Multi-frame multi-target tracking using radar and AIS

Erik Liland

Report submitted to

Norwegian University of Science and Technology

Department of Engineering Cybernetics

O.S. Bragstads Plass 2D

Elektroblokk D, Gløshaugen,

7034 Trondheim, Norway

December 2016

Preface

The work presented in this thesis

Erik Liland
Trondheim, 05.06.2017

Abstract

The answer is 42

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

An Autonomous Surface Vessel (ASV) is an unmanned vessel which can for instance be used to transport cargo and people, as well as for surveillance and other tasks. To avoid collisions and dangerous situations with other vessels, the route planner needs a real time image of its surroundings in addition to map data. In the maritime environment, this is primarily done by radars mounted on the ship itself. All commercial maritime vessels and vessels over 15 meter are also required to have an Automatic Identification System (AIS) installed, and with AIS being more and more common in leisure vessels, its usefulness in anti collision perspective increases.

1.2 Problem description

The proposed title for this project, and the following master-thesis, was "Multi-target tracking using radar and AIS". The key idea behind this formulation is that radar and the maritime AIS have different strengths and weaknesses, and if utilized properly, the strengths of both systems can be exploited, while the weaknesses can be reduced. For this specialization project, it was decided to focus fully on the gold standard within multi-target tracking, namely Multi Hypothesis Tracking (MHT) . To ensure the tracking system would be sufficiently understood, correctly implemented and thoroughly tested, the AIS integration was postponed to the master thesis. The following tasks was proposed for this project:

- Write a survey on multi-target and multi-sensor tracking methods
- Implement a multi-target tracking method
- Describe the method and summarize the findings in a report

The goals for this project is to implement a Track Oriented Multi Hypothesis

Tracker (TOMHT) algorithm in Python, test it with simulated data, analyse the performance for different conditions and discuss methods for improvement.

1.3 Outline of report

In section **??**, different methods for target tracking and data association are presented. Here, the focus is to highlight the key difference between the most common tracking methods with particular attention on the properties these lead to. In section **??**, a TOMHT is thoroughly explained, and in section **??** the integer optimization problem that arises in section **??** is elaborated. The implementation of the outlined algorithm is presented on a structural and pseudo-code level in section **??**. At last, the simulation and results are presented in section **??** and discussed in section **??**.

2 Survey of multi-target tracking methods

The aim of this section is to give the reader a brief overview of tracking as a problem, and a feeling for the most popular methods. Their assumptions and strong and weak properties seen from a two dimensional maritime anti collision perspective are also presented.

2.1 Tracking

Tracking of an object (target) is the process of estimating its state (i.e. position and velocity) based on discrete measurements from an observation system. An observation system can be a radar, sonar or any other sensor that, passively or actively, detects objects within an area or volume. Any observation system will be prone to noise, both in form of internal- and external noise from the environment. This noise will cause false measurements that the tracking system must take care of. These false measurements are often referred to as clutter.

2.2 Tracking system

A tracking system can be interpreted as either the complete system from the signal processing level to the finished tracks, or as I define it in this text: *A system that process consecutive measurements from an observation system and collects measurements from the same target into tracks or initiate new tracks.* A track is a subset of all the measurements from the observation system that is believed to originate from the same target. The challenge of knowing which measurement originates from which real target, is the core at any tracking system. This association problem is non-trivial even under ideal conditions, and the addition of spurious measurements and missed targets only increases the complexity.

There has been developed a large variety of methods to solve this association problem, and most of them have several sub-variants. In the following subsections, some of the most common and popular methods will be presented.

2.3 Nearest Neighbour Filter

The Nearest Neighbour Filter (NNF) is the simplest approach in tracking [?], where the closest neighbour is always selected as the consecutive measurement in the track. Normally the distance is measured as the Euclidean distance (??).

$$\left[(\mathbf{z}_k - \mathbf{z}_{k-1}) \cdot (\mathbf{z}_k - \mathbf{z}_{k-1})\right]^{1/2} \tag{1}$$

This approach is very vulnerable to clutter and dense target scenarios, but can be somewhat improved by predicting an a-priori state through a Kalman Filter and selecting the nearest neighbour to the prediction. This extension is sometimes refereed to as Nearest Neighbour Standard Filter (NNSF) [?] and also differs from the original in that it uses the Mahalanobis distance (??) which is a measure of the distance from a measurement to a distribution.

$$\left[\mathbf{z}_k - \hat{\mathbf{z}}_k\right]^T \boldsymbol{S}(k+1)^{-1} \left[\mathbf{z}_k - \hat{\mathbf{z}}_k\right] \tag{2}$$

Under the standard assumption that each target can, at maximum, generate one measurement, the NNF and NNSF are both *single-target* methods since they may assign the same measurement to more than one track. They can, however, be expanded to multi-target variants by formulating the problem as a global least squares integer optimization problem, often called Global Nearest Neighbour Filter (GNNF). With this extension, the NNSF is almost becoming a zero-scan multi hypothesis tracker, in the sense that it seeks to select the optimal combination of mutual exclusive measurement-to-track associations while only looking at the most recent scan. NNF, NNSF and GNNF can be viewed as non-probabilistic models, as they do not assume specific models for noise, clutter, false alarm rate or similar.

2.4 Probabilistic Data Association Filter

Probabilistic Data Association Filter (PDAF) is a *single-target* Bayesian association filter which is analysing single scan probabilities for measurements. The target is assumed initialized and modelled by (??). At each scan, the algorithm

calculates the association probabilities for all the measurements inside a validation gate, with the assumption that maximum one of the measurements inside the validation gate is the true target. This leads to the state update equation (??), where the current state is updated with a combined innovation. The flowchart of the PDAF algorithm is shown in Figure ??, taken from [?].

$$\begin{aligned} \boldsymbol{\nu}_k &\triangleq \sum_{i=1}^{m_k} \beta_k^i \tilde{\mathbf{y}}_k^i \\ \hat{\mathbf{x}}_k &= \bar{\mathbf{x}}_k + \boldsymbol{K}_k \boldsymbol{\nu}_k \end{aligned} \tag{3}$$

$\beta_k^i =$ the probability of measurement z^i beeing the correct one

$\tilde{\mathbf{y}}_k^i = \mathbf{z}_k^i - \hat{\mathbf{z}}_k^i$ the measurement innovation for measurement i at time step k

$\boldsymbol{K}_k =$ the Kalman Gain for the k-th time step

PDAF is computationally modest (approximate 50% more computationally demanding than a Kalman Filter [?] p.163) and have good results in an environment with up to about 5 false measurement in a 4σ validation region [?]. PDAF does not include track initialization and assumes that at most one measurement can originate from an actual target. It also assumes that clutter is uniformly distributed in the measurement space and that the targets history is approximated by a Gaussian with a calculated mean and covariance.

Figure 1: PDAF flowchart

PDAF can be used in multi-target scenarios, but only as multiple copies of the single-target filter [?]. It can suffer from track coalescence, which is

a phenomenon that merges two tracks into one. This coalescence occurs when two targets have similar paths, and the resulting tracks will be an "average" of the two actual tracks. Some work has been done to overcome this coalescence [?].

2.5 Joint Probabilistic Data Association Filter

The Joint Probabilistic Data Association Filter (JPDAF) is a *multi-target* extension of the PDAF in which joint posteriori association probabilities are calculated for every target at each scan. Both PDAF and JPDAF use the same weighted sum (**??**), the key difference is the way the weight β_i^j is calculated. Whereas PDAF treats all but one measurement inside its validation region as clutter, JPDAF treats the interacting targets (one cluster) as connected. The connected β_i^js are computed jointly across the cluster with a given set of active targets inside the cluster. The probability [?] of a measurement i belonging to a target j is (**??**). Since JPDAF is using all the measurements inside the gate for each track, one measurement can be used to update more than one track if it is within more that one gate.

$$\begin{aligned} \beta_i^j &= \sum_{\chi} P\{\chi|Y^k\}\hat{\omega}_{ij}(\chi) \\ \beta_0^j &= 1 - \sum_{i=1}^{m_k} \beta_i^j \end{aligned} \tag{4}$$

$$P\{\chi|Y^k\} = \frac{C^\phi}{c} \prod_{i:\tau_i=1} \frac{\exp[-\frac{1}{2}(\tilde{\mathbf{y}}_\mathbf{i}^\mathbf{j})^T S_{j_i}^{-1}(\tilde{\mathbf{y}}_\mathbf{i}^\mathbf{j})]}{(2\pi)^{M/2}|S_{j_i}|^{1/2}} \prod_{j:\delta_j=1} P_D^j \prod_{j:\delta_j=0} (1 - P_D^j) \tag{5}$$

$$\beta_i^j = \text{the probability that measurement } i \text{ belongs to target } j$$

$$\beta_0^j = \text{the probability that no measurement belongs to target } j$$

$$\chi = \bigcap_{i=1}^{m_k} \chi_{ij_i} \quad \text{All feasible events}$$

$$Y^k = \text{all candidate measurements up to and included time } k$$

$$\hat{\omega}_{ij} = \begin{cases} 1, \text{if } \chi_{ij} \text{ occurs} \\ 0, \text{otherwise} \end{cases}$$

$$m_k = \text{number of measurements in scan } k$$

Since the JPDAF is calculating joint probabilities for all the combinations of measurement associations in the cluster, the computation demand is growing exponentially with the numbers of tracks and measurements in the cluster. A real time implementation of the JPDAF has been developed and patented by QinetiQ [**?**], and described in Horridge and Maskell [**?**].

JPDAF suffers from the same coalescence problem as PDAF, and seen from an anti-collision perspective, track coalescence is highly undesirable. An improvement to the JPDAF considering track coalescence has been proposed by Blom and Bloem [**?**].

2.6 Multi Hypothesis Tracker

MHT is a decision logic which generates and maintains alternative hypotheses when new measurements are received and within the gate. By making several possible hypotheses, the decision on which measurement to choose can be propagated into the future when more information is available. Each hypothesis is given a score or probability as a reflection of how well the measurement fits the model, which are accumulated to evaluate the combinations of consecutive measurements.

In contrast to the PDAF and JPDAF methods who suffers from track coalescence, MHT methods split when in doubt. The idea of using multiple hypotheses was first introduced by Singer. et al. [**?**], but the first complete algo-

rithm was presented by Reid [?], where a Hypothesis Oriented Multi Hypothesis Tracker (HOMHT) was developed. Following this, a TOMHT was proposed by Kurien [?] and the score function for MHT was later deduced and discussed by Bar-Shalom [?] since no explicit track-score function were given in [?]. MHT is, in the same way as PDAF/JPDAF, developed under two main assumptions; that at most one measurement can originate from each target in each scan, and that a target does not necessarily show on every scan, or in other words, Probability of detection (P_D) less than 1.

If the Maximum A-Posteriori Probability (MAP) hypothesis from a MHT is extracted, apparent inconsistency in the output to the user can appear whenever a new set of measurements are received since all association N time-steps back are reconsidered. If this is unwanted, an alternative representation is to output the N-best tracks and display them to the user in a way that visualises the probability. A third option proposed by Blackman [?], is to output an averaged state estimate of the tracks along with the covariance of the average that reflects the uncertainty.

The MHT approach to tracking and data association was for a long time dismissed by many because of its computationally large cost. The dramatic increase in computational capability from the 1980s to the late 2010s have, however, lead to a new spring for MHT, with an increasing interest for use in tracking system. In 2004 Blackman stated that "Multiple hypothesis tracking is generally accepted as the preferred method for solving the data association problem in modern multiple target tracking system" [?]. Already in 2001 did Blackman publish a demonstration that MHT is capable of real-time demands [?].

There are two main approaches to MHT, HOMHT and TOMHT.

2.6.1 Hypothesis Oriented MHT

HOMHT, also known as Measurement Oriented Multi Hypothesis Tracker (MOMHT), is a tracking approach where direct probabilities of global joint measurement-to-target association hypothesis are calculated. The algorithm initiates tracks and handles missing measurements, has a recursive nature and allows for clustering

for quicker computation.

When a new set of measurements is received, Reids method [**?**] defines a set of hypotheses, each containing a complete set of associations of the existing tracks and new measurements. By defining the hypotheses in this way, they become compatible in the sense that one, and only one, is the true hypothesis. When the next set of measurements arrives, each of the current hypotheses are expanded with all measurement-to-track assignments for the new measurements. This way, the hypotheses keeps their compatibility.

When evaluating the alternative hypotheses, each one is assigned a probability. This probability takes into account the false-alarm statistics of the measurement system, the expected density of targets and clutter and the accuracy of the target estimates. The probability of each data association hypothesis was by Reid [**?**] expressed as (**??**).

$$P_i^k = \frac{1}{c} P_D^{N_{DT}} (1 - P_D)^{(N_{TGT} - N_{DT})} \beta_{FT}^{N_{FT}} \beta_{NT}^{N_{NT}} \left[\prod_{i=1}^{N_{DT}} \mathcal{N}(\mathbf{Z_m} - \boldsymbol{H}\bar{\mathbf{x}}, \boldsymbol{R}) \right] P_g^{k-1} \tag{6}$$

where

$$P_i^k = \text{the probability of hypothesis } \Omega_i^k \text{ given measurements up through time } k$$
$$P_g^{k-1} = \text{the probability hypothesis } \Omega_g^{k-1}$$
$$P_D = \text{the probability of detection}$$
$$\beta_{FT} = \text{the density of targets}$$
$$\beta_{NT} = \text{the density of previously unknown targets that have been detected}$$
$$N_{DT} = \text{number of designated target}$$
$$N_{FT} = \text{the number of false targets}$$
$$N_{NT} = \text{the number of new targets}$$
$$N_{TGT} = \text{the number of targets}$$
$$\mathbf{z_i} = \text{the i-th measurement in the current scan}$$
$$\boldsymbol{H} = \text{the state observation matrix}$$
$$\boldsymbol{R} = \text{the measurement covariance}$$
$$\mathcal{N}(\mu, \sigma^2) = \text{the Normal distribution PDF with mean } \mu \text{ and variance } \sigma^2$$

As with all MHT algorithms, it is crucial to prune the hypothesis tree to avoid an infinite memory and computational cost. Reid [**?**], utilized several pruning techniques, were the most important one is likely the procedure to remove all hypotheses with probability below a threshold. A second technique that were proposed in [**?**] is to merge hypotheses with the last N data scans in common, where the importance of preserving earlier rather than later hypotheses is also argued.

The initialization of tracks is done through the probability of each hypothesis, where a hypothesis would go from a tentative track to a confirmed track when the probability of that hypothesis exceeds a threshold, i.e. 99%. By treating all new measurement as tentative targets, and calculate their joint probability using information such as density of false targets, density of new targets and probability of detection and thresholding this, the algorithm has an advanced build-in initialization mechanism.

2.6.2 Track Oriented MHT

TOMHT is a "bottom-up" approach where the tracks are assumed initialized, and for each scan the tracks splits whenever there are more than one feasible measurement in the validation regions (in addition to the no-measurement hypotheses). This give rise to a track tree, as shown in Figure **??**, where each initialized track has a root node and multiple branches, were each level represents a scan. A hypothesis is now a set of compatible tracks, with minimum and maximum one track from each track tree, were a hypothesis' score is the sum of its tracks score. Tracks are compatible if they do not share any measurements. The optimal hypothesis, in MAP sense, is the hypothesis with the highest score.

In the special case were each track tree does not share any measurements (single target situation), the optimal track is the leaf node with the highest score, which can be found by atraversal of the tree in linear time [**?**]. However, when a measurement is assigned to two or more targets, the problem of selecting the optimal combination of tracks becomes a multi dimensional assignment problem, since the best hypothesis from each track tree might be mutual exclusive.

New track hypotheses are generated from the filtered estimate from a Kalman Filter, and a score is calculated for instance using (20) from [**?**]. Both [**?**] and [**?**] suggests that modern tracking systems could improve performance in certain scenarios by using an Interacting Multiple Model (IMM) approach in stead of a single Kalman Filter. However, this would inevitably increase the computatinal complexity. Following the addition of new track hypotheses, the tracks are divided into clusters in which tracks with common measurements are grouped. The clusters can then be analysed as standalone global problems to find the best combination of possibly mutual exclusive measurement associations. Linear Programming (LP) and Integer Linear Programming (ILP) based methods can be used to find the best combinations of newly created track hypotheses in accordance to the assumption that a measurement can only be assigned to one target and that one target can maximally create one measurement.

To limit the size of the track hypothesis tree, various pruning schemes have been developed. The maybe single most important pruning technique is the N-

scan pruning [?], also known as N-scan sliding window. This is a simple technique for limiting the size of the track tree and the following number of leaf nodes. The pruning is done by selecting the node that is N levels higher that the current best leaf node as the new root node.

One of the largest drawbacks with the TOMHT approach is the difficulty of track initialization. This is because TOMHT normally only considers measurements that are within any already initialized targets' gate, and automatically discards the unused measurements in order to limit the computation time. An alternative could be to treat all unused measurements as new targets. This would, however, increase the computational demand and runtime considerably. The most common approach to add track initialization to TOMHT is to run a separate initialization algorithm in parallel with the tracking loop. This initialization method can be chosen freely, and some popular alternatives are; a dedicated HOMHT running on all or the unused measurements from the main loop, a N-of-M method based on either nearest neighbour or PDAF/JPDAF or simply a manual initialization by the user. The options for this aiding module are endless, and the specific situation will influence the choice of method.

3 Algorithm walk-through

3.1 Flowchart

Figure **??** shows a flowchart of the TOMHT algorithm presented in this section. An important difference between this approach compared to Reid's original HOMHT [**?**], is the need for external initialization of targets, as mentioned in section **??**.

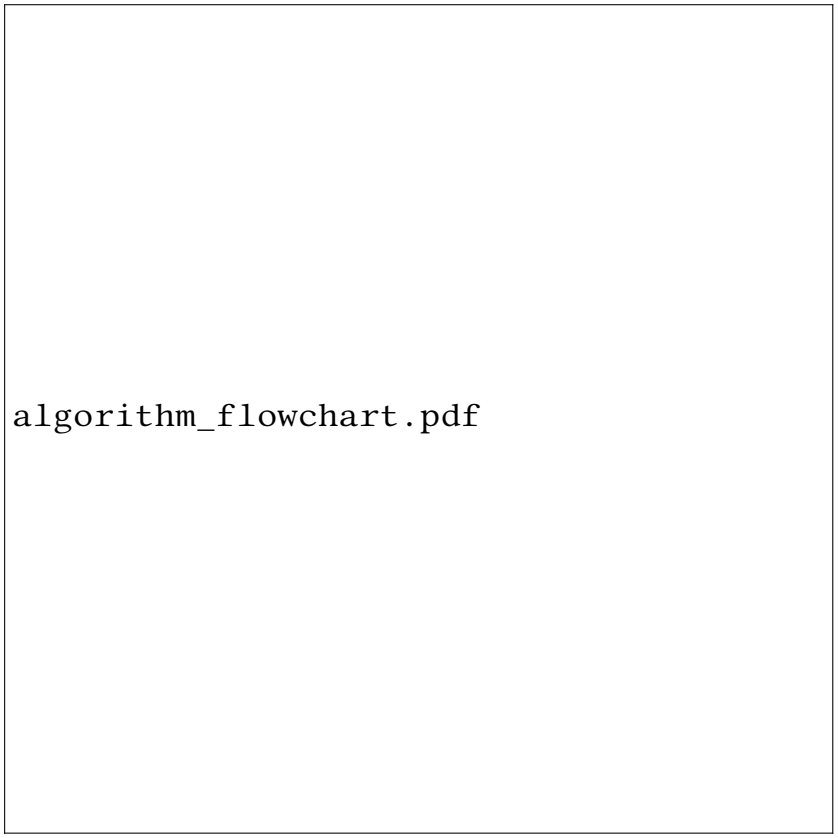

Figure 2: Algorithm flowchart

3.2 State estimation

When a new set of measurements arrives, it is desirable to predict where the target might be before looking for matching measurements. This can be done though a model of the targets dynamics and a state estimator. For the purpose of

tracking ships in a local frame (Cartesian plane), we compose a state vector with four states

$$\mathbf{x} = \begin{bmatrix} x & y & \dot{x} & \dot{y} \end{bmatrix}^T \tag{7}$$

where the change of velocity (acceleration) is assumed to be white Gaussian noise. For a linear system with white and uncorrelated system- and measurement noise, the Kalman Filter is an optimal estimator with (??) as time evolution model.

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{\Phi}\mathbf{x}(k) + \mathbf{\Gamma}\mathbf{w} \\ \mathbf{z}(k) &= \boldsymbol{H}\mathbf{x}(k) + \mathbf{v} \end{aligned} \tag{8}$$

$$\begin{aligned} \mathbf{\Phi} &= \text{the state transition matrix} \\ \mathbf{\Gamma} &= \text{the disturbance matrix} \\ \mathbf{w} &= \text{the process noise} \\ \boldsymbol{H} &= \text{the state observation matrix} \\ \mathbf{v} &= \text{the observation noise} \end{aligned} \tag{9}$$

The procedure of predicting the target state at the next time step is according to the time update equations of the Kalman Filter (??),

$$\begin{aligned} \bar{\mathbf{x}}(k+1) &= \mathbf{\Phi}\hat{\mathbf{x}}(k) \\ \bar{\boldsymbol{P}}(k+1) &= \mathbf{\Phi}\hat{\boldsymbol{P}}(k)\mathbf{\Phi}^T + \boldsymbol{Q} \end{aligned} \tag{10}$$

where $\boldsymbol{Q}$ is the process noise covariance matrix. The constant velocity model used in this work have the following parameters,

$$\mathbf{\Phi} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{\Gamma} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{Q} = \sigma_v^2 \begin{bmatrix} \frac{T^3}{3} & 0 & \frac{T^2}{2} & 0 \\ 0 & \frac{T^3}{3} & 0 & \frac{T^2}{2} \\ \frac{T^2}{2} & 0 & T & 0 \\ 0 & \frac{T^2}{2} & 0 & T \end{bmatrix} \quad \boldsymbol{R} = \sigma_r^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where T is the time between the current and the previous measurement, σ_v^2 is the system velocity variance and σ_r^2 is the measurement variance. This model is very common due to its simplicity, and is used in among others [?], [?] and [?].

3.3 Gating

To avoid calculating the likelihood for all possible combinations of targets and measurements, some sort of selection criteria is needed when creating new hypotheses. One way of doing these selections is to select all measurements that are within certain confidence regions or gates (**??**), as illustrated in Figure **??**.

Figure 3: Validation region

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{H}\bar{\boldsymbol{P}}\boldsymbol{H}^T + \boldsymbol{R} \\ (\mathbf{Z_m} - \boldsymbol{H}\bar{\mathbf{x}})^T \boldsymbol{B}^{-1} (\mathbf{Z_m} - \boldsymbol{H}\bar{\mathbf{x}}) &\leq \eta^2 \end{aligned} \tag{11}$$

This is a very common approach, and is used by most tracking systems at some stage in the processing of new measurements. In (**??**), η is the inverse Cumulative Distribution Function (CDF) of the chi-square distribution for a given confidence with degrees of freedom equal to the sensor system. For each measurement inside the region, a filtered state and covariance is calculated using the measurement update equations in the Kalman Filter (**??**), followed by the creation of a new track hypothesis.

$$\begin{aligned} \tilde{\mathbf{y}} &= \mathbf{z} - \boldsymbol{H}\bar{\mathbf{x}} \\ \boldsymbol{S} &= \boldsymbol{H}\bar{\boldsymbol{P}}\boldsymbol{H}^T + \boldsymbol{R} \\ \boldsymbol{K} &= \bar{\boldsymbol{P}}\boldsymbol{H}^T\boldsymbol{S}^{-1} \\ \hat{\mathbf{x}}(k) &= \bar{\mathbf{x}} + \boldsymbol{K}\tilde{\mathbf{y}} \\ \hat{\boldsymbol{P}}(k) &= (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\,\bar{\boldsymbol{P}} \end{aligned} \tag{12}$$

The χ^2 value for selected confidence values are listed in Table **??**, where $95\% - 99\%$ is a common interval for gate size.

Confidence	70%	80%	90%	95%	97.5%	99%	99.5%
η^2	2.41	3.22	4.61	5.99	7.38	9.21	10.60

Table 1: Inverse χ^2 CDF for two degrees of freedom

3.4 Scoring

Each track hypothesis is scored according to [**?**]:

$$\begin{aligned} \text{NLLR}_{i,j}(k) &= \frac{1}{2}\left[\tilde{y}_k^j(i)^T S_k^{j\,-1}\tilde{y}_k^j(i)\right] + \ln\frac{\lambda_{ex}|2\pi S_k^j|^{1/2}}{P_D} \\ \tilde{y}_k^j(i) &= z_k^i - \hat{z}_{k|k-1}^j \end{aligned} \tag{13}$$

The cumulative NLLR is then

$$\text{cNLLR}_k^j \triangleq \sum_{l=0}^{k} NLLR_{i,j}(l) \tag{14}$$

3.5 Clustering

Since the global problem of finding the optimal selection of hypotheses is growing exponentially with the number of hypotheses, it is computationally beneficial to split the problem into smaller problems. This can only be done to targets that does not share any measurements, since their track hypotheses can be mutual exclusive. The clustering can be done efficiently through breath-first-search or depth-first-search on a graph made from the hypothesis tree.

By constructing a 0-1 adjacency matrix describing the connection between all the nodes in the track forest, the clustering problem is equivalent to the *connected components* problem in graph theory [**?**].

3.6 Association

When the targets are divided into independent clusters, each of them can be treated as a global problem where we want to minimize the cost or maximize the score of the selected track hypotheses (leaf nodes). The selected track hypotheses

must also fulfil the constraints, that each measurement can only be a part of one track, and that minimum and maximum one track hypothesis can be selected from each target. Since only binary values, selected or not selected, is desired for selection of hypotheses, the problem becomes an ILP. A solution to this problem is explained in section **??**.

3.7 N-scan pruning

To keep the computational cost within reasonable limits, it is necessary to limit the amount of time steps backwards in time that the algorithm computes. This is done by removing all branches but the active track hypothesis at the current root node, and assign the one remaining node as new root node. This procedure is graphically explained in Figure **??**, where a solid square frame indicates the current root node, and a dotted square frame indicates the new root node. The bold arrows in the figure represents the active track.

A similar, and in scenarios with low P_D perhaps better strategy, is N-observation pruning proposed by Blackman [**?**]. The idea is to prune N *measurements* back in time instead of N *scans*. This approach will lead to larger and more costly track trees for tracks that have low probability of detection. Therefore, in any practical implementation, a N-observation pruning scheme will be followed by a traditional M-scan pruning, where M is larger than N. The combination of these two methods can be seen as a form of adaptive pruning where tracks with high probability of detection can manage with fewer history steps than tracks with low probability of detection.

Figure 4: Pruned track hypothesis tree

Pruned-tree.pdf

4 Linear programming

The aim of this section is to elaborate the use of LP to solve the data association problem in MHT that arises when there are multiple, possibly mutual exclusive, possibilities of measurement arrangements within the existing set of tracks. As with any optimization problem, an objective function indicating how good or bad a given assignment is, and a set of constraints that limits the solution to physical limits and our assumptions are needed.

4.1 Problem formulation

Multiple optimization formulations of the association problem for multi-target tracking have been proposed through the history. The first was Morefield [?] proposing a 0-1 ILP, later an ILP scheme with LP relaxation and Greedy Rounding Procedure (GRP) as solver was proposed by Storms and Spieksma [?], and a framework for both association and removal of competing tracks were proposed by Coraluppi [?].

All the three publications have essentially the same objective functions, though some use minimize and other use maximize in their formulation. The essence of them all is (??), where $\mathbf{c}$ is a vector of costs (minimize) or scores (maximize) and $\boldsymbol{\tau}$ is a selection vector, where each row in $\mathbf{c}$ and $\boldsymbol{\tau}$ represents one branch in the track hypothesis tree.

$$\min_{\boldsymbol{\tau}} \quad \mathbf{c}^T \boldsymbol{\tau} \tag{15}$$

Further, the constraints that shall ensure that each measurement is not assigned to more than one track are generally formulated as (??) or (??) , where $\boldsymbol{A}$ is a binary matrix whose rows are branches in a track hypothesis tree and $\mathbf{b}$ is a vector with ones.

$$\begin{aligned} &\boldsymbol{A}\boldsymbol{\tau} = \mathbf{b} \\ &\boldsymbol{\tau} \in \{0,1\}^M \end{aligned} \tag{16}$$

$$\begin{aligned} &\boldsymbol{A}\boldsymbol{\tau} \leq \mathbf{b} \\ &\boldsymbol{\tau} \in \{0,1\}^M \end{aligned} \tag{17}$$

The difference between (??) and (??) originates in some fundamental assumptions in different MHT versions. When used on a HOMHT tree, (??) includes track initialization since the equality demands that all measurements must be assigned. On the other hand, both (??) and (??) requires a-priori knowledge about the number of tracks.

In this work, the problem is formulated slightly different, in that there are two sets of constraints (??), one equality and one inequality. The inequality constraints $\boldsymbol{A_1}\boldsymbol{\tau} \leq \mathbf{b_1}$ ensures that each measurement are maximum (but not minimum) used one time. The equality constraints $\boldsymbol{A_2}\boldsymbol{\tau} = \mathbf{b_2}$ ensures that minimum and maximum one track from each track tree is selected. The complete ILP formulation becomes (??), where $\boldsymbol{\tau}$ is a binary vector with dimension equal the number of leaf nodes in the track forest.

$$\begin{aligned} \max_{\boldsymbol{\tau}} \quad & \mathbf{c}^T\boldsymbol{\tau} \\ \text{s.t.} \quad & \boldsymbol{A_1}\boldsymbol{\tau} \leq \mathbf{b_1} \\ & \boldsymbol{A_2}\boldsymbol{\tau} = \mathbf{b_2} \\ & \boldsymbol{\tau} \in \{0,1\}^M \end{aligned} \tag{18}$$

$\boldsymbol{A_1}$ is a $N_1 \times M$ binary matrix with N_1 real measurements and M track hypotheses (all leaf nodes), where $\boldsymbol{A_1}(l,i) = 1$ if hypothesis l are utilizing measurement i, 0 otherwise. The measurements and hypothesis are indexed by the order they are visited by Depth First Search (DFS). $\boldsymbol{A_2}$ is an $N_2 \times M$ binary matrix where N_2 is the number of targets in the cluster and $\boldsymbol{A_2}(l,j) = 1$ if hypothesis l belongs to target j. $\mathbf{b_1}$ is a N_1 long vector with ones and $\mathbf{b_2}$ is a N_2 long vector with ones. $\mathbf{c}$ is a M long vector with a measure of the goodness of the track hypotheses. For example, in Figure ?? at time step 2, the $\boldsymbol{A}$ matrices and $\mathbf{C}$ vector would be (??).

Figure 5: Track hypothesis tree

$$
\begin{aligned}
\boldsymbol{A_1} &= \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{b_1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\
\boldsymbol{A_2} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \mathbf{b_2} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
\mathbf{c} &= \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 & \lambda_6 & \lambda_7 & \lambda_8 & \lambda_9 \end{bmatrix}^T
\end{aligned} \tag{19}
$$

The explicit enumeration that becomes necessary when creating these $\boldsymbol{A}$ matrices is exhaustive since the dimension of $\boldsymbol{\tau}$, which is equal to the number of leaf nodes in the track forest, can be very large. Both $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ grows quadratically with the number of track hypotheses and real measurements or number of targets respectively.

4.2 Solvers

There are a lot of off-the-shelf ILP and Mixed Integer Linear Programming (MILP) solvers on the marked, both free open source and commercial. Since the problem in this report is formulated on standard form, it can easily be executed on several solvers, and we can compare runtime and performance. To have the broadest possible test suite, both free and commercial solvers are used. As every solver have different approached to how they pre-process, divide and solve problems, their performance can vary from situation to situation. The following solvers are tested in this work.

- CBC (Free, COIN-OR)
- CPLEX (Commercial (Free academic), IBM)
- GLPK (Free, GNU)
- Gurobi (Commercial (Free academic), Gurobi)

5 Implementation

The aim of this section is to elaborate the decisions that were made during the implementation of the algorithm outlined in section **??** and **??**.

5.1 Programming language

This work is done as a part of a larger project (AUTOSEA) where all code is developed in Python or C++. Python is known for its large pool of modules, easy portability and quick programming, which makes it a great choice for research and development of algorithms. It is far from the fastest out-of-the-box programming language available, there are however side versions of Python like Cython that allows for optimization and compilation of Python to C code. If Graphics Processing Unit (GPU) acceleration should be possible and desirable, Python also have modules that allows for very easy and powerful usage of CUDA cores from Nvidia products.

5.2 Code structure

The implementation of a TOMHT in this project is structured in an object oriented way with two primary classes, a *Tracker* and a *Target* class. The Tracker is the class the user initializes and configures with parameters that are not target specific, like P_D, λ_ν, λ_ϕ, η^2, N and a state space model. The Tracker is also the root for the track forest, meaning that all tracks are trees of Target objects originating from a single Tracker object.

When a new track is initialized, an *initiateTarget* procedure is called on the tracker, and a new root for a track hypothesis tree is created in the form of a Target object. As new scans are received, the trackers *addMeasurementList* method is called with the measurements, where upon all targets are predicted, gated and created, scored, selected and pruned, before the result is return to the user, see Figure **??**.

Of the aforementioned steps, the prediction, gating and creation of new track hypotheses will be referred to as the *tree growing* task. The growing of the tree

is in the single-core case implemented as a recursive function that traverse the track forest and gates and creates new nodes whenever it encounter a leaf node as described in Algorithm **??**.

Algorithm 1 Tree growing

```
1: procedure TARGET::PROCESSNEWMEASUREMENTS
2:     if self.childs = None then
3:         self.predictMeasurement
4:         self.gateAndCreateNewNodes
5:     else
6:         for  do node in self.children
7:             node.processNewMeasurements
```

In addition to expanding the tree structure, the tree growing procedure updates a *set*[1] stored for each target, containing the measurements in the target tree. This set contains no new information compared to the tree structure, but increases the speed of the clustering later on since a set is hashed and can be accessed in constant time. The multi-core implementation of the tree growing is done slightly different, since the worker processes can not access the tree structure directly in Python. This is because Python pickle arguments that are sent to other processes, a procedure that converts the arguments to a byte stream. The effect of this pickling is that arguments are effectively passed as copies instead of references. The workaround for this is to run a procedure on each leaf node where a copy of that node is sent, and a processed node with added measurements are returned. Through this process, the copying of the large tree structure is avoided. These steps requires a complete list of all leaf nodes, which can be obtained fast through traversing the forest.

The clustering of the targets is done by building an adjacency matrix where all targets and nodes (measurements) in the track forest are represented. Targets and nodes are connected with edges where there is a usage of that measurement.

[1]Python: Unordered collections of unique elements

The adjacency matrix is then used in a connected components algorithm which returns labels on all the nodes according to their clustering.

After the clustering, the algorithm iterates each cluster and solve the optimization problem for clusters with more than one targets, and searches for the best hypothesis through traversing on clusters with one target.

5.3 Performance considerations

Of the previously mentioned steps, the growing is the most time consuming. For example, a scan which takes 1758 ms in total to process, 1496 ms are used to grow the track hypothesis tree when running on a single Central Processing Unit (CPU) core. Since the search after leaf nodes is little time consuming, there is a possibility for running all the leaf nodes in the track forest in parallel on a GPU. A GPU is built with a high number of cores, typically in the range from 200 to 2000. In comparison with CPU cores which are designed to execute different logical operations on each core at the same time, GPU cores are designed to execute the same logical operation on all its cores at the same time. This is very favourable for the process of tree growing.

According to Nvidia [2], it is possible to get performance increase up to 20x-2000x, when running pure Python code on GPUs. This could potentially reduce the grow time from about 1500 ms to 75-1 ms.

[2]https://developer.nvidia.com/how-to-cuda-python

6 Results

6.1 Testing scheme

The evaluation of the MHT algorithm is two-sided. Firstly the algorithm must be able to track under challenging conditions, and secondly it must be able to do so without having an ever growing computational cost and run time. The first performance metric is how well the algorithm is estimating the true position to the object it is tracking. We measure this by means of the Euclidean distance between the estimated and true track (**??**).

$$\Delta P = \|\mathbf{p}_{track} - \mathbf{p}_{target}\|_2 \tag{20}$$

The track is considered correct if $\Delta P \leq \varepsilon_p$ for all t after initial convergence. If a track is deviating more than the threshold and never return within the threshold again, it is considered lost at the time-step when it exceeded the threshold. If the track should converge after exceeding the threshold, it is considered restored at the time-step it is returning within the limit.

The algorithm is tested on six scenarios:

- Five fully cooperating ships
- Five partially cooperative ships
- Five ships avoiding obstacles with large space
- Five ships avoiding obstacles with little space
- Five almost parallel ships (normal speed radar)
- Five almost parallel ships (high speed radar)

(a) First scenario (b) Second scenario

(c) Third scenario (d) Fourth scenario

(e) Fifth scenario (f) Sixth scenario

Figure 6: True track for scenario 1-6

6.2 Simulation data

Scenario one through four was generated as a recording of time and position from an ASV simulator with Collision Avoidance (COLAV), developed by Dr. D. Kwame Minde Kufolaor at Norwegian University of Science and Technology (NTNU). The ships are configured such that they need to maneuver to avoid collision with each other, which allows for tracking of maneuvering targets in close proximity to each other. These scenarios where sampled at 1 Hz which is a little faster than a normal high speed vessel radar at 0.8 Hz (48 Rotations Per Minute (RPM)). The fifth and sixth scenario was generated as a part of this project, and is composed by linear parallel paths with white Gaussian system noise as maneuvering. These scenarios were sampled from the same data set at 0.5 Hz and 1 Hz respectively, where 0.5 Hz is a little higher that a normal maritime coastal radar at 0.4 Hz (24 RPM). The usage of two different sample rates were to compare the most common radar update frequencies used in the maritime sector with respect to tracking performance. The different ranges, Field Of View (FOV), and expected clutter points for the different scenarios are summarized in Table **??**.

			$\lambda_\phi \cdot FOV$				
Scenario	Radar range	FOV	1	2	4	6	8
1-4	$65m$	$1.3 \cdot 10^4 m^2$	1.3	2.6	5.2	7.38	9.21
5-6	$155m$	$7.5 \cdot 10^4 m^2$	3.22	4.61	5.99	7.38	9.21

Table 2: Expected number of clutter measurements in scenarios

6.3 Simulations

Scenario one through six was simulated with the following variations with all four solvers.

$$\mathbf{P_D} = \begin{bmatrix} 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \end{bmatrix}$$
$$\mathbf{N} = \begin{bmatrix} 0 & 1 & 3 & 6 & 9 \end{bmatrix}$$
$$\boldsymbol{\lambda_\phi} = \begin{bmatrix} 0 & 1 \cdot 10^{-4} & 2 \cdot 10^{-4} & 4 \cdot 10^{-4} & 6 \cdot 10^{-4} & 8 \cdot 10^{-4} \end{bmatrix}$$

Each of the 3600 variants was simulated 160 times with different seeded random clutter measurements and misdetections. These simulations required approximately 864 CPU hours, and were for the most part simulated on a 36 core virtualized server. For each simulation, the estimated tracks were compared with the true tacks and categorised in successful and lost tracks, where the track loss threshold ϵ_p was 4 meters. Figure **??** display a mid-simulation plot with current best tracks in solid lines, and hypotheses in dotted lines. Notice how many of the hypotheses have very similar paths, and with a certain margin duplicates each other.

Figure 7: Track hypotheses example

Figure **??** shows the track performance from the "GLPK" solver. The performance of the other solvers can be found in the appendix, Figures **??** - **??**.

(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

parallel_targets_0.5hz-GLPK.pdf

(f) Scenario 6

(e) Scenario 5

Figure 8: Simulation results for all scenarios

6.4 Runtime

Figure ?? to ?? displays the average the runtime for the different solvers. The runtime along the z-axis displays the time used for simulating the entire scenario.

Figure 9: Runtime for scenario 1

Figure 10: Runtime for scenario 2

Figure 11: Runtime for scenario 3

Figure 12: Runtime for scenario 4

Figure 13: Runtime for scenario 5

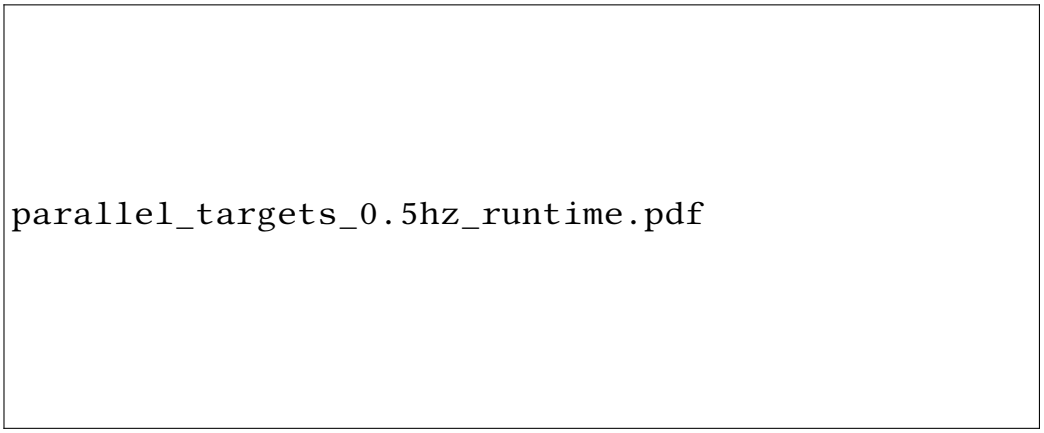

Figure 14: Runtime for scenario 6

6.5 Track loss performance

From the simulation results in Figure **??** to **??** in the appendix, it can be observed that the different solvers perform practically identically when it comes to track performance. From Figure **??**, it can be seen that the number of lost tracks is proportional to the clutter level at an inverse proportional rate to the probability of detection P_D. An interesting observation is the return on investment regarding the number of scans to evaluate (N-scan). For instance, with $P_D = 0.7$ the difference between $N = 3$ and $N = 6$ is marginal, at least when the clutter level is within reasonable levels ($\lambda_\phi < 4 \cdot 10^{-4}$). With $P_D = 0.5$ is can be seen that the pay-off is much higher for the extra computational cost with 15% improvement between $N = 3$ and $N = 6$.

Scenario 1-4 was simulated within a very small area where the targets had good separation both at the start and at the end of the simulation. This gives the algorithm more scans to figure out the correct association, after any multi-target conflicts that could arise when tracks are close to each other, and hence gives better track loss figures. In scenario 5-6 the targets start off spatially much closer

and are moving faster. This leads to a situation more vulnerable to misdetections, and is an example of a situation where a high N-scan is beneficial.

6.6 Time performance

From Figure **??** to **??** it can be seen that the solvers have similar execution times, though with a very consistent difference in scenario 1 to 4. In these scenarios the GLPK solver is the fastest with 5-10 seconds compared with the slowest in these scenarios, CPLEX. This trend is most likely caused by the different amount of overhead in the solvers, where IBM's CPLEX, which is marked leading in many ways, might have way more initialization and setup procedures compared GNU's GLPK. In scenario 5 and 6, a different and more varying trend is visible. Here we see that both Gurobi and GLPK are fastest in different situations, and generally all other solvers than CPLEX perform quite similar.

The run time increases linearly with the amount of clutter, which is expected as most of the operations run in the algorithm are based on tree operations with $O(E + V)$ run time. The structure in a track-tree implies that each vertex has one edge, hence any DFS based operations will have run time $O(2V) = O(V)$. The run time increases exponentially with the size of N, which is natural since the number of hypotheses that must be considered is exponentially larger.

6.6.1 Bottle necks

To study the effect parallel computation have on the tree growing task, a selected scan processing runtime was recorded with different amount of CPU cores. Table **??** compares the run time averaged over 10 runs for the same scan, with different amount of parallel processes on a 4 core computer.

Since the growing task is the only part of the algorithm that is utilizing on

Processes	Total	Grow	Cluster	Optimize	Prune

Table 3: Runtime of a scan with different amount of processes (time in ms)

multiple cores, it is the only task that changes significantly. The multi processing of the growing task is implemented in a master-worker configuration, where the task is divided into $n-1$ equal chunks running on $n-1$ cores with the main process handling all the "worker" processes. The run time increases when going from one to two processes, which is expected since the entire job is copied to another process, processed, and copied back. With any further increase in processor count, the execution time is expected to decline, which is also the case. At a certain core count, the time used to splitting up the task, distribute it to all the cores and collecting their results will become larger than the benefit of faster computation in each core. At this point, the overhead of parallizing the work leads to starvation of the workers and increased total run-time. On the four core machine, this limit is not reached, but by looking at Table **??** in the appendix, displaying the runtime of the same program on a server with virtual CPUs, it can be seen that the fastest execution time is with 6 cores. The scaling per core is worse on the virtualized server, possibly because the overhead of moving data to and forth the cores is larger. This means that running on a dedicated CPU with more than four cores could yield higher optimal core count and thus better runtime performance.

From Table **??**, we can see that the tree growing task is responsible for about one quarter of the run-time for this particular scan. Further analysis of the tree growing reveals that the creation of new nodes are the main contributor to the run time, with $97-98\%$ of the growing task.

Processes	**Grow**	**Cluster**	**Optimize**	**Prune**

Table 4: Runtime distribution percentage

Processes	Search	Predict	Create	Add

Table 5: Worker runtime distribution percentage

7 Discussion

The field of tracking and data association has been dominated by firstly military applications, and by secondly civil aviation control. Both of which are areas with hight performance demands and large development contracts [**?**]. Although much work is published on this topic, few comparisons have been made between actual implementations of different trackers and filters. This may to some extent be due to the nature of business secrets and military classifications to hide the intricate details that goes into an actual implementation and optimization of any tracking system. It is also, on a slightly more speculative note, due to disagreements about what kind of performance measures one should use.

One of the main objectives for this work was to examine the feasibility of using off-the-shelf solvers to solve the assignment problem in a MHT system. This report have shown that all of the tested solvers ere capable of solving the assignment problem. It has also shown that there are some drawbacks with using off-the-shelf solvers. The largest ones beeing that the assignment problem must be written to a file that the solver can interpret. This is a major drawback with the chosen approach, but can be partially compensated for with fast Solid State Storage (SSD) in stead of traditional rotating hard drives. Another major drawback is the need for explicit enumeration of all hypotheses in the ILP formulation, which is costly and in many situations unnecessary. This could possibly be addressed by column generating optimization methods [**?**], starting with a feasible set of columns and adding more and more columns (constraints) until it reaches the optimal assignment. Intuitively this makes sense, as most of the constraints will never be active, and therefore not needed for finding the optimal solution. This would, however, make the optimization a lot more tailor-made to this application, and render the usage of off-the-shelf solvers impossible.

8 Closing remarks

8.1 Conclusion

This report has shown that off-the-shelf ILP solvers, both free and commercial, are capable of solving the data association problem in a TOMHT. It has also provided evidence that the tracking performance does benefit very little from more than $N = 3$ scan history in moderate manoeuvring scenarios when $P_D > 70\%$, even with severe amounts of clutter.

Through simulations on a single-core implementation of a TOMHT in Python, it have also shown the feasibility for real-time applications with five closely spaced targets with $N = 5$ in a heavy clutter scenario ($\lambda_\phi = 1 \cdot 10^{-3}$), and $N = 6$ for a moderate clutter scenario ($\lambda_\phi = 5 \cdot 10^{-4}$). Further, this report has outlined the possibilities for significant rune-time improvements through the use of multi-core CPUs and GPUs.

8.2 Future work

More complex pruning and merging of hypotheses is an area where there might be substantial benefits to both performance and run time, since it allows for longer history. By nature, many hypotheses will have very similar paths (see Figure **??**) and could therefore be candidates for being merged. The challenge with this is that the assignment problem could be rendered infeasible and crash the algorithm. A novel approach could be to formulate the pruning as an optimization problem. With this approach it could be possible to also remove hypotheses with a low score, subject to the problem still being feasible.

In the maritime context, it could be very beneficial to use the maritime AIS to aid the tracking. There are different approaches to include this extra set of data, all with their pros and cons. One option could be to use the AIS track to weight the measurements from the radar; this way the measurement that are more likely to originate from a target is given a higher score. Another possibility is to do track-to-track fusion / filtering, where the best or N-best tracks from the

radar are filtered with the tracks from the AIS system. Track-to-track fusion can also be handled within the ILP framework, see [**?**].

A final but very important part, required for the system to be complete, is track initialization. As with AIS assisted tracking, there are plenty of ways to tackle this challenge, some of them mentioned in section **??**.