Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2022

# Collision Avoidance for Complex and Dynamic Obstacles

## A study for warehouse safety

**Ester Brandås**
**Sandra Ljungberg**

Master of Science Thesis in Electrical Engineering

**Collision Avoidance for Complex and Dynamic Obstacles: A study for warehouse safety**

Ester Brandås
Sandra Ljungberg

LiTH-ISY-EX--22/5516--SE

Supervisor: **Carl Hynén Ulfsjöö**
ISY, Linköping University
**Håkan Therén**
Toyota Material Handling Manufacturing Sweden
**Carl Westman**
Toyota Material Handling Manufacturing Sweden

Examiner: **Daniel Axehill**
ISY, Linköping University

*Division of Automatic Control*
*Department of Electrical Engineering*
*Linköping University*
*SE-581 83 Linköping, Sweden*

Copyright © 2022 Ester Brandås
Sandra Ljungberg

# Acknowledgments

We would like to thank our supervisors for their invaluable input during this thesis work. Carl Hynén Ulfsjöö at Linköping University for always having new interesting feedback and ideas, and Håkan Therén and Carl Westman at TMHMS, for all of their support and guidance. We would also like to thank our examiner Daniel Axehill for his shown interest in the work. In addition, we want to give a special thank you directed to our fellow thesis writers at TMHMS, Anton Blåberg, Erik Sellén, Gustav Lindahl, and Robert Sehlstedt, for making this period so enjoyable. Lastly, we would like to thank family and friends for all of their support.

*Linköping, June 2022*
*Ester Brandås and Sandra Ljungberg*

# Abstract

Today a group of automated guided vehicles at Toyota Material Handling Manufacturing Sweden detect and avoid objects primarily by using 2D-LiDAR, with shortcomings being the limitation of only scanning the area in a 2D plane and missing objects close to the ground. Several dynamic obstacles exist in the environment of the vehicles. Protruding forks are one such obstacle, impossible to detect and avoid with the current choice of sensor and its placement. This thesis investigates possible solutions and limitations of using a single RGB camera for obstacle detection, tracking, and avoidance.

The obstacle detection uses the deep learning model YOLOv5s. A solution for semi-automatic data gathering and labeling is designed, and pre-trained weights are chosen to minimize the amount of labeled data needed.

Two different approaches are implemented for the tracking of the object. The YOLOv5s detection is the foundation of the first, where 2D-bounding boxes are used as measurements in an Extended Kalman Filter (EKF). Fiducial markers build up the second approach, used as measurements in another EKF.

A state lattice motion planner is designed to find a feasible path around the detected obstacle. The chosen graph search algorithm is ARA*, designed to initially find a suboptimal path and improve it if time allows.

The detection works successfully with an average precision of 0.714. The filter using 2D-bounding boxes can not differentiate between a clockwise and counter-clockwise rotation, but the performance is improved when a measurement of rotation is included. Using ARA* in the motion planner, the solution sufficiently avoids the obstacles.

# Contents

**Acknowledgments** **iii**

**Abstract** **v**

**1 Introduction** **1**
1.1 Background and Purpose . . . . . 1
1.2 Delimitations and Prerequisites . . . . . 2
1.2.1 Automated Guided Vehicle . . . . . 2
1.2.2 Obstacles and Environment . . . . . 3
1.3 Related Work . . . . . 3
1.4 Problem Statement . . . . . 4

**2 Theory** **5**
2.1 Computer Vision . . . . . 5
2.1.1 Pinhole Model and Camera Calibration . . . . . 5
2.2 Vision Based Object Detection . . . . . 6
2.2.1 YOLO . . . . . 6
2.2.2 Model Evaluation . . . . . 7
2.3 Object Tracking Using a Monocular Camera . . . . . 8
2.3.1 Kalman Filtering . . . . . 8
2.4 State Lattice Motion Planning Algorithm . . . . . 9
2.4.1 Design of the State Space Lattice . . . . . 10
2.4.2 Graph Search Algorithm . . . . . 11

**3 Method** **13**
3.1 System Architecture . . . . . 13
3.1.1 Flowchart . . . . . 13
3.1.2 Hardware Setup . . . . . 13
3.2 Vision Based Object Detection . . . . . 14
3.2.1 Fiducial Markers . . . . . 14
3.2.2 Data Acquisition . . . . . 15
3.2.3 Training a Network for Object Detection . . . . . 18
3.3 Object Tracking Using a Monocular Camera . . . . . 18

3.3.1 Tracking of 3D-object with 2D-bounding Boxes . . . . . 19
3.3.2 Tracking a Fiducial Marker . . . . . 21
3.4 State Lattice Motion Planning . . . . . 21
3.4.1 Building Outer Bound of the Object . . . . . 21
3.4.2 Checking if an Obstacle is Obscuring a Point . . . . . 23
3.4.3 Design of the State Space Lattice . . . . . 23
3.4.4 Graph Search Algorithm . . . . . 25

**4 Result and Discussion** **27**
4.1 Vision Based Object Detection . . . . . 27
4.2 Object Tracking Using a Monocular Camera . . . . . 28
4.2.1 Planning and Executing the Tests . . . . . 28
4.2.2 Tests with Zero Angular Velocity . . . . . 29
4.2.3 Tests with Non-Zero Angular Velocity . . . . . 34
4.2.4 Tests Including an Extra Measurement of Rotation . . . . . 36
4.3 State Lattice Motion Planning . . . . . 37

**5 Conclusion** **43**
5.1 Answering the Problem Statements . . . . . 43
5.2 Future Work . . . . . 44

**Bibliography** **47**

# 1 Introduction

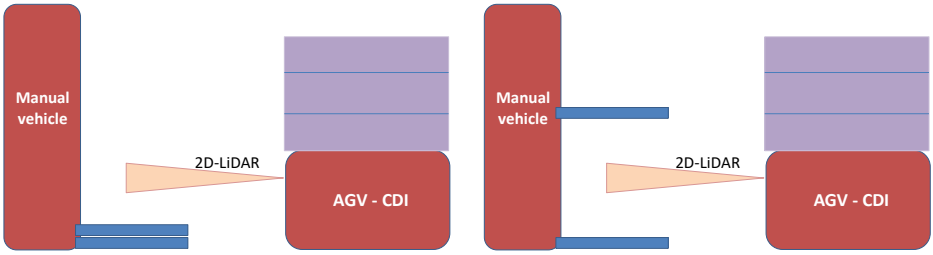
This thesis focuses on improving warehouse safety by implementing collision avoidance for a group of automated guided vehicles (AGV's) used at Toyota Material Handling in Mjölby, Sweden. The following chapter introduces the thesis by discussing the identified problem and purpose.

## 1.1 Background and Purpose

With the development of automated vehicles, the importance of collision avoidance continues to increase. It is a topical issue in today's warehouses, where humans co-exist with both manual forklifts and several types of automated vehicles. A rapidly changing environment gives the incentive to take dynamic aspects into account when discussing collision avoidance and warehouse safety.

Today a group of automated guided vehicles at Toyota Material Handling Manufacturing Sweden (TMHMS) can detect and avoid objects primarily by using 2D-LiDAR and slowing down or coming to a complete stop if an obstacle gets too close. The shortcomings of this sensor, shown in Figure 1.1, is the limitation of only scanning the area in a 2D plane. This entails a risk of missing objects beneath or above the visible plane.

This master thesis investigates possible solutions and limitations of using a single RGB camera instead of 2D-LiDAR for obstacle detection, and adds software for tracking the object and planning a new route around it.




***(a)*** *Scenario with forks close to ground, usually seen during transportation. The scanned 2D-plane misses the forks.*

***(b)*** *Scenario with forks lifted, occurs during loading and unloading of goods. The scanned 2D-plane misses the forks.*

***Figure 1.1:*** *Two currently problematic scenarios (a) and (b) using 2D-LiDAR for collision avoidance.*

## 1.2 Delimitations and Prerequisites

The application is implemented for one of the existing AGV's at TMHMS, in synchronization with the current system.

### 1.2.1 Automated Guided Vehicle

The automated guided vehicle used in this thesis, seen in Figure 1.2, is a differential drive robot with functions for safety fields and emergency stops based on 2D-LiDAR. The system is compatible with further development and more features can be added with new software. An external application handles localization, navigation, and control of the AGV.

***(a)*** *Without load.*

***(b)*** *With load.*

***Figure 1.2:*** *The type of automated guided vehicle (AGV) used in the thesis.*

It is a requirement that the solution can handle floor conditions present at the TMHMS factory. It also needs to communicate with the current software on the

AGV. The available sensor, located at the front of the AGV two decimeters over the floor, is a single RGB camera.

### 1.2.2 Obstacles and Environment

The AGV can map its static environment when introduced to new surroundings. It is done with simultaneous localization and mapping (SLAM) and can be reconfigured each time conditions change.

Several dynamic obstacles exist in tandem with the AGV with the ability to move position over time, as opposed to the fixed environment. The focus of this thesis is forklifts since the protruding forks will make the obstacles more complex and impossible to detect with the current choice and placement of the sensor.

As a limitation, this thesis only includes stacker forklifts with fork sizes 130x60x12 centimeters.

## 1.3 Related Work

Traditional algorithms for obstacle avoidance often use real-time data from a 2D-LiDAR when computing a trajectory. Problems with this choice can occur since some obstacles go unnoticed by the 2D-LiDAR [1]. One approach to solve this problem is to add a camera to the set of sensors and use vision-based object detection.

Deep learning is a machine learning approach commonly used in vision-based object detection. Liu et al. [1] discuss one such alternative with a combination of 2D-LiDAR and data from an RGB camera. A deep learning algorithm stands for the cognitive part with the visual images as inputs and the probability of obstacles ahead as output. Jia et al. [2] instead suggest a solution for pallet detection using a deep learning algorithm and data from a time-of-flight camera. This method results in coordinates of the center of the pallet, and it has a high recognition rate in complex backgrounds.

An alternative approach, by Chowdhury et al. [3], compare the foreground image with the background image to extract vehicles. Another study, by Nguyen and Yoo [4], uses image processing with a fuzzy logic fusing strategy for data from a camera together with data from a 3D-LiDAR. Using fiducial markers, one can estimate the position of an object. Kallwies et al. [5] examine detection rate, runtime, and localization accuracy of square fiducial marker detection systems. It is also possible to structure the obstacle avoidance differently and detect free space instead of occupied space, a method used by Pazhayampallil [6].

Knowledge of the location of a detected complex object, either in a global world or relative to the camera, is needed to successfully perform obstacle avoidance. Michels et al. [7] suggest an approach where supervised learning estimates relative depths to obstacles in an image. Either camera images labeled with ground-truth distances to the closest obstacles, or synthetic graphic images, are used to train the

learning algorithm. Masoumian et al. [8] propose a method that uses two deep networks, where objects are detected and localized in one and the depth of the image computed in the other. They introduce two models to estimate depth, supervised deep learning and unsupervised deep learning. Supervised learning requires both original depth maps and color images to train the model. Mustamin [9] instead uses a width-based method for the distance estimation. Their study shows that a distance between two vehicles can be estimated as long as the actual vehicle width and the camera's focal length are known parameters.

The state lattice motion planning algorithm is a popular deterministic method. Introduced by Pivtoraiko et al. [10], it is built on a discretized state space. Each state is connected to its reachable neighbors by feasible motions, further referred to as motion primitives. States and edges build up a connected graph and use systematic graph search algorithms to find a feasible path.

Andersson et al. [11] propose an extension to the state lattice motion planning algorithm [10] that takes both static and dynamic obstacles into account. The proposed method is a receding-horizon lattice-based motion planner. Moving obstacles impose constraints on planning time, which gives the method incentive to have a shorter computational time. It includes a two-step approach, where a time limit exists on the first graph search. If no feasible solution exists before the time limit, the dynamic motion planning problem is designed to be fast and solved in a receding horizon fashion. Another necessary extension when trying to avoid collision with moving obstacles is permission for the vehicle to wait in the same state or revisit a previous state.

## 1.4 Problem Statement

With regards to the identified purpose and requirements taken into account, the following three research questions are formulated.

- Is it possible to use a single RGB camera to detect complex obstacles in form of protruding forks?
- Based on data from a single RGB camera, is it possible to track the object's position in the world?
- Can motion planning be designed to avoid complex dynamic obstacles based on RGB-camera data?

# 2
# Theory

This chapter presents theory related to the application. First, some concepts in computer vision are addressed, then theory related to object detection, tracking, and avoidance is described.

## 2.1 Computer Vision

A digital 2D image constructed from an RGB camera is a projection of a real 3D world on a pixelated 2D plane. Translating between a real-world coordinate and a pixel coordinate requires knowledge of the relation between the camera coordinate system and the image, gathered by calibration and coordinate system transfer.

### 2.1.1 Pinhole Model and Camera Calibration

The pinhole camera model, illustrated in Figure 2.1, describes a real-world object viewed through a pinhole. Rays of light passing through the aperture cause projection of an inverted image on the 2D image plane at the back of the camera [12].

Intrinsic parameters depend on the camera and include focal point $(c_x, c_y)$ and focal length $(f_x, f_y)$, defined as the distance between the focal point and the 2D image plane. These parameters create the camera matrix, which maps the camera coordinates to the image plane [12].

As the pinhole camera model does not have a lens, applying it to a real-life camera requires the inclusion of radial and tangential lens distortion. Camera calibration estimates the intrinsic parameters and distortion coefficients used to rectify the image from distortion [12].

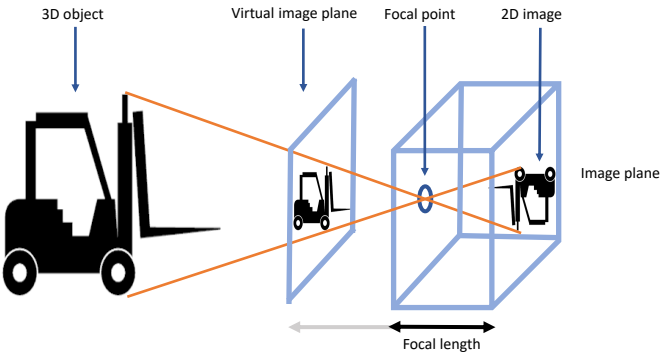



***Figure 2.1:*** *Visualization of the pinhole model.*

## 2.2 Vision Based Object Detection

Vision-based object detection, recognizing specific objects in an image or video stream, can be done using only traditional image processing with filters or by including artificial intelligence approaches. A subset of artificial intelligence is called deep learning, which traditionally learns from scratch for every new task by extracting features from structured and labeled data before using the gained knowledge to classify objects on new data. Rebuilding the solution is required when conditions change. Pre-processing and human knowledge are both necessary when determining which features are relevant to extract [13]. Deep Learning consists of neural networks built with layered nodes to mimic the behavior of a human brain. Deep learning does not need as much data for pre-processing as traditional machine learning, as it can process unstructured data and automates the extraction of features and ranking of their importance [14].

A deep learning model can either be trained from scratch or used with pre-trained weights. The first approach requires more gathered and labeled data than the second. Pre-trained weights must have been trained for a similar problem as the new task to work successfully [14].

### 2.2.1 YOLO

You Only Look Once (YOLO) is a deep neural network created for object localization. It resizes the image, runs a convolutional neural network, and uses the calculated confidence of the model as a detection threshold. The model uses information both of context and appearance during training and learns a generalization of the representation of objects. YOLO is fast and identifies objects quickly with a maximum delay of 25 ms run on a Titan X GPU [15].

As a first step, YOLO divides the input image into an S x S grid. One grid cell is responsible for detecting every object with its center present in it by predicting bounding boxes and calculating scores for how confident the model is. The bounding box itself predicts x- and y-coordinates for its center, width, and height and confidence of the predictions. Every bounding box is also responsible for predicting class probabilities, which in testing are used together with the confidence score to calculate how confident the model is that A: the object in the bounding box is the predicted class, and B: the bounding box fits the object well [15].

Detecting an overfitted YOLO model is done by analyzing the loss curves generated during training. When a model gives a smaller loss value for a large number of epochs for the training data, but the loss value is constant or increases for the validation data, the model has been overfitted [16]. Bounding box regression loss, also called box loss, is the mean squared error between the ground truth and the predicted bounding box with both box size and location taken into account. Objectness, a score of how likely the box has an object inside, creates a second loss function [17].

### 2.2.2 Model Evaluation

Intersection over Union (IoU), the area of the intersection divided by the area of the union, calculates the overlap between the ground truth and the prediction of bounding boxes. A value of 0 means no overlap and 1 a full overlap. IoU is used for setting a threshold $\alpha$ to decide when a model has detected an object correctly or not. A detected object when $IoU \geq \alpha$ means a true positive, and with $IoU \leq \alpha$ a false positive. A false negative occurs with a missed detection [18].

Precision describes the degree of exactness of the model detecting only the wanted objects [18]:

$$Precision = \frac{True_{positive}}{True_{positive} + False_{positive}}. \tag{2.1}$$

The recall is the ratio between the images correctly classified as positive and all images labeled as positive in ground truth:

$$Recall = \frac{True_{positive}}{True_{positive} + False_{negative}}, \tag{2.2}$$

and a high value means that all positive labels are classified correctly [18].

The precision-recall curve, $p(r)$, shows the trade-off between precision and recall, with a theoretically perfect model having both values equal to 1. A network with high precision and low recall accurately classify positive images but seldom classifies images as positive. A network with low precision and high recall classifies positive images well but gives a higher number of false positives [19].

Average precision evaluated at the IoU threshold $\alpha$:

$$AP@\alpha = \int_0^1 p(r)dr, \tag{2.3}$$

is the area under the Precision-Recall Curve [19].

## 2.3 Object Tracking Using a Monocular Camera

One problem studied in signal processing is how to separate a signal $s$ from noise $n$, where a measurement $y_k$ at sample $k$ is modeled as:

$$y_k = s_k + n_k. \tag{2.4}$$

A signal can be reconstructed by either using filtering, prediction or smoothing, where the available measurements, iterated with $l$, up to sample $m$ are:

$$y_l, \quad l \leq m. \tag{2.5}$$

When a signal is estimated directly when the corresponding measurement is available, filtering is used and $m = k$. Prediction uses measurements with $m < k$ hence the signal is predicted forward in time. Smoothing is the last case, where $m > k$ and future measurements are used when estimating the signal [20].

### 2.3.1 Kalman Filtering

Kalman filtering solves filtering, prediction, and smoothing problems while also being suitable for real-time implementations [20]. The original Kalman filter assumes a discrete-time linear model, later extended to include continuous-time systems and nonlinear models. The aim is to minimize the covariance of the estimation error [21]. A discrete-time linear state-space model is described with:

$$x_{k+1} = F_k x_k + G_k^u u_k + G_k^w w_k, \tag{2.6a}$$
$$y_k = H_k x_k + H_k^u v_k + e_k, \tag{2.6b}$$

where its measurement relation and the dynamics of the system are linear functions of the state. $F_k$, $G_k^u$, $G_k^w$, $H_k$, $H_k^u$ are matrices describing the system, $e_k$ is the measurement noise and $w_k$ is the process noise, at sample $k$.

The Kalman filter is initiated with an initial state estimate $x_{0|-1}$ for sample $k = 0$, and initial covariance matrix of the initial state estimate $P_{0|-1}$. The next step is a measurement update phase:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}, \tag{2.7a}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - H_k \hat{x}_{k|k-1} - H_k^u u_k), \tag{2.7b}$$
$$P_{k|k} = (I - K_k H_k) P_{k|k-1}, \tag{2.7c}$$

where $R_k$ is the covariance of the measurement noise at sample $k$. This is followed by a time update phase where $Q_k$ is the covariance of the process noise:

$$\hat{x}_{k+1|k} = F_k \hat{x}_{k|k} + G_k^u u_k, \tag{2.8a}$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + G_k^w Q_k G_k^{uT}. \tag{2.8b}$$

The measurement update and time update are then repeated for sample $k = k + 1$ [21].

The extended Kalman filter (EKF) is an extension of the Kalman filter, using linearization to handle nonlinear models. A nonlinear model can be described as:

$$x_{k+1} = f(x_k, w_k), \tag{2.9a}$$

$$y_k = h(x_k) + e_k, \tag{2.9b}$$

where $f$ and $h$ are nonlinear functions.

The initialization of the state estimate and covariance of the state estimate is done as been previously described, however, the measurement update phase is extended to include a linearization step:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - h(\hat{x}_{k|k-1})), \tag{2.10a}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}, \tag{2.10b}$$

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}, \tag{2.10c}$$

where $H_k = (\nabla_x h(x)|_{x=\hat{x}_{k|k-1}})^T$. The time update phase is also extended:

$$\hat{x}_{k+1|k} = f(\hat{x}_{k|k}, 0), \tag{2.11a}$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + G_k Q_k G_k^T, \tag{2.11b}$$

where $F_k = (\nabla_x f(x, 0)|_{x=\hat{x}_{k|k-1}})^T$ and $G_k = (\nabla_w f(\hat{x}_{k|k}, w)|_{w=0})^T$. The measurement update and time update are then again repeated for sample $k = k + 1$ [21].

## 2.4 State Lattice Motion Planning Algorithm

A popular deterministic method in motion planning is the state lattice motion planning algorithm. This algorithm is introduced by Pivtoraiko et al. [10] with the goal to find a feasible path between two given states on a discretized state space while not colliding with arbitrary obstacles. The discrete states are connected to their reachable neighbors, on the discrete grid, by feasible motions. The states and edges build up a connected graph and create a possibility to use systematic graph search algorithms to find a feasible path. A mathematical description

of the discrete graph search problem is presented in Bergman et al. [22]:

$$
\begin{aligned}
\underset{\{m_p^k\}_0^{N-1},N}{\text{minimize}} \quad & \sum_{k=0}^{N-1} L_p(m_p^k) \\
\text{subject to} \quad & x_0 = x_{init},\ x_N = x_{goal}, \\
& x_{k+1} = f_{m_p}(x_k, m_p^k), \qquad k \in [0, N-1] \\
& m_p^k \in \mathcal{P}(x_k), \qquad k \in [0, N-1] \\
& c(x_k, m_p^k) \in \mathcal{X}_{free}, \qquad k \in [0, N-1],
\end{aligned}
$$

where $x_{init}$ is the state where the vehicle starts and $x_{goal}$ is the state where it aims to go. Both states must be part of the state space discretization $\mathcal{X}_d$. $m_p^k$ is a dynamically feasible trajectory, checked by $\mathcal{P}$, that moves the vehicle between two states on $\mathcal{X}_d$. The resulting state transition to $x_{k+1}$ from $x_k$ is described with $f_{m_p}(x_k, m_p^k)$. $\mathcal{X}_{free}$ ensures that the trajectory is in free-space and $L_p$ is a cost function. The solution to the motion planning problem consists of an ordered sequence of feasible trajectories, motion primitives, that takes the vehicle from $x_{init}$ to $x_{goal}$.

### 2.4.1 Design of the State Space Lattice

One way to build a state lattice is by choosing the state space discretization $\mathcal{X}_d$. The grid resolution is chosen based on the application since a more dense grid will lead to heavier calculations in the graph search. An example showing a discrete state space with a grid resolution of 2 meters is viewed in Figure 2.2. The

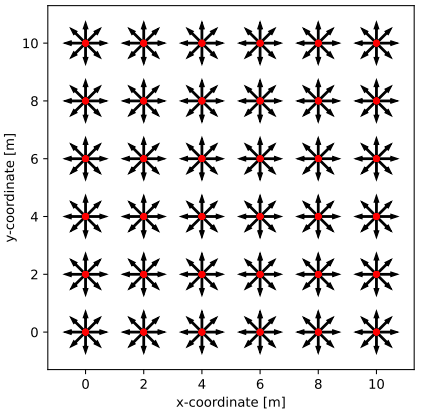
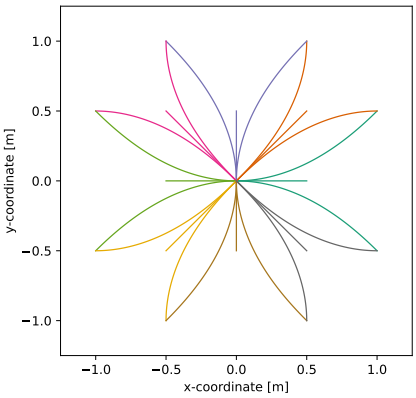
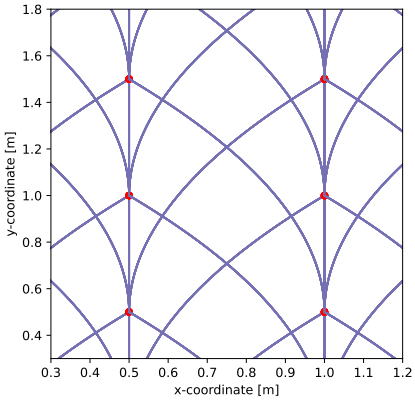



***Figure 2.2:*** *A state lattice with states (x, y, θ) and a grid resolution of two meters. The red dots represent the (x, y)-coordinate, and the arrows represent the feasible heading directions* $\theta = \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{-3\pi}{4}, \frac{-\pi}{2}, \frac{-\pi}{4}\}$.




***(a)*** *A generated set of motion primitives. The different colors correspond to the feasible heading directions.*




***(b)*** *The generated motion primitive for heading direction $\theta = \frac{\pi}{2}$ shown in a state lattice with a grid resolution 0.5 meters.*

***Figure 2.3:*** *An example of generated motion primitives. (a) shows all heading directions $\theta = \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{-3\pi}{4}, \frac{-\pi}{2}, \frac{-\pi}{4}\}$ and (b) shows how the heading $\theta = \frac{\pi}{2}$ (purple) builds edges between discrete states. Only one heading is shown for clarity.*

edges that connect the states are called motion primitives, found by solving a set of boundary value problems (BVPs) offline [22]. Arbitrary obstacles are disregarded when computing the motion primitives since they are unknown until the vehicle drives online.

A position invariant system leads to easier calculations [22], where a motion primitive connecting two states also connects two other identically arranged states. It is only necessary to compute motion primitives from the origin to neighboring states since they are reused for all other states in the grid. This is viewed in Figure 2.3 where one set of motion primitives generated around the origin binds a discrete set of states.

### 2.4.2 Graph Search Algorithm

After a formulated discrete graph search problem, finding a feasible path requires a graph search algorithm. One such algorithm is the A* algorithm [23]. The algorithm starts with the initialization of a priority queue used to store states that have been encountered but not yet searched. The state with the highest priority in the queue, $s$, is accessed and searched for its neighboring states. If any neighboring states, $s'$, have not yet been encountered, they are added to the queue. The search is initialized by the starting state $s_{start}$. Termination of the search occurs when $s_{goal}$ becomes the highest-ranked element in the queue, or

when the queue is empty with no more states to search.

Each state has a connected cost, defined as the cumulative cost from $s_{start}$ to $s$. The cost of a neighboring state is the parent's cost plus the cost of going between them. The priority connected to a state is, in A*, the combination of the cost and a heuristic. The heuristic $h(s)$ is an estimated cost of going from $s$ to $s_{goal}$ and reduces the number of states to explore in a search. The choice of heuristic is application-based, but if $h(s)$ is an underestimate of the optimal cost to go for all $s$, then A* is guaranteed to find an optimal path [23].

# 3
# Method

The method is divided into four main sections. First a general description of the system architecture and how the hardware is set up. Then two methods are presented on how to achieve object detection. Thirdly a method for object tracking is presented and lastly the motion planning algorithm is described.

## 3.1 System Architecture

This section describes the structure of the software and hardware of the application.

### 3.1.1 Flowchart

Figure 3.1 shows the general structure of the application. It consists of one main program, one thread responsible for communicating with the software on the AGV, and one thread running the object detection, object tracking, and obstacle avoidance.

### 3.1.2 Hardware Setup

The communication between the application to software on the AGV is with TCP socket. M03933 by IFM is the camera head used with the application, located on the front of the AGV. It works together with the computer M03975, its python library ifm3dpy [24] and must be calibrated before use.

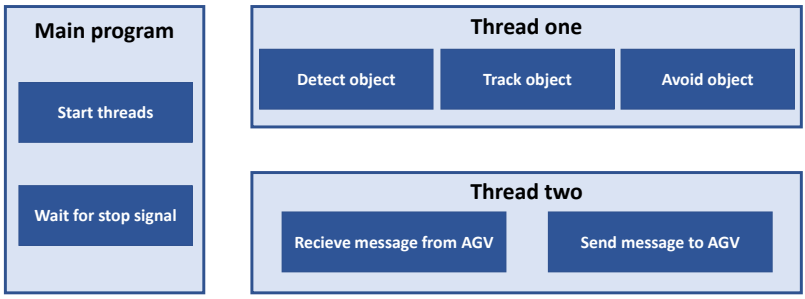



***Figure 3.1:*** *General structure of the application.*

## 3.2 Vision Based Object Detection

The following chapter describes the investigation of two methods to detect objects, one using fiducial markers and the other an object detection algorithm with 2D-bounding boxes.

### 3.2.1 Fiducial Markers

A fiducial marker is a square consisting of a binary code that builds up a unique identification [25]. The idea is to place one or several fiducial markers on the object and then detect the marker instead of the object itself. The Open Source Computer Vision library ArUco Marker Detection [26] is used in the implementation. The library builds on the ArUco Library by Garrido-Jurado et al. [25].

The first step is to generate a marker by choosing a predefined dictionary in the ArUco module. The chosen dictionary for this thesis consists of 250 markers, each with a size of 6x6 bits. The size of the marker image in pixels is also specified. Lastly, the marker ID is chosen, valid between 0-249 for this dictionary. A generated ArUco marker from this dictionary is shown in Figure 3.2.

Once a marker is generated and placed on the object, it is possible to detect it in each image frame as long as the camera's resolution is good enough to see the square shape of the tag and the inner codification. The output from detection is the position of the marker's four corners in the image and its ID. Once the corners of the marker are detected, it is possible to do pose estimation. A 3D transformation from the marker coordinate system to the camera coordinate system is done, including a rotation and translation. A successful estimation requires a known

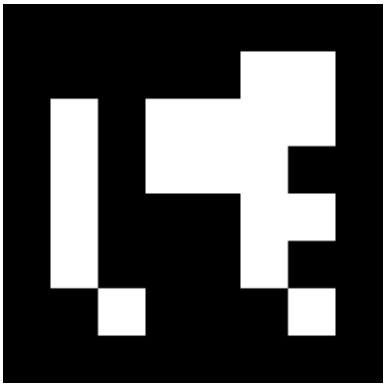


***Figure 3.2:*** *An example of a generated ArUco tag with marker ID 1.*

marker size and camera matrix, the second found during camera calibration. After these steps, it is possible to detect markers placed on an object and find their poses described in the coordinates of the camera coordinate system.

### 3.2.2 Data Acquisition

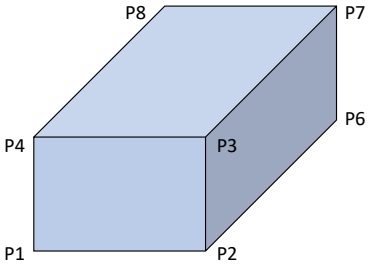
The object detection algorithm requires, in the training stage, data from test runs. This data includes images of the object as well as corresponding labels. Since no cloud services are allowed for confidentiality reasons, and the time frame for data gathering is small, labeling using web-GUI is disregarded. Instead, a semi-automatic solution for labeling the bounding boxes live during test runs is designed.

The idea is to place the object in a known location in world coordinates and measure the eight corners that span the 3D object. The corners are then expressed in pixel coordinates if a series of coordinate-system transformations are applied. A selection of the eight corners will always span up the 2D-bounding box in pixel values, as shown in Figure 3.3.

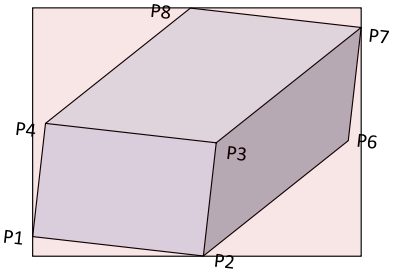
Three coordinate system transformations are needed to go from world coordinates to pixel coordinates. The first is between the world coordinate system and the AGV coordinate system. The position of the AGV's origin ($d_{x,AGV}$, $d_{y,AGV}$, $d_{z,AGV}$), as well as its orientation $\theta_{AGV}$, are known variables. The coordinate system rotates around the z-axis, as seen in Figure 3.4a. The combined translation and rotation relation is described with:

$$\begin{bmatrix} P_{x,world} \\ P_{y,world} \\ P_{z,world} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_{AGV}) & -\sin(\theta_{AGV}) & 0 & d_{x,AGV} \\ \sin(\theta_{AGV}) & \cos(\theta_{AGV}) & 0 & d_{y,AGV} \\ 0 & 0 & 1 & d_{z,AGV} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{x,AGV} \\ P_{y,AGV} \\ P_{z,AGV} \\ 1 \end{bmatrix}. \tag{3.1}$$

The second coordinate system transformation is between the AGV coordinate sys-

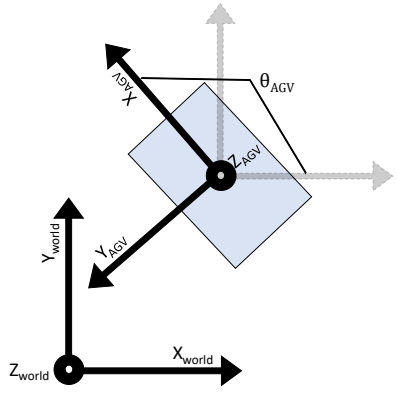



*(a) The eight corners that span up the entire 3D object.*

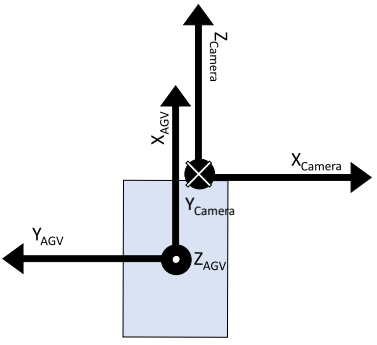



*(b) From this view the corners P1, P2, P7 and P8 span up the 2D-bounding box.*

***Figure 3.3:*** *In (a) the eight corners are shown and (b) is an example that shows that four corners will span up the 2D-bounding box.*




*(a) The AGV coordinate system and the world coordinate system.*




*(b) The AGV coordinate system and the camera coordinate system.*

***Figure 3.4:*** *The coordinate systems in relation to each other.*

tem and the camera coordinate system, as seen in Figure 3.4b. The rotation and translation are fixed as long as the camera position does not change. The relation is described with:

$$P_{x,camera} = -P_{y,AGV} + \text{offset}_{side}, \tag{3.2a}$$

$$P_{y,camera} = -P_{z,AGV} + \text{offset}_{ground}, \tag{3.2b}$$

$$P_{z,camera} = P_{x,AGV} - \text{offset}_{front}. \tag{3.2c}$$

The third coordinate system transformation is the camera coordinate system to the pixel coordinate system. This transformation is not invertible since information is lost going from 3D to 2D. The camera matrix found during calibration is used, and the relation is described with:

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{x,camera} \\ P_{y,camera} \\ P_{z,camera} \end{bmatrix}, \tag{3.3}$$

where $u$ and $v$ are the pixel coordinates. Transferring from Homogeneous coordinates to Cartesian coordinates is done by dividing the first two elements with the factor $w$. A summary of the semi-automatic labeling during the test runs process is found in Algorithm 1. Observe step 1, where removing the fiducial markers is crucial to avoid unwanted features in the data.

**Algorithm 1** Semi-automatic labeling during test runs with stationary object location.

**Step 1: Initialize by finding the world coordinates of the corners spanning the 3D object.**

- Place fiducial markers on the corners and detect them with the camera.
- Transform the detected markers from the camera coordinate system to the world coordinate system.
- Remove fiducial markers.

**Step 2: Find the corresponding pixel values of the 3D-object corners.**

- Transform the corners from the world coordinate system to the AGV coordinate system.
- Transform the corners from the AGV coordinate system to the camera coordinate system.
- Transform the corners from the camera coordinate system to pixel values.

**Step 3: Find the bounding box.**

- Find the center, width and height of the bounding box by checking which of the corners have the largest and smallest pixel-coordinates both in $u$ and $v$ direction.

**Step 4: Save the image and corresponding bounding box label.**

- Move the AGV to a new position and go back to step 2 again.

### 3.2.3 Training a Network for Object Detection

YOLOv5s, a small but fast version of YOLO designed to compute detection with higher speed but lower accuracy than the larger versions available, is the chosen model for object detection. The forks on the vehicle are chosen as the object to train with, as they are close to the ground and visible to the camera at smaller distances than the whole forklift. The weights used for training are yolov5s from Jocher [27], pre-trained for the COCO dataset.

Training data is created by acquiring 1000 images and their labels using the method earlier described. 900 images are of forks positioned in several world locations of the given laboratory space to create different backgrounds with both humans, clutter, and other types of forklifts. 100 images consist of only background to reduce false positives [28]. Labels with bounding box coordinates exceeding the frame size are removed. The resulting training dataset is of 922 images, of which 822 are forks.

250 additional images are collected, half split into validation data and the other test data. The acquisition occurs during a separate run, with new positions in the factory. 30 background images are added to each of the splits, resulting in a division of approximately 80 percent training data, 10 percent validation data, and 10 percent test data, a general recommendation [29].

The model is trained with the training dataset for 300 epochs with no alterations to the hyperparameters, a recommendation from Ultralytics [28] when using YOLO models. The validation data is used after every trained epoch to validate the created weights.

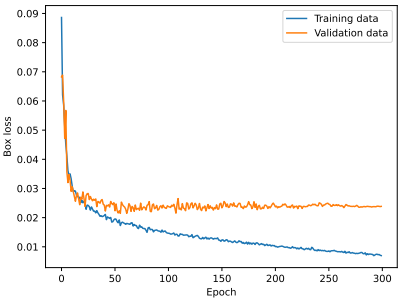
The box loss for both training and validation data is shown in Figure 3.5a. The box loss for the training data continuously decreases while the box loss for the validation data converges to a constant value from approximately epoch 75, a sign of overfitting. The objectness loss for both training and validation data is shown in Figure 3.5b, where it is evident that the two curves diverge after epoch 75 and that the model is overfitted.

As recommended [28], the number of epochs is decreased. A value of 75 epochs, motivated by the start of the curves diverging, is chosen and the model is trained again on the same dataset as before with the same pre-trained weights and hyperparameters.

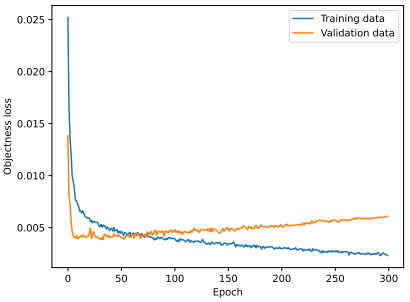
The best weights from this training are used for object detection. For evaluation of the final trained model and its weights, precision and recall are used together to view the AP of the test data. This is presented in Section 4.1.

## 3.3 Object Tracking Using a Monocular Camera

One object tracking algorithm uses the detected fiducial marker as measurement, and the other uses the bounding box output from detection with the trained YOLOv5s model.




*(a) Box loss (Bounding box regression loss) for the training data continuously decreases while the box loss for the validation data converges at approximately epoch 75, the model shows sign of overfitting.*




*(b) Objectness loss for the training data and validation data diverge at approximately epoch 75, the model shows sign of overfitting.*

***Figure 3.5:*** *Loss functions for training YOLOv5s 300 epochs with pretrained weights yolov5s.*

### 3.3.1 Tracking of 3D-object with 2D-bounding Boxes

Reich and Wuensche [30] present a method for object tracking using an Extended Kalman filter (EKF). The initialization of the object state in 3D space, and the corresponding initial covariance, is found using a monocular 3D object tracker. Measurements of 2D-bounding boxes are used in the following time steps to improve the current state estimate in the update step of the EKF. The focus of this master thesis is the tracking using 2D-bounding boxes after the state initialization.

A coordinated turn model describes the motion of the tracked object. This model is based on curvilinear particle motion and is described fully by Roth et al. [31]. The state components include the position in the horizontal plane $x, y$ [m], the magnitude of the velocity vector $v$ [m/s], the heading angle $\theta$ [rad] and turn rate $\omega = \frac{d\theta}{dt}$ [rad/s]. The motion can, in discrete time with velocity in polar coordinates, be described with:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{k+1} \\ \theta_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \frac{2v_k}{\omega} \sin(\frac{\omega_k T}{2}) \cos(\theta_k + \frac{\omega_k T}{2}) \\ y_k + \frac{2v_k}{\omega_k} \sin(\frac{\omega_k T}{2}) \sin(\theta_k + \frac{\omega_k T}{2}) \\ v_k \\ \theta_k + \omega_k T \\ \omega_k \end{bmatrix} + \begin{bmatrix} \frac{T^2}{2}\cos(\theta_k) & 0 \\ \frac{T^2}{2}\sin(\theta_k) & 0 \\ T & 0 \\ 0 & \frac{T^2}{2} \\ 0 & T \end{bmatrix} \begin{bmatrix} a_k \\ \alpha_k \end{bmatrix} \tag{3.4}$$

where $a$ [m/$s^2$] is the linear acceleration and $\alpha$ is the rotational acceleration [rad/$s^2$]. Here it is assumed that $a$ and $\alpha$ are zero during the discretization and constant between sampling instants. The time update phase of the EKF uses this

discretized motion model when predicting the state update. The covariance update depends on a linearized version of (3.4). The design parameter Q is based on the standard deviation of the linear acceleration and rotational acceleration according to:

$$Q = \begin{bmatrix} \sigma_a & 0 \\ 0 & \sigma_\alpha \end{bmatrix}^2. \tag{3.5}$$

The standard deviation of linear- and rotational acceleration of the object is not known, but normal values of linear acceleration for the AGV are between 0.5 to -1.0 $m/s^2$, and a normal upper bound for the rotational acceleration of approximately 1.5 $rad/s^2$.

The measurements entering the EKF in the measurement update phase are on the form:

$$y_{2D} = \begin{bmatrix} u_{min} \\ v_{min} \\ u_{max} \\ v_{max} \end{bmatrix}. \tag{3.6}$$

The first two elements correspond to the upper left corner in pixel coordinates and the last two to the lower right corner. The measurement equation projects the state estimate into the measurement space to make the state estimate and measurement comparable:

$$\hat{y}_{2D} = h_{2D}(\hat{x}_{k|k-1}). \tag{3.7}$$

All eight corners that span the object are found from the tracked pose in global coordinates. They are transformed from the global coordinate system to the camera coordinate system and projected onto the image plane. A minimum enclosing 2D bounding box is calculated from the eight corners, done similarly as in Section 3.2.2. This measurement equation is used when updating the state estimate. To update the covariance estimate and Kalman gain, a linearized version of the measurement equation is used instead.

The design parameter $R_{2D}$ describes the uncertainty in the measurement:

$$R_{2D} = \begin{bmatrix} \sigma_{u_{min}} & 0 & 0 & 0 \\ 0 & \sigma_{v_{min}} & 0 & 0 \\ 0 & 0 & \sigma_{u_{max}} & 0 \\ 0 & 0 & 0 & \sigma_{v_{max}} \end{bmatrix}^2. \tag{3.8}$$

Reich and Wuensche [30] observed increased noise in image coordinates for closer, and therefore larger, objects and suggested a standard deviation of $\frac{200}{d}$, where $d$ is the distance from the camera origin to the object's center. It gives a standard deviation of 100 pixels for the bounding box corners if the distance to the object is 2 meters and 50 pixels if the distance to the object is 4 meters.

### 3.3.2 Tracking a Fiducial Marker

When using fiducial markers in the tracking, the time update phase of the Extended Kalman filter is the same as described in Section 3.3.1, since the motion model of the tracked object is the same. The measurement update phase is, however, different. It enters the EKF linearly in this case since it is possible to directly access measurements of the states $x$, $y$, and $\theta$ after the output from the detected fiducial marker is transformed to global coordinates. This is described with:

$$\hat{y}_{marker} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{v}_{k|k-1} \\ \hat{\theta}_{k|k-1} \\ \hat{\omega}_{k|k-1} \end{bmatrix}. \tag{3.9}$$

The design parameter $R_{marker}$ describes measurement uncertainty. The fiducial marker has a higher uncertainty in the rotation measurement than the distance measurement, which this thesis takes into account when choosing the magnitude of the elements.

## 3.4 State Lattice Motion Planning

The obstacle avoidance is done with a lattice planner and includes constructing the state lattice, generating motion primitives, and finding a good graph search algorithm.

### 3.4.1 Building Outer Bound of the Object

Creating an outer bound of a detected object based on the estimated states is possible since its dimensions are known. Using the information in the covariance matrix of the state estimate, the uncertainty in $x$, $y$, and $\theta$ is considered when creating the outer bound. All elements concerning these three states are extracted and build a 3x3 sub-matrix:

$$P = \begin{bmatrix} \sigma_x^2 & \sigma_{x,y} & \sigma_{x,\theta} \\ \sigma_{y,x} & \sigma_y^2 & \sigma_{y,\theta} \\ \sigma_{\theta,x} & \sigma_{\theta,y} & \sigma_\theta^2 \end{bmatrix} \tag{3.10}$$

Since the state estimate covariance is in global coordinates, the covariance $P$ is transformed to the object's coordinate system using the rotation matrix $R$:

$$P_R = RPR^T. \tag{3.11}$$

As a simplification, the covariance between states is assumed to be equal to zero,

and only the diagonal elements of $P_R$ are used. It results in the covariance matrix:

$$P_{R,d} = \begin{bmatrix} \sigma^2_{x,object} & 0 & 0 \\ 0 & \sigma^2_{y,object} & 0 \\ 0 & 0 & \sigma^2_{\theta,object} \end{bmatrix}. \tag{3.12}$$

The standard deviations in position and heading can be extracted and described in the object's coordinate system.

The uncertainty in the x-coordinate, y-coordinate, and heading is added as object width and length. It is assumed that the estimated state differ with an absolute distance $\tilde{\sigma}_{x,object}$, $\tilde{\sigma}_{y,object}$ and absolute heading $\tilde{\sigma}_{\theta,object}$, based on one standard deviation. The outer bound of the object, with the added length $L'$ and width $W'$, is illustrated in Figure 3.6. The geometric definitions for the extra length and width are:

$$L^{'} = |(L + \tilde{\sigma}_{x,object})\cos(\tilde{\sigma}_{\theta,object}) + (W + \tilde{\sigma}_{y,object})\sin(\tilde{\sigma}_{\theta,object}) - L|, \tag{3.13a}$$

$$W^{'} = |(L + \tilde{\sigma}_{x,object})\sin(\tilde{\sigma}_{\theta,object}) + (W + \tilde{\sigma}_{y,object})\cos(\tilde{\sigma}_{\theta,object}) - W|. \tag{3.13b}$$

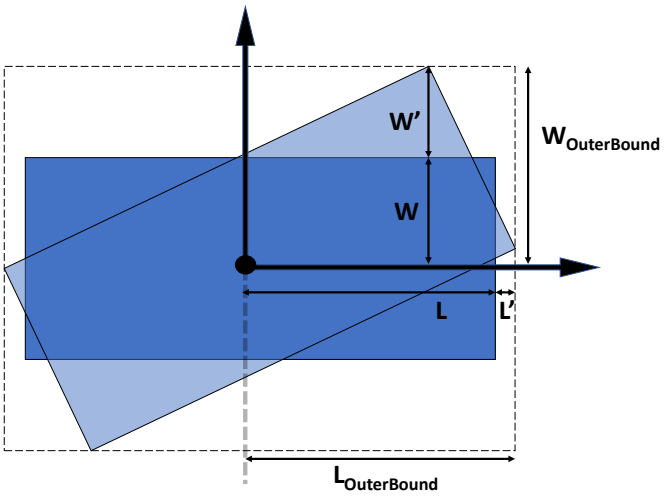



***Figure 3.6:*** *The extra length and width caused by uncertainty in the state estimate.*

The entire outer bound of the obstacle is described from the object's origin:

$$L_{OuterBound} = L + L^{'}, \tag{3.14a}$$

$$W_{OuterBound} = W + W^{'}, \tag{3.14b}$$

and the change in width and length are symmetrical.

### 3.4.2 Checking if an Obstacle is Obscuring a Point

In the implementation, it is more calculation heavy to check if a point is inside an object-oriented bounding box than in an axis-aligned bounding box. The first step is therefore based on a relaxed condition to check if the point is somewhere in the smallest axis-aligned bounding box covering the object-oriented bounding box: the light gray area in Figure 3.7. If the point is inside the gray area, the next step is to see if it also exists in the red one by checking the scalar products of vectors. Point P is inside the red area if:

$$(0 < \vec{P_1P} \cdot \vec{P_1P_2} < \vec{P_1P_2} \cdot \vec{P_1P_2}) \wedge (0 < \vec{P_1P} \cdot \vec{P_1P_4} < \vec{P_1P_4} \cdot \vec{P_1P_4}). \tag{3.15}$$

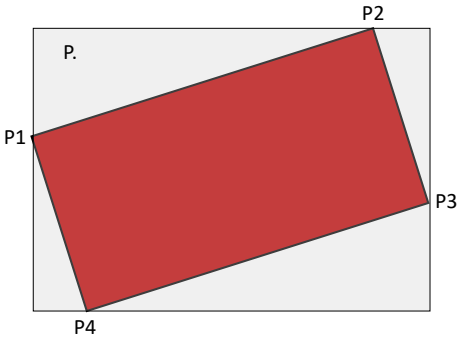



***Figure 3.7:*** *The red area is the occupied space, and the grey area is the axis-aligned bounding box covering the object-oriented bounding box. P is the point to check if located in the occupied space.*

### 3.4.3 Design of the State Space Lattice

The application is developed in a laboratory space of approximately 12 by 12 meters, motivating these dimensions for the state space lattice. The number of discrete orientations is set to eight:

$$\theta = \left\{ -\frac{3\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{2} \right\}, \tag{3.16}$$

and the grid resolution to 0.5 meters ad hoc. It gives good enough routes without the computation time becoming too large for real-time application. A larger grid resolution results in less computation time since the graph search problem becomes smaller.

The AGV uses differential drive kinematics described with a differential drive model. A unicycle model is a simplified version of the differential drive model [23] and happens to be easier to use in this implementation. The unicycle model [32]

is described by:

$$\dot{x} = v\cos(\theta), \tag{3.17a}$$
$$\dot{y} = v\sin(\theta), \tag{3.17b}$$
$$\dot{\theta} = \omega, \tag{3.17c}$$

where $x$, $y$, and $\theta$ are the AGV's position and orientation. The input controls include the linear velocity $v$ and the angular velocity $w$.

The motion primitives are solved offline using CasADi [33] together with the nonlinear solver IPOPT [34]. The optimization problem consists of two steps. An initial motion primitive is solved, where the aim is to move the robot one step in the discrete orientation space. Since this solution does not guarantee that the motion primitive will end on one of the discretized states, another optimization is performed with the first one as the initial guess. Straight maneuvers use the same procedure. Figure 3.8 shows the generated motion primitives located on the generated state lattice.

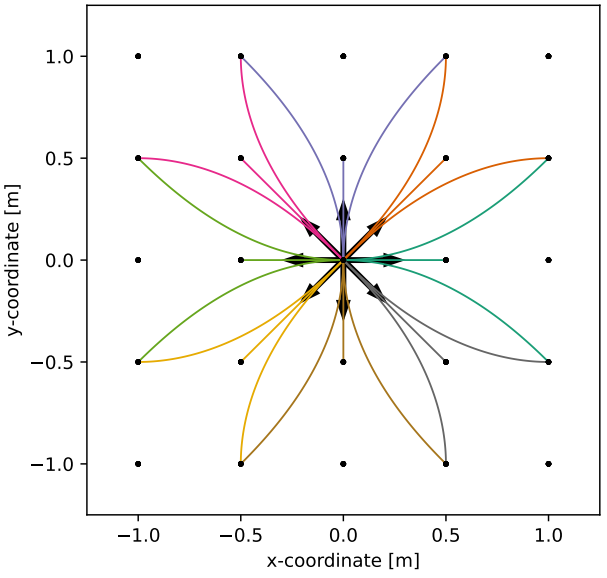



***Figure 3.8:*** *The generated motion primitives with a grid resolution of 0.5 meters. The black arrows show the eight discrete orientations* $\theta = \{-\frac{3\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{2}, \pi\}$.

### 3.4.4 Graph Search Algorithm

A*, described in Section 2.4.2, was implemented to perform the graph search but some scenarios generated overly large calculation times. Since the goal is compatibility with real-time applications, it was necessary to limit the computation time. ARA* (Anytime Repairing A*) [35], an extension of A* designed to find a suboptimal solution quickly before tuning during the rest of the available search time, became the new choice. If time allows, ARA* will find an optimal solution. To compute efficiently, it reuses previous search efforts and does not recalculate from scratch during the iterations.

ARA* uses three lists, $OPEN$, $CLOSED$, and $INCONS$, where $OPEN$ is a priority queue. It executes A* multiple times by starting with a high value of the factor $\epsilon$ that inflates the heuristic $h(s)$ and yields faster searches with fewer state expansions. The heuristic chosen in this implementation is the euclidean distance from a state $s$ to a goal state $s_{goal}$. The cost to go between a state $s$ to a succeeding state $s'$ is defined with $c(s, s')$. The cumulative cost of the current path from the starting state $s_{start}$ to state $s$ is defined with $g(s)$. The term locally inconsistent, central in ARA*, describes when a decrease in $g(s)$ introduces a local inconsistency between that g-value and the g-value of its successors. When the state is expanded, this inconsistency is corrected by re-evaluating the g-values of the successors. $\epsilon'$ is a suboptimality bound. A description of ARA* in pseudocode is found in Algorithm 2. The interested reader is referred to [35] for a full explanation of the algorithm.

**Algorithm 2** ARA*-algorithm in pseudocode.

```
procedure fvalue(s)
  return g(s) + ϵh(s)

procedure ImprovePath()
while fvalue(s_goal) > min_{s∈OPEN} fvalue(s) do
    s ← OPEN.GetPrioritizedElement()
    CLOSED.insert(s)
    for each successor s' of s do
        if s' not visited before then
            g(s') = ∞
        end if
        if g(s') > g(s) + c(s, s') then
            g(s') = g(s) + c(s, s')
            if s' ∉ CLOSED then
                OPEN.insert(s', fvalue(s'))
            else
                INCONS.insert(s')
            end if
        end if
    end for
end while

procedure Main()
g(s_goal) = ∞; g(s_start) = 0
OPEN = CLOSED = INCONS = ∅
OPEN.insert(s_start, fvalue(s_start))
ImprovePath()
ϵ' = min(ϵ, g(s_goal)/ min_{s∈OPEN∪INCONS} (g(s) + h(s)))
Publish current ϵ'-suboptimal solution
while ϵ' > 1 do
    decrease ϵ
    OPEN ← OPEN ∪ INCONS
    Update priorities fvalue(s) for all s ∈ OPEN
    CLOSED = ∅
    ImprovePath()
    ϵ' = min(ϵ, g(s_goal)/ min_{s∈OPEN∪INCONS} (g(s) + h(s)))
    Publish current ϵ'-suboptimal solution
end while
```

# 4

# Result and Discussion

In the following chapter the results are presented and discussed. First the vision based object detection, secondly the object tracking and thirdly the state lattice motion planning.

## 4.1 Vision Based Object Detection

Together with the best weights gathered after training YOLOv5s for 75 epochs, the test data set is used to evaluate the final model. The model correctly labels every pair of forks in the test data, and no forks are found in the background images. The model has both precision and recall values equal to 1. The AP is 0.995 for the confidence threshold range 0:0.5 and 0.714 for 0.5:0.95.

In the solution of this thesis, four different ArUco markers were placed on the corners of the forks to estimate the object's location in the semi-automatic labeling algorithm. Moving the object became time-consuming since the ArUco markers had to be relocated for each new location. It caused a limitation on the number of fork positions in the global world of the laboratory space.

If the object's location in the world was known and automatically updated when altering the positioning of the forks, an increased number of images and corresponding labels with more diverse features in different environments could have been gathered. This would lead to less correlated images.

For time efficiency, a single ArUco marker initially estimated an object's position. Based on the position and rotation measurement from the marker, it is theoretically possible to find the four corners when an object's dimensions are known. It would have given an efficient solution for finding the global location and an

increased number of possible fork positions used for the data gathering. This solution did not work in the current status of this thesis. The ArUco marker rotation measurements were too noisy to give an accurate approximation of the real-world location, which led to a faulty minimal bounding box of the object.

## 4.2 Object Tracking Using a Monocular Camera

The object tracking is done using ArUco marker measurements and 2D-bounding box measurements from YOLOv5s, separately in two different Extended Kalman filters. The two results are compared with both each other and the raw ArUco marker measurements. This is done since the actual position and heading of the object are unknown and no ground truth values are available. First, the set-up of the tests is explained in more detail and potential problems and limitations are brought up. Then three subcategories of test cases and their respective results are shown and discussed.

### 4.2.1 Planning and Executing the Tests

Since one limitation with this master thesis is the time available in the lab facility, the tests are divided into an online and offline part. The computer used is also a limiting factor, leading to a long sampling time. The application was never implemented on the planned GPU accelerated computer M03975, because of time limitations. A long sampling time can cause problems since the object's position and rotation can change by a large amount in a short time. This is especially problematic if there are several bad measurements in a row. To compensate for this phenomenon, the object was driven at an unnaturally slow velocity during the online data gathering.

#### Online Data Gathering

The measurements are gathered in the lab facility with a sampling time of 0.5 seconds. After each test, the following data is saved to files:

- Measurements of the AGV's position and heading expressed in world coordinates.
- Measurements from the ArUco marker transformed live from camera coordinates to world coordinates. This includes both position and heading.
- 2D-bounding box measurements from detections with YOLOv5s.

Both the measurements from the ArUco marker and the 2D-bounding box from the detection with YOLOv5s are generated on the same camera frame. The ArUco marker is located on the rotation center of the object since that is the chosen point to track based on what is most compatible with the coordinated turn model.

#### Offline Calculations and Tuning

After the online data gathering, the rest of the calculations are performed offline. This includes using the raw measurements as input to each Extended Kalman filter. This is done with one sample at a time, to simulate how it would work in an online setting. The first available ArUco marker measurements are set as the initial values of position and heading for the Extended Kalman filter using ArUco marker measurements. For the Extended Kalman filter using 2D-bounding box measurements, the filtered ArUco marker positions are used instead. The initial heading is set manually in this case and changes between tests since the filtered ArUco marker measurement of heading is considered too noisy to use as an initial value for this filter. The last two states, velocity, and angular velocity are set manually close to zero since all tests are designed to start from a standstill.

The parameters $Q_{ArUco}$, $R_{ArUco}$, $Q_{2D}$, and $R_{2D}$ are also tuned offline, to find values that yield good performance on the tests. The values are presented in a table at the start of each subcategory of test cases. The initial state covariance is also presented there.

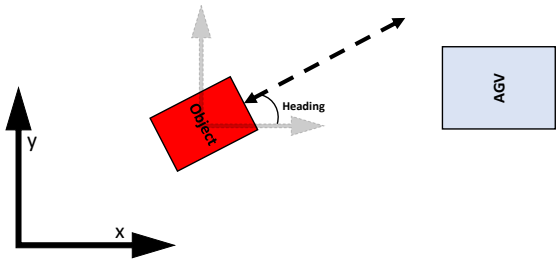
### 4.2.2 Tests with Zero Angular Velocity

The first subcategory of tests includes tests that have zero angular velocity. These tests show how well the filters perform when the heading is constant, and hence the focus is on how well the filters can notice changes in position. To make sure that the filter uses the measurements, and not only bases the estimation on the underlying model, the motions are performed repetitively with changing directions, and hence the velocity both changes size and sign. Three tests are performed in this subcategory and can all be seen in Figure 4.1. The tuned parameters for this subcategory of tests can be found in Table 4.1. $R_{2D}$ is based on the discussion from Section 3.3.1.

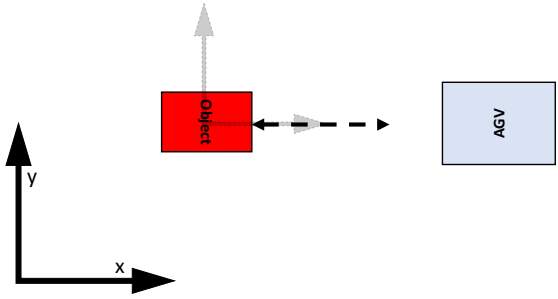
***Table 4.1:*** *Tuning parameters and initial state covariance for tests in the subcategory with zero angular velocity.*

| | **ArUco marker** | **2D-bounding box** |
|---|---|---|
| **Q** | $\text{diag}(0.2, 0.1)^2$ | $\text{diag}(1.5, 0.1)^2$ |
| **R** | $\text{diag}(0.4, 0.4, 2\pi/3)^2$ | $\text{diag}(200/d, 200/d, 200/d, 200/d)^2$ |
| **P0** | $\text{diag}(0.25, 0.25, 0.1, \pi/2, 0.1)^2$ | $\text{diag}(0.25, 0.25, 0.1, \pi/2, 0.1)^2$ |

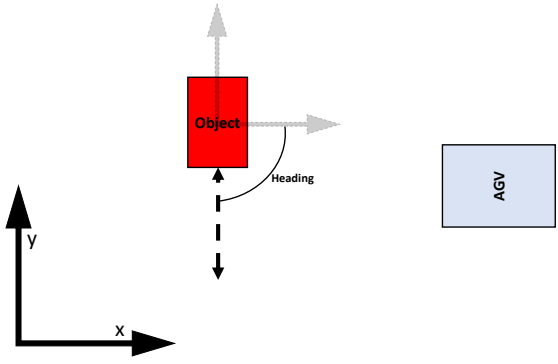
Figure 4.1a shows how the first test in this subcategory is designed. The object moves diagonally in front of the AGV with a constant heading in a repetitive manner. This test yields measurements that are based on a change in both the x-coordinate and y-coordinate. The 2D-bounding box measurements are observed in Figure 4.2, they are stable which indicates no large outliers. Figure 4.3 shows the resulting signals of position and heading. The raw ArUco marker measurements are noisier in heading than in position, getting the largest improvement after filtering. It is also seen that the filtered signal from the ArUco measure-




***(a)*** *The object moves diagonally in front of the AGV with a constant relative heading.*




***(b)*** *The object moves straight toward the AGV with a relative heading of zero.*




***(c)*** *The object moves with a constant relative heading, in a ninety-degree rotation compared to the AGV.*

***Figure 4.1:*** *Three tests with a constant relative heading. Each motion is executed several times.*

ments works well with smoothing out the noisy heading measurements. The filter using the 2D-bounding boxes also gives a smooth estimate of the heading. It is likely possible to optimize both filters' behavior if the parameters are tuned more thoroughly and not ad hoc. All three signals follow a similar trend when looking at the x- and y-coordinate with only small differences. It is not possible to say which one performs the best since no real ground truth values are available for reference. However, both filters pick up on changes in both directions, which was the purpose of this test.

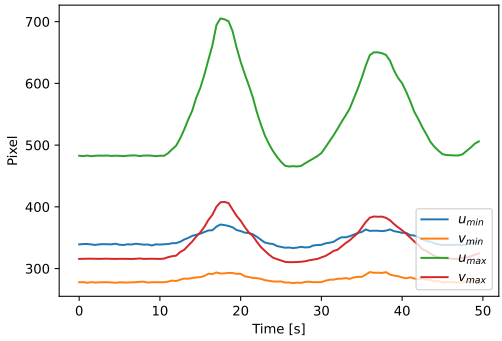
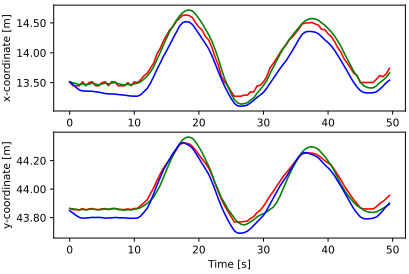


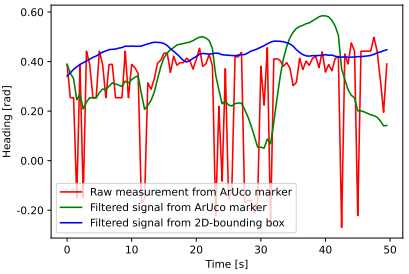

***Figure 4.2:*** *The bounding box measurements when the object moves diagonally in front of the AGV, as described in Figure 4.1a.*

Figure 4.1b shows how the second test in this subcategory is designed. The object moves straight toward the AGV and hence should only yield small differences in the y-coordinate. It is however noted that the object is moved manually, and small differences will occur. In Figure 4.4 the 2D-bounding box measurements are stable which indicate no real outliers. Similar results as the first test are observed in Figure 4.5. The filtered heading signal, generated from the measurements of the 2D-bounding boxes, gives an even smoother signal than the signal filtered from the raw ArUco measurements. Both filters pick up on the change in the x-coordinate, and only small changes occur in the y-coordinate, which is what is expected from this test.

The third and last test in this subcategory includes seeing the object passing in front of the AGV with a constant rotation of 90 degrees. This can be seen in Figure 4.1c. This is a test where the ArUco marker can not be used normally since it faces the same direction as the heading of the object. However, for this test, the marker is moved so that the raw measurements of the x-, and y-coordinates and heading can be used as a rough comparison. The 2D-bounding box measurements are viewed in Figure 4.7, and it is seen that they are stable, which indicates no large outliers. The object is moving with an approximately constant heading of $-\frac{\pi}{2}$ radians and the results are viewed in Figure 4.6. Overall, the tracking correctly captures the general trend in the x-, y-coordinate, and heading and it is seen that the filtered signal approximately follows the expected heading




*(a) A comparison of the resulting signals in x- and y-direction respectively.*




*(b) A comparison of the resulting signals of heading.*

***Figure 4.3:*** *The result when the object moves diagonally in front of the AGV, as described in Figure 4.1a.*

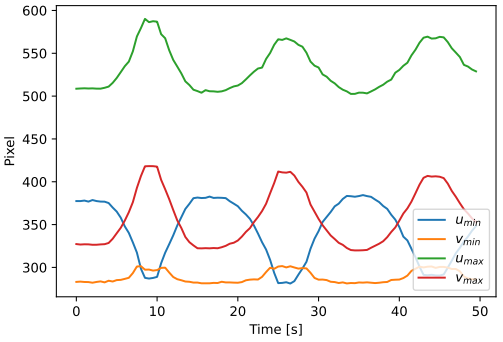
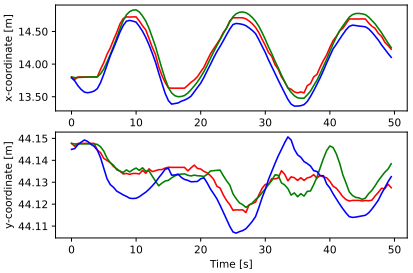
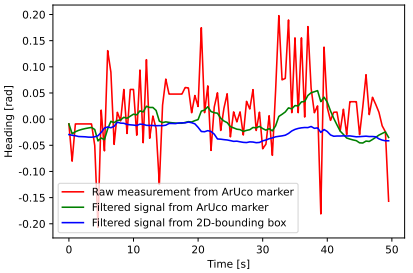



***Figure 4.4:*** *The bounding box measurements when the object moves straight toward the AGV, as described in Figure 4.1b.*

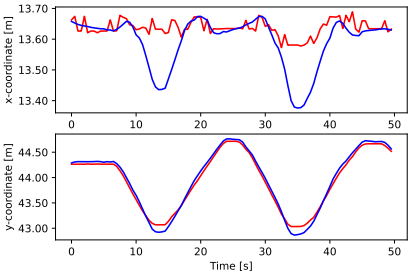



***(a)*** *A comparison of the resulting signals in x- and y-direction respectively.*

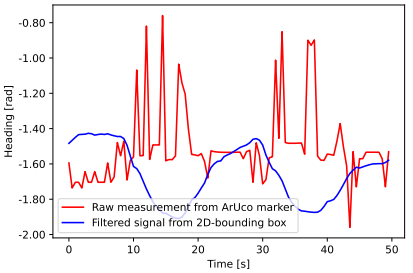



***(b)*** *A comparison of the resulting signals of heading.*

***Figure 4.5:*** *The result when the object moves straight toward the AGV, as described in Figure 4.1b*




***(a)*** *A comparison of the resulting signals in x- and y-direction respectively.*




***(b)*** *A comparison of the resulting signals of heading.*

***Figure 4.6:*** *The result when the object moves with a relative 90-degree rotation to the AGV, as described in Figure 4.1c.*

of $-\frac{\pi}{2}$. However, smaller fluctuations are noted both in the heading and in the x-coordinate, the trend occurring at the same time in both signals. When looking at the measurements in Figure 4.7, these fluctuations occur when the object changes direction on one of the extreme points. Another thing worth noting is that the filtered signal using 2D-bounding boxes is capable of performing well on a test that the ArUco marker is incapable of generating a result on. This is only true for the chosen current location of the marker, but no matter where the marker is placed, there will be scenarios where it can not be seen by the camera. It is therefore concluded that the 2D-bounding box measurements give a more versatile filter where the only limitation is that the object is within the camera frame.

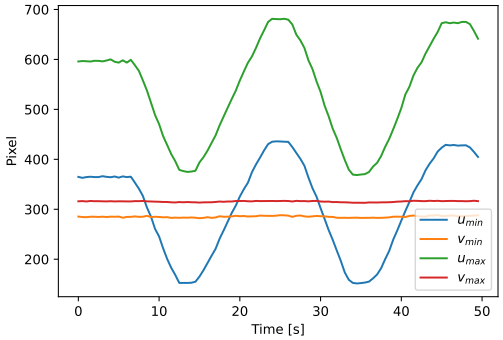



***Figure 4.7:*** *The bounding box measurements when the object moves with a relative 90-degree rotation to the AGV, as described in Figure 4.1c.*

### 4.2.3 Tests with Non-Zero Angular Velocity

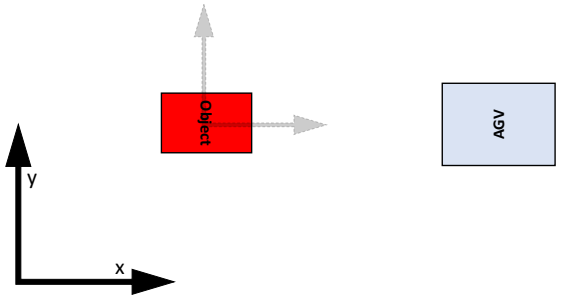
The second subcategory only includes one test. This test has non-zero angular velocity, but zero linear velocity. It will show how well the filters perform when the position is constant, and hence the focus is on changes in heading. The tuned parameters for this subcategory of tests can be found in Table 4.2. To see any change in rotation, the second element of the $Q_{2D}$-parameter representing the standard deviation of rotational acceleration needs to be increased by a large amount compared to the previous tests. This increase does however not yield good results on the tests with zero angular velocity. This indicates that the filter in the first subcategory mainly bases the estimate of heading on the model, and when the estimate is more based on the measurement, problematic behaviors occur. One potential reason for this will be investigated in this second subcategory.

***Table 4.2:*** *Tuning parameters and initial state covariance for tests in the subcategory with non-zero angular velocity.*

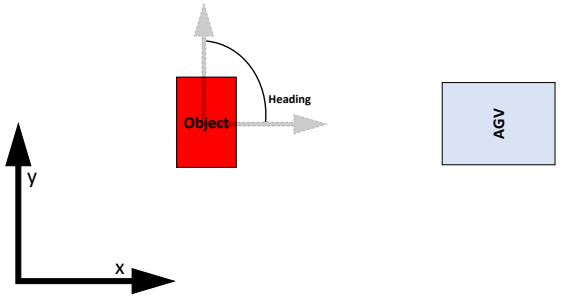
| | **ArUco marker** | **2D-bounding box** |
|---|---|---|
| **Q** | $\text{diag}(0.2, 0.1)^2$ | $\text{diag}(1.5, 15)^2$ |
| **R** | $\text{diag}(0.4, 0.4, 2\pi/3)^2$ | $\text{diag}(200/d, 200/d, 200/d, 200/d)^2$ |
| **P0** | $\text{diag}(0.25, 0.25, 0.1, \pi/2, 0.1)^2$ | $\text{diag}(0.25, 0.25, 0.1, \pi/2, 0.1)^2$ |

The test includes a counter-clockwise rotation of the object, where the object moves from a relative heading of zero to a positive $\frac{\pi}{2}$ radians, shown in Figure 4.8. The ArUco marker does not yield measurements for large rotations and the results are shown in Figure 4.9.

The test generates a heading estimate with the wrong sign on the filtered signal using the 2D-bounding box measurements, compared to the filtered signal from the ArUco marker measurements. The reason behind this is found if the 2D-bounding box measurements from both a clockwise and an anti-clockwise ro-

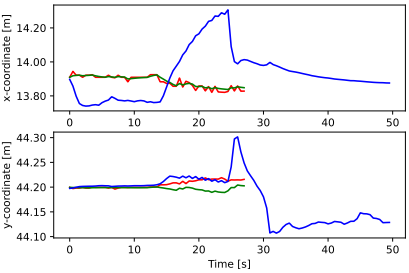



***(a)*** *The start position with relative heading equal to zero.*

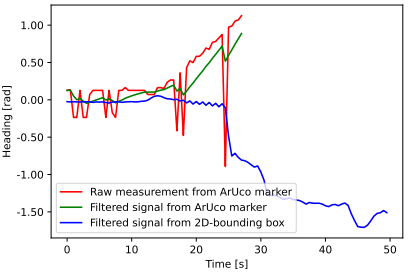



***(b)*** *The object is rotated from the start position in (a) to a relative heading of* $\frac{\pi}{2}$.

***Figure 4.8:*** *The set-up for the test with non-zero angular velocity.*




*(a) A comparison of the resulting signals in x- and y-direction respectively.*




*(b) A comparison of the resulting signals of heading.*

***Figure 4.9:*** *The result when the object rotates counter-clockwise, as described in Figure 4.8b.*

tation are shown together as in Figure 4.10. These bounding box measurements are found when the object:

1. Has a heading of zero radians, from approximately time 0 to time 5
2. Moves to a positive $\frac{\pi}{2}$ radians, from approximately time 5 to time 25
3. Moves back to zero radians, from approximately time 25 to time 45
4. Moves to a negative $\frac{\pi}{2}$ radians, from approximately time 45 to time 70
5. Moves back to zero radians, from approximately time 70 to time 100

It is seen that both the clockwise and counter-clockwise rotations generate the same trend in the measurements, and therefore the filter will not be able to differentiate between the two cases. The reason behind this phenomenon is that the object is symmetrical around its rotation center. This is one noted case when the 2D-bounding box measurements cause a problematic filtered signal, but several more cases likely exist since a 2D-bounding box is a simplification of a 3D object.

### 4.2.4 Tests Including an Extra Measurement of Rotation

To compensate for this limitation in the filter, the last subcategory of tests includes an extra measurement of rotation for the filter using 2D-bounding box measurements. The purpose is to see if this extra measurement can help the filter better differentiate between problematic cases. The extra rotation measurement is the heading of the ArUco marker, but it would have been possible to also generate the measurement from the camera. The tuned parameters for this subcategory of tests can be found in Table 4.3, where $Q_{2D}$ is now lowered again to be the same as in the tests with zero angular velocity. $R_{2D}$ now includes an extra element caused by the extra measurement.

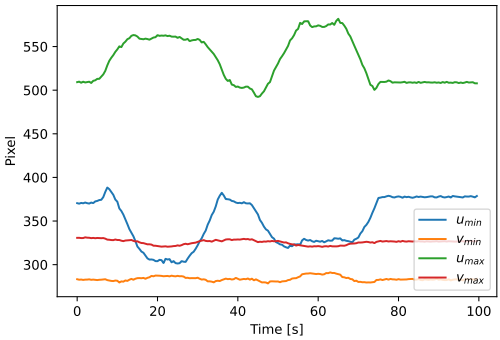



***Figure 4.10:*** *The 2D-bounding box measurements when the object moves from zero radians, goes to positive $\frac{\pi}{2}$ radians, back to zero radians, goes to negative $\frac{\pi}{2}$ radians and lastly back to zero radians again.*

***Table 4.3:*** *Tuning parameters and initial state covariance for tests in the subcategory with an extra measurement of rotation.*

| | **ArUco marker** | **2D-bounding box** |
|---|---|---|
| **Q** | $\text{diag}(0.2, 0.1)^2$ | $\text{diag}(1.5, 0.1)^2$ |
| **R** | $\text{diag}(0.4, 0.4, 2\pi/3)^2$ | $\text{diag}(200/d, 200/d, 200/d, 200/d, \pi/4)^2$ |
| **P0** | $\text{diag}(0.25, 0.25, 0.1, \pi/2, 0.1)^2$ | $\text{diag}(0.25, 0.25, 0.1, \pi/2, 0.1)^2$ |

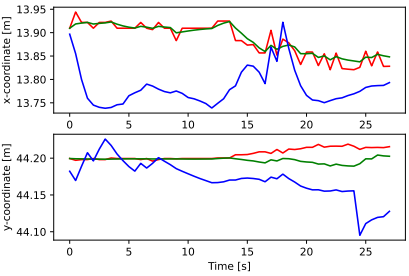
To see if this extra measurement has the desired effect, the same test as in Figure 4.9, is tested again. The results are viewed in Figure 4.11. It is seen that the filter now can differentiate between the two rotations, which is the desired outcome of this test. However, the heading estimates do not reach the desired angles. The reason for this is that no ArUco marker measurements are generated for large rotations, and hence that will be a limitation when using that measurement in both filters.

The last test includes the object moving in a curve toward the camera. It uses both non-zero linear velocity and non-zero angular velocity, and tests change both in the x-coordinate, y-coordinate, and heading. The results are shown in 4.12, and it is seen that both filters perform similarly.

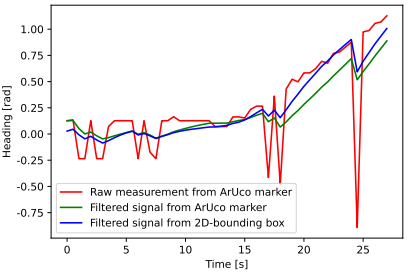
## 4.3 State Lattice Motion Planning

The filtered signal from the ArUco marker measurements, when the object moves diagonally in front of the AGV, is used to test the state lattice motion planner.

The outer bounds of the detected object, at two different samples, are viewed in Figure 4.13. Only the current state and its covariance, in each sample, are used to create the outer bound. It would however also be possible to predict the motion

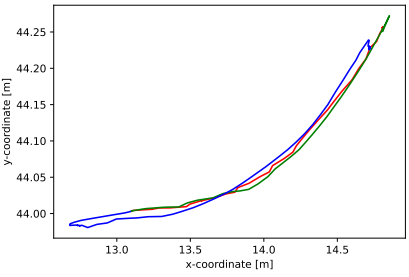



***(a)*** *A comparison of the resulting signals in x- and y-direction respectively.*

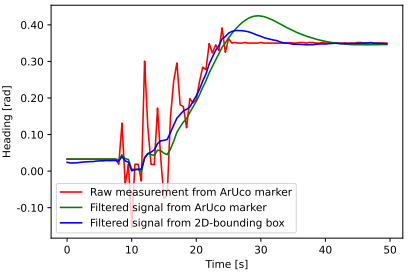



***(b)*** *A comparison of the resulting signals of heading.*

***Figure 4.11:*** *The result when the object rotates counter-clockwise, as described in Figure 4.8b. The filtered signal using 2D-bounding box measurements also uses the raw rotation measurement from the ArUco marker.*

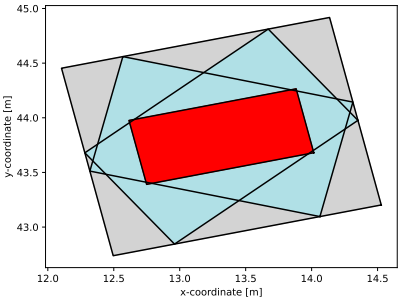



***(a)*** *A comparison of the resulting signals in x- and y-direction.*

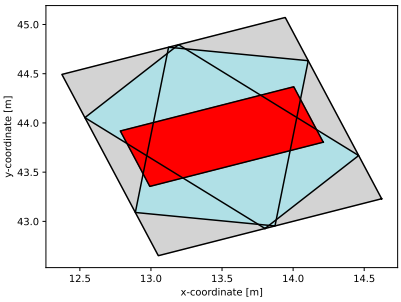



***(b)*** *A comparison of the resulting signals of heading.*

***Figure 4.12:*** *The object travels toward the AGV starting at rotation 0 and moving in a curvilinear motion. The filtered signal using 2D-bounding box measurements also uses the raw rotation measurement from the ArUco marker.*




***(a)*** *The outer bound of the object at sample 50.*




***(b)*** *The outer bound of the object at sample 1.*

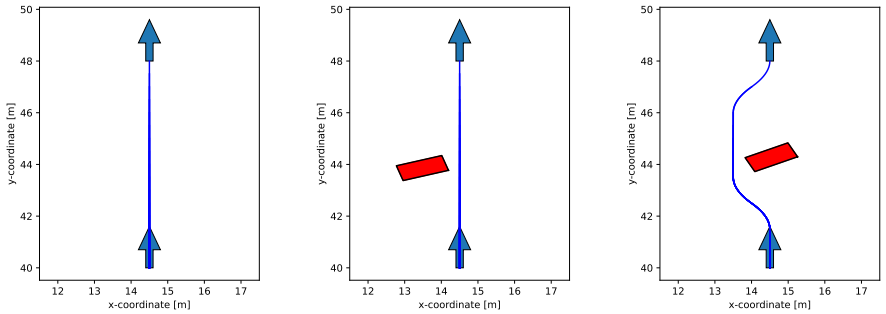
***Figure 4.13:*** *The red rectangle is an approximation of the detected object based on known dimensions. The light blue rectangles include uncertainties in the length and width of the object, as well as the rotation. The gray area is the outer bound of the object taking this uncertainty into account.*

of the object m-steps ahead in each sample and use all this knowledge to create m-number of outer bounds. The uncertainty would increase the further into the future the object is predicted, potentially causing no feasible path to be found at all if the uncertainty becomes too large.

The red obstacle, in Figure 4.13, is the detected object if no uncertainty is taken into account. The light blue rectangles show the rectangle rotated with one standard deviation, with added width and length based on the uncertainty in the x- and y-coordinate. The gray rectangle is the risky area where the object might be located. It can be seen in Figure 4.13a that the gray area fully covers the original object as well as the uncertain versions, and hence creates a valid outer bound. In Figure 4.13b a limitation with this approach is shown, where the sample has a large uncertainty in the estimated rotation. At a first glance, the outer bound covers the original object as well as the uncertain versions. However, a smaller standard deviation in the rotation would yield a larger length of the object. It is therefore observed that this approach works for small uncertainties, but not as well for larger ones.

The resulting planned paths can be seen in Figure 4.14. The planned path works as expected, and the obstacle is avoided when it obscures the path. The shown paths are the last iterations from ARA*, and are therefore optimal paths that would be the same as if A* was run instead.

Figure 4.15 shows some of the suboptimal solutions from the same test. The $\epsilon'$-suboptimal solution guarantees that the length of the found solution is no larger than $\epsilon'$ times the length of the optimal solution [35]. This is seen to be true, and thus a trade-off between computational time and path quality can be used to

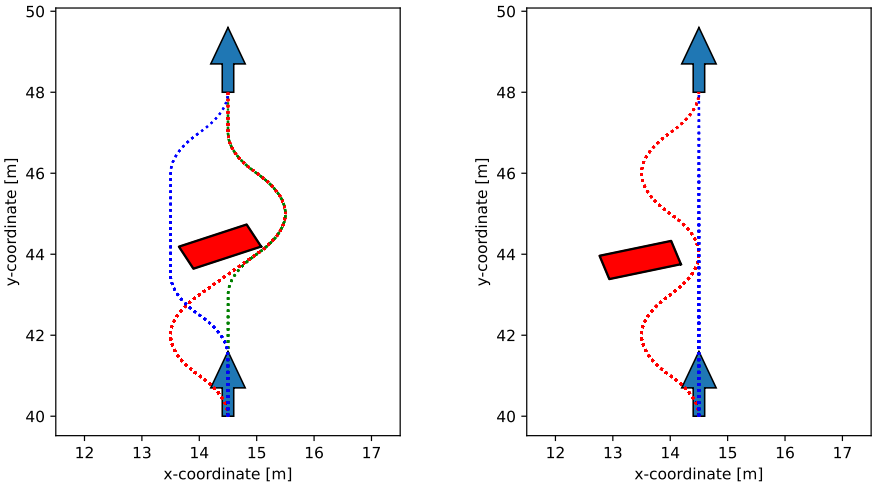



***(a)*** *No obstacle.* ***(b)*** *Obstacle appear.* ***(c)*** *Obstacle in the way.*

***Figure 4.14:*** *Results from the motion planning with ARA*, where the detected obstacle follows a diagonal line from a filtered signal generated with measurements from the ArUco marker. The bottom arrow shows where the AGV is currently located, and the upper arrow where the AGV aims to go. The detected obstacle is shown in red and the planned path in blue. (a) shows the planned path when no obstacle has yet been detected. (b) shows the planned path when the obstacle is detected. (c) shows the new planned path when the obstacle crosses the original planned path.*

decide if more iterations should be made or if the path generated is good enough.

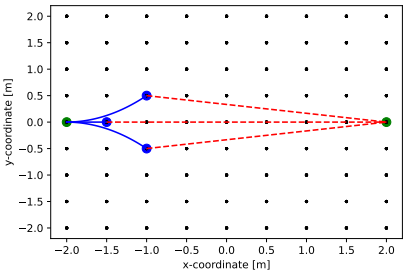
ARA* generates, in some scenarios, an unintuitive path when the heuristic is inflated. One such example is seen in Figure 4.16. Intuitively, the chosen path would be a straight line from start to goal when there are no obstacles in the world. However, since ARA* priorities the node search based on $g(s) + \epsilon h(s)$, it will instead prioritize a longer route if $h(s)$ is smaller. Hence, when $\epsilon > 1$ in this scenario, the dotted green line will be chosen even though there are no obstacles present. If more time is available, ARA* will however generate the optimal path eventually. This is also seen with the solid green line when $\epsilon$ has decreased.



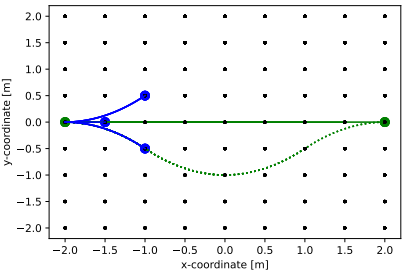

***(a)*** *The red path shows the $\epsilon' = 1.163$ suboptimal solution with length 9.30 meters and the green path shows the $\epsilon' = 1.052$ suboptimal solution with length 8.591 meters. Lastly the blue path shows the $\epsilon' = 1$ suboptimal solution with length 8.591 meters.*

***(b)*** *The red path shows the $\epsilon' = 1.147$ suboptimal solution with length 9.181 meters. The blue path shows the $\epsilon' = 1$ suboptimal solution with length 8 meters.*

***Figure 4.15:*** *Two different searches with ARA* are viewed, where some chosen suboptimal solutions are drawn with dotted lines. The bottom arrow shows where the AGV is currently located and the upper arrow where the AGV aims to go.*




***(a)*** *The red paths show the heuristics from the successors of the start node to the goal node.*




***(b)*** *The green dotted path is generated when $\epsilon > 1$ and the solid green line is generated when $\epsilon = 1$.*

***Figure 4.16:*** *An example when ARA* generates an unintuitive first path when $\epsilon > 1$. The start node and goal node are green. The successors of the start node, and the path there, are seen in blue. In (a) the length of the heuristic is shown in red. In (b) the resulting paths are shown for two scenarios of $\epsilon$.*

# 5
# Conclusion

## 5.1 Answering the Problem Statements

Three problem statements were presented in Chapter 1.4 and analyzed in this master thesis. Each research question is in the following Section addressed separately.

- Is it possible to use a single RGB camera to detect complex obstacles in form of protruding forks?

To perform vision based object detection using one RGB camera, a deep learning approach using a YOLOv5s model was chosen. After data acquisition, the model was trained with custom data to detect 2D-bounding boxes around forks. A good result was found with an average precision of 0.714, for a confidence threshold of 0.5:0.95, on the test data. To conclude, detecting complex obstacles with a single RGB camera is possible.

- Based on data from a single RGB camera, is it possible to track the object's position in the world?

Two methods were designed to track the object's position in the world. One Extended Kalman filter using raw measurements from a detected ArUco marker and the other 2D-bounding boxes from the YOLOv5s detection. The second one uses the assumption that an initial object state estimate in 3D space is known, a limitation that does not exist with the first. Both filters result in similar performance when the angular velocity is zero. The filter using 2D-bounding box measurements performs less reliably when adding a non-zero angular velocity. An identified potential reason is that measurements from a clockwise and counterclockwise rotation result in similar 2D-bounding box measurements. An extra

rotation measurement was added to better differentiate between the cases. Improved results were obtained. To conclude, tracking the object's position in the world based on a single RGB camera is possible.

- Can motion planning be designed to avoid complex dynamic obstacles based on RGB-camera data?

A state lattice motion planning approach was designed to avoid complex dynamic obstacles. To limit the computation time an anytime planner ARA* was implemented, which can find suboptimal paths in a shorter time than A*. The results show that the obstacles are sufficiently avoided. Therefore, it is concluded that motion planning based on RGB-camera data can be designed to avoid complex dynamic obstacles.

## 5.2 Future Work

This master thesis has only begun to uncover the potential problems and possibilities with using a single RGB camera for collision avoidance. Several interesting aspects that would benefit from more research and improvement exist.

Because of time limitations, the application was never implemented on the computer M03975. This would be interesting to test to see if all calculations, object detection, tracking, and motion planning, could be done during the critical time limit imposed by real-time applications or if more optimization is needed.

The object detection works well in this application, but several possible improvements exist. One potential problem is that all training, validation, and test data is gathered in the laboratory facilities at Toyota Material Handling Manufacturing Sweden. The detection would most likely not perform as well upscaled in a factory setting. Including a large amount of data in diverse environments would solve this problem.

The object tracking using 2D-bounding box measurements is more robust with an added measurement of object rotation. This measurement is, in this application, extracted from an ArUco marker. In future work, it would be interesting to see if a similar measurement can be generated differently and make the ArUco marker obsolete. Two potential ways to approach this problem would be to use deep learning or look at SIFT features in consecutive frames.

This master thesis did not focus on the initialization of the object state in 3D space, something worth developing. One way to approach this would be as proposed by Reich and Wuensche [30] with a monocular 3D object tracker. Another possible aspect is to include an additional RGB camera to see if this is solvable using stereo-vision.

The state lattice motion planner works well in this application but could be improved. ARA* is currently the implemented graph search algorithm, but other graph search algorithms could perform even better. D* Lite [36] is a graph search algorithm that works well in dynamic environments where the edge costs can

change while the robot moves toward the goal. Another suitable graph search algorithm that has the potential to work well for this application is the Anytime Dynamic A* algorithm [37].

# Bibliography


[1] Zesen Liu, Chuanhong Guo, Sheng Bi, Kezheng Yu, Guojun Peng, and Yuyang Yue. A robot obstacle avoidance approach with lidar and rgb camera data combined. *2021 IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 140–145, 2021.

[2] Fengyuan Jia, Zhaosheng Tao, and Fusong Wang. Pallet detection based on halcon and alexnet network for autonomous forklifts. *2021 International Conference on Internet, Education and Information Technology (IEIT)*, pages 86–89, 2021.

[3] Partha Narayan Chowdhury, Tonmoy Chandra Ray, and Jia Uddin. A vehicle detection technique for traffic management using image processing. *2018 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2)*, pages 1–4, 2018.

[4] Tri Nguyen and Myungsik Yoo. Fusing lidar sensor and rgb camera for object detection in autonomous vehicle with fuzzy logic approach. *2021 International Conference on Information Networking (ICOIN)*, pages 788–791, 2021.

[5] Jan Kallwies, Bianca Forkel, and Hans-Joachim Wuensche. Determining and improving the localization accuracy of apriltag detection. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8288–8294, 2020.

[6] Joel Pazhayampallil. Free space detection with deep nets for autonomous driving. 2015.

[7] Jeff Michels, Saxena Ashutosh, and Ng Andreq Y. High speed obstacle avoidance using monocular vision and reinforcement learning. *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

[8] Armin Masoumian, David G.F. Marei, Saddam Abdulwahab, Julián Cristiano, Domenec Puig, and Hatem A. Rashwan. Absolute distance prediction based on deep learning object detection and monocular depth estima-

tion models. *Artificial Intelligence Research and Development*, Oct 2021. ISSN 1879-8314. doi: 10.3233/faia210151. URL http://dx.doi.org/10.3233/FAIA210151.

[9] Nurul Fathanah Mustamin. Relative distance measurement between moving vehicles for manless driving. *2017 International Seminar on Application for Technology of Information and Communication (iSemantic)*, pages 1–4, 2017.

[10] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of field robotics. 26(3)*, pages 308–333, 2009.

[11] Olov Andersson, Oskar Ljungqvist, Mattias Tiger, Daniel Axehill, and Fredrik Heintz. Receding-horizon lattice-based motion planning with dynamic obstacle avoidance. *2018 IEEE Conference on Decision and Control (CDC)*, pages 4467–4474, 2018.

[12] MathWorks. What is camera calibration? URL https://se.mathworks.com/help/vision/ug/camera-calibration.html.

[13] IBM Cloud Education. Ai vs. machine learning vs. deep learning vs. neural networks: What's the difference?, . URL https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

[14] IBM Cloud Education. What is deep learning?, . URL https://www.ibm.com/cloud/learn/deep-learning.

[15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[16] Jeremy Jordan. Evaluating a machine learning model., Aug 2018. URL https://www.jeremyjordan.me/evaluating-a-machine-learning-model/.

[17] Jonathan Hui. Real-time object detection with yolo, yolov2 and now yolov3, Aug 2019. URL https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

[18] Qisong Song, Shaobo Li, Qiang Bai, Jing Yang, Xingxing Zhang, Zhiang Li, and Zhongjing Duan. Object detection method for grasping robot based on improved yolov5. *Micromachines*, 12(11), 2021. ISSN 2072-666X. URL https://www.mdpi.com/2072-666X/12/11/1273.

[19] Kiprono Elijah Koech. On object detection metrics with worked example, Mar 2022. URL https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e.

[20] Fredrik Gustafsson, Lennart Ljung, and Mille Millnert. *Signal processing*. Studentlitteratur, Lund, 1:3 edition, 2010.

[21] Gustaf Hendeby. *Performance and Implementation Aspects of Nonlinear Filtering*. PhD thesis, 03 2008.

[22] Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. Improved optimization of motion primitives for motion planning in state lattices. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2307–2314, 2019. doi: 10.1109/IVS.2019.8813872.

[23] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[24] IFM. Python api reference - o3r documentation, 2022.

[25] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2014.01.005. URL https://www.sciencedirect.com/science/article/pii/S0031320314000235.

[26] OpenCV. Open source computer vision library: Aruco marker detection, 2015.

[27] Glenn Jocher. Train custom data · ultralytics/yolov5 wiki, 2022. URL https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data.

[28] Ultralytics. Tips for best training results - yolov5 documentation, 2021. URL https://docs.ultralytics.com/tutorials/training-tips-best-results/.

[29] Pragati Baheti. Train, validation, and test sets: How to split your machine learning data, 2022. URL https://www.v7labs.com/blog/train-validation-test-set.

[30] Andreas Reich and Hans-Joachim Wuensche. Monocular 3d multi-object tracking with an ekf approach for long-term stable tracks. In *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, pages 1–7, 2021. doi: 10.23919/FUSION49465.2021.9626850.

[31] Michael Roth, Gustaf Hendeby, and Fredrik Gustafsson. Ekf/ukf maneuvering target tracking using coordinated turn models with polar/cartesian velocity. In *17th International Conference on Information Fusion (FUSION)*, pages 1–8, 2014.

[32] Ricardo Carona, A. Pedro Aguiar, and José Gaspar. Control of unicycle type robots tracking, path following and point stabilization. 11 2008.

[33] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[34] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, May 2006. ISSN 0025-5610. doi: 10.1007/s10107-004-0559-y. Copyright: Copyright 2008 Elsevier B.V., All rights reserved.

[35] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 767 – 774, December 2003.

[36] Sven Koenig and Maxim Likhachev. D*lite. In *Eighteenth National Conference on Artificial Intelligence*, page 476–483, USA, 2002. American Association for Artificial Intelligence. ISBN 0262511290.

[37] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'05, page 262–271. AAAI Press, 2005. ISBN 1577352203.