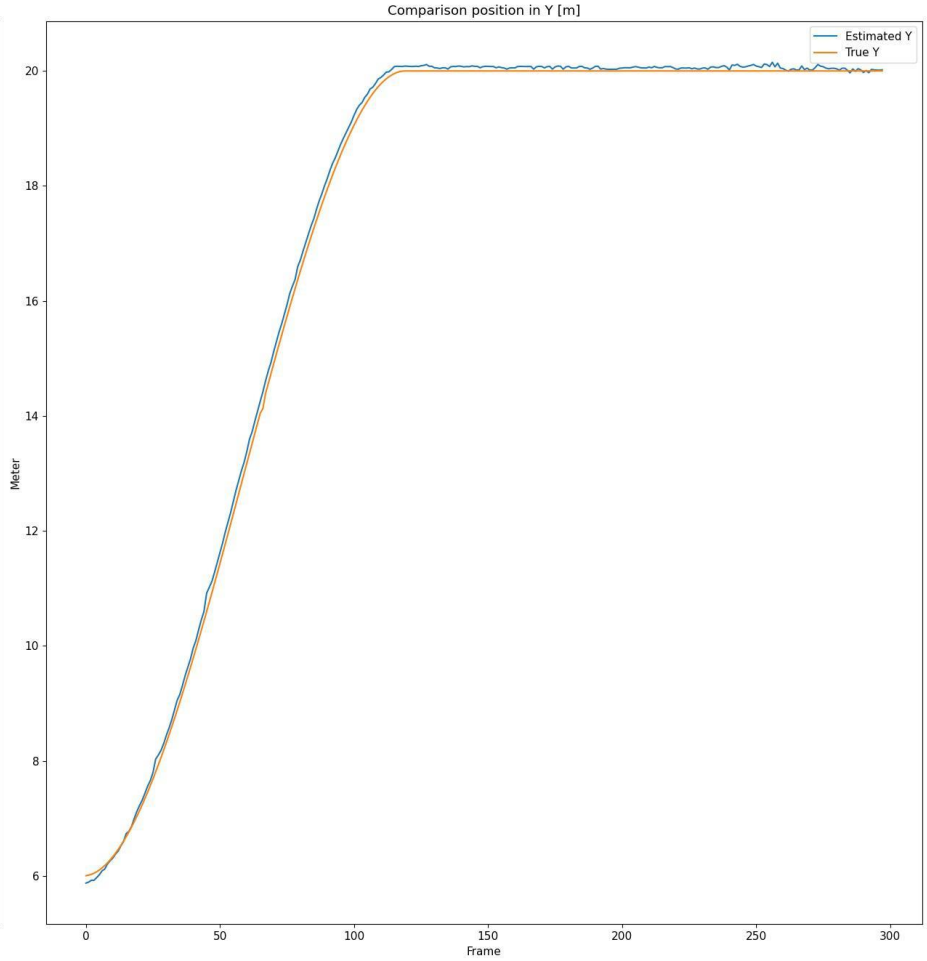
```
for index = (1, nBBoxes) do
  if index ∈ available bboxes then
    add index to intersected bbox
    remove index from available bboxes
  switcher = row
  end = False
  while Not end and available bboxes is not empty do
    current index = index
    if switcher = row then
      loop_flag = true
      if max of IoUMatrix[index,:] = 0, no overlapping boxes then
        no overlapping boxes
        loop_flag = false
        end = True
      end if

      i = 1
      while loop_flag, intersecting boxes exist do
        iou_value = max(sorted(IoUMatrix[current index,:]))
        index = index of iou_value

        if iou_value = 0 then
          No overlapping boxes
          loop_flag = false
          end = True
        end if

        if length of index > 1 then
          if iou_value > 0 then
            for i ∈ range 1 to length of index do
              IoUMatrix[current index, index[i]] -= 0.01*i
            end for
            index = index of iou_value
          else
            loop_flag = false
            end = True
            Break
```

```
38:
39:                 if index in available bboxes then
40:                   if all IoUMatrix[i,index] > 0 then
41:                     for i ∈ intersected bbox do
42:                       add  index to intersected bbox
43:                       remove  index from avalible bboxes
44:                       loop_flag = false
45:                     end for
46:                   else
47:                     i += 1
48:                   end if
49:                 else
50:                   i += 1
51:                 end if
52:               end if
53:             end if
54:           end while
55:           switcher = col
56:
57:         else if switcher = col then
58:           current index = index
59:           i = 1
60:           loop_flag = true
61:           while loop_flag, intersecting boxes exist do
62:             iou_value = max(sorted(IoUMatrix[current index,:]))
63:             index = index of iou_value
64:
65:             if iou_value = 0 then
66:               No overlapping boxes
67:               loop_flag = false
68:               end = True
69:             else if iou_value = 0 and i = 1 then
70:               loop_flag = false
71:               end = True
72:             else
```

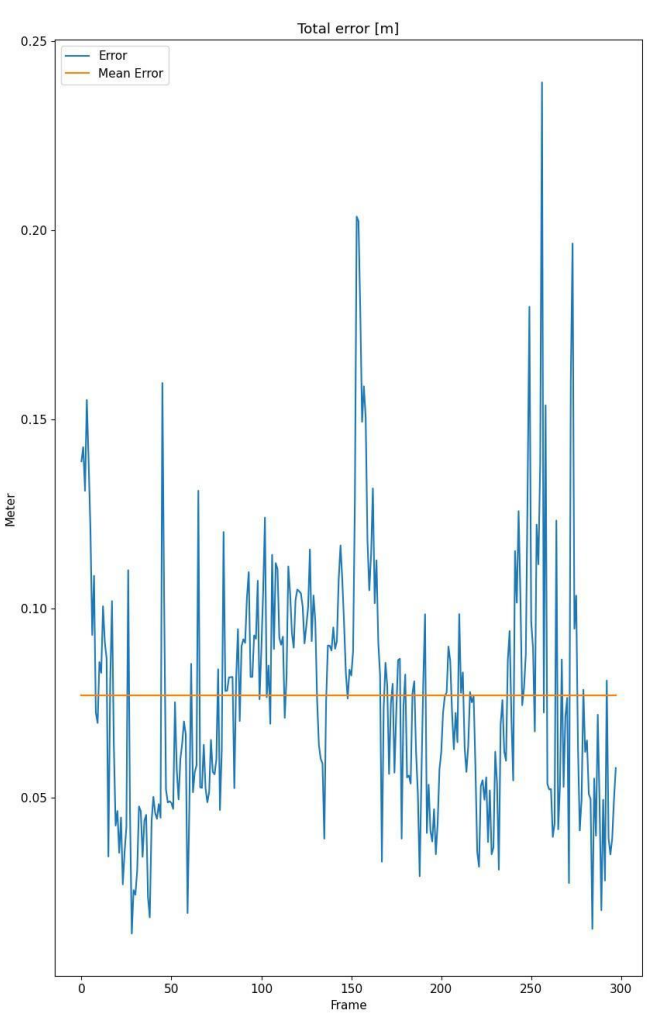
```
                index = index of iou_value
                if length of iou_value > 1 then
                  if iou_value > 0 then
                    for i ∈ range 1 to length of index do
                      IoUMatrix[current index, index[i]] -= 0.01*i
                    end for
                    index = index of iou_value
                  else
                    loop_flag = false
                  end if
                else
                  if index ∈ avalible bboxes then
                    if all IoUMatrix[i,index] > 0 then
                      for i ∈ intersected bbox do
                        add  index to intersected bbox
                        remove  index from available bboxes
                        loop_flag = false
                        switcher = row
                      end for
                    else
                      i += 1
                    end if
                  else
                    i += 1
                  end if
                end if
              end if
            end while
          end if
        end while
      end if
    end for
```

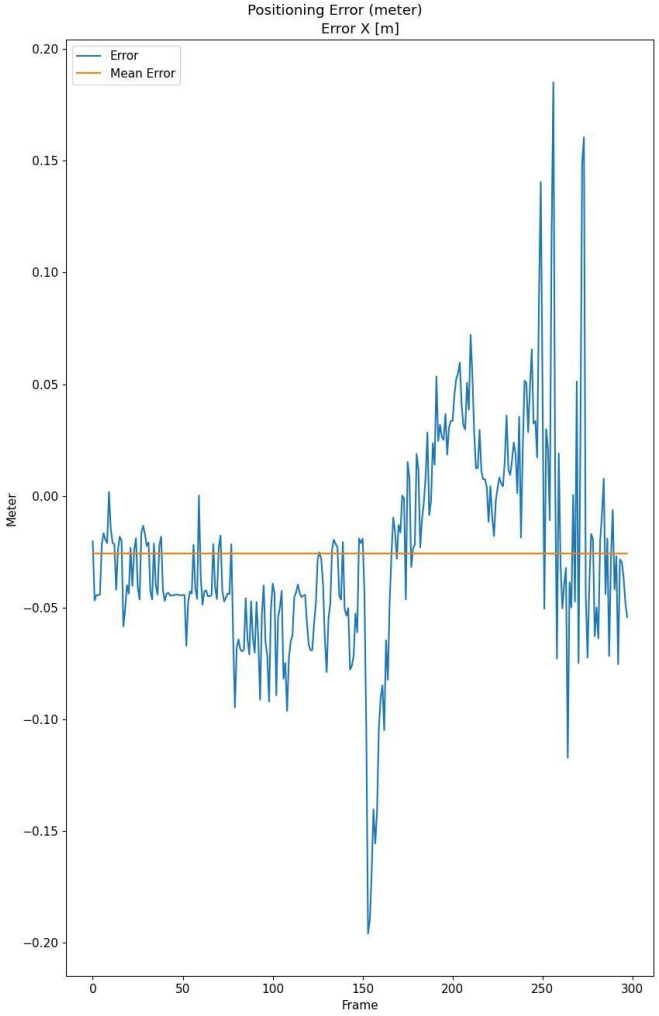
A.2 Position Comparison

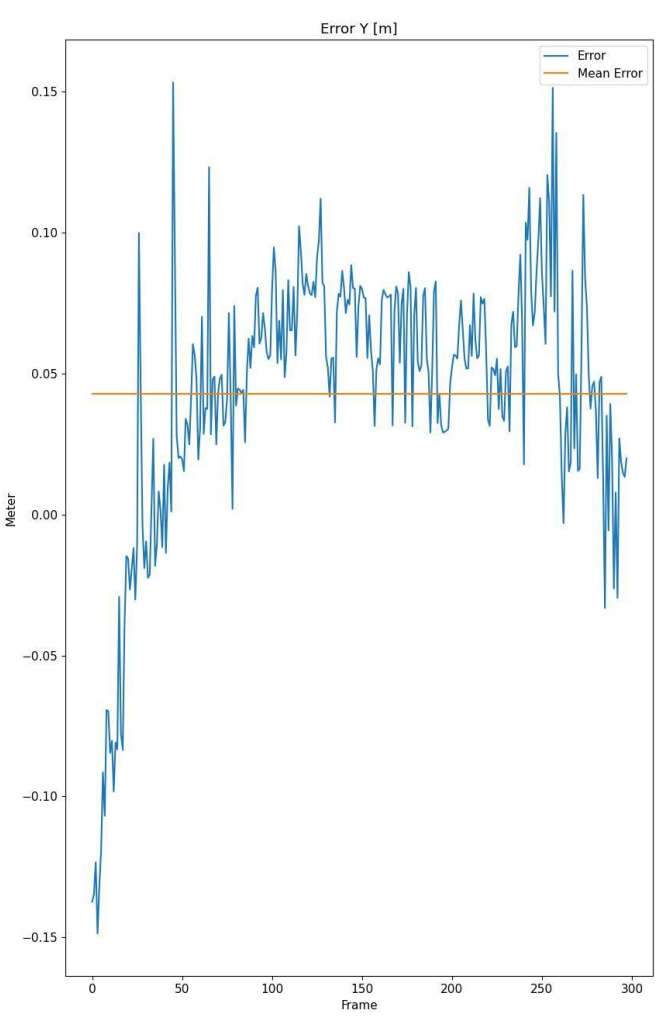

Comparison position in X [m]
Estimated X
True X
Meter
Frame
18
16
14
12
10
8
6
0
50
100
150
200
250
300


Comparison position in Y [m]
Estimated Y
True Y
Meter
Frame
20
18
16
14
12
10
8
6
0
50
100
150
200
250
300

A.3 Position Error


Total error [m]
Error
Mean Error
Meter
Frame
0.25
0.20
0.15
0.10
0.05
0
50
100
150
200
250
300


Positioning Error (meter)
Error X [m]
Error
Mean Error
Meter
Frame
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
50
100
150
200
250
300


Error Y [m]
Error
Mean Error
Meter
Frame
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
50
100
150
200
250
300

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY