BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

MULTIMODAL SYSTEM FOR MULTI-OBJECT TRACKING IN REAL-TIME

MULTIMODÁLNÍ SYSTÉM PRO MULTI-OBJECT TRACKING V REÁLNÉM ČASE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. ADAM KUČERA**
AUTOR PRÁCE

SUPERVISOR **JAROSLAV ROZMAN, Ing., Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Kučera Adam, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Počítačové vidění
Název: **Multimodální systém pro multi-object tracking v reálném čase**
Multimodal System for Multi-Object Tracking in Real-Time
Kategorie: Zpracování obrazu
Zadání:

1. Nastudujte problematiku sledování více objektů - tzv. multi-object-tracking, dále jen MOT. Zaměřte se na řešení agregující polohová data z více různých senzorů.
2. Navrhněte obecné multimodální MOT řešení schopné pracovat v reálném čase a agregovat data z více lokalizačních/trackovacích senzorů s odlišnou přesností, latencí, dostupností atd. Navrhněte dostatečně obecné vstupní/výstupní datové rozhraní k tomuto systému, které umožní pomocí adaptérů/ovladačů přijímat polohová data např. z dopravních kamer, GPS senzorů ve vozidlech, nebo dopravních radarů.
3. Implementujte navržený systém a ukažte jeho schopnosti na simulovaných a reálných datových sadách. Zaměřte se na posouzení přesnosti a robustnosti tohoto řešení, simulujte výpadky dílčích senzorů a testujte různé jejich kombinace.
4. Vytvořte plakát a video demonstrující vaši práci. Nastiňte možný budoucí vývoj.

Literatura:

- Challa, S., Morelande, M., Mušicki, D., & Evans, R. (2011). Fundamentals of Object Tracking. Cambridge University Press.
- E. R. Davies (2017). Computer Vision, 5th Edition. Academic Press.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 3. listopadu 2021

Abstract

This thesis deals with the topic of multi-object multi-sensor tracking. A conventional track-oriented multiple hypothesis tracking (TOMHT) pipeline is implemented in C++ programming language and an implementable interface is designed, enabling to easily extend the core algorithm with arbitrary sensors and measured target attributes, making the system multimodal, i.e. applicable in heterogeneous systems of sensors. A novel algorithm for solving combinatorial optimization arising in TOMHT is proposed. Finally, few example implementations of the interface are provided and the system is evaluated in simulated and real-world scenarios.

Abstrakt

Tato práce se zabývá tématem multi-objektového multi-senzorového sledování. V programovacím jazyce C++ je implementován konvenční řetězec pro track-oriented multiple hypothesis tracking (TOMHT) a je navrženo implementovatelné rozhraní, které umožňuje snadno rozšířit základní algoritmus o libovolné senzory a měřené cílové atributy, čímž se systém stává multimodálním, tj. použitelným v heterogenních systémech senzorů. Je navržen nový algoritmus pro řešení kombinatorické optimalizace vznikající v TOMHT. Nakonec je poskytnuto několik příkladů implementace rozhraní a systém je vyhodnocen v simulovaných a reálných scénářích.

Keywords

object tracking, multiple-object tracking, multiple-hypothesis tracking, multiple-sensor tracking, heterogeneous systems of sensors, data fusion, data association, tracking in geo-registered space, extendable object tracking framework, computer vision, combinatorial optimization

Klíčová slova

sledování objektů, sledování více objektů, sledování přes více hypotéz, sledování s více senzory, heterogenní systémy senzorů, fúze dat, asociace dat, sledování v geo-registrovaném prostoru, rozšiřitelný systém pro sledování objektů, počítačové vidění, kombinatorická optimalizace

Reference

KUČERA, Adam. *Multimodal System for Multi-Object Tracking in Real-Time.* Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Jaroslav Rozman, Ing., Ph.D.

Multimodal System for Multi-Object Tracking in Real-Time

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Rozman. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Adam Kučera
May 18, 2022

Contents

1 Introduction **3**

2 Basics of Object Tracking **5**
2.1 Conventional Pipeline . . . 5
2.2 Motion Filtering and Prediction . . . 6
2.2.1 Kalman Filters . . . 6
2.2.2 Particle Filters . . . 8
2.2.3 Recurrent Neural Networks . . . 9
2.2.4 Interacting Multiple Model . . . 10
2.3 Data Association . . . 10
2.3.1 Global Nearest Neighbors . . . 11
2.3.2 Joint Probabilistic Data Association . . . 12
2.4 Multiple Hypothesis Tracking . . . 13
2.4.1 Reid's Algorithm . . . 13
2.4.2 Track-Oriented Approach . . . 15

3 Sensors **18**
3.1 Vision Sensors . . . 18
3.1.1 Image Capturing . . . 18
3.1.2 Object Detection . . . 19
3.2 Emitting Sensors . . . 20
3.2.1 Radar . . . 20
3.2.2 Lidar . . . 20
3.3 Collaborative Localization . . . 21
3.3.1 V2X - Vehicle to Everything . . . 21

4 Visual Trackers **22**
4.1 Miscellaneous Algorithms . . . 22
4.1.1 Template Matching . . . 22
4.1.2 Mean Shift . . . 23
4.1.3 Optical Flow . . . 23
4.2 Tracking with Machine Learning . . . 24
4.2.1 Multiple Instance Learning . . . 24
4.2.2 Tracking-Learning-Detection . . . 25
4.2.3 Deep Regression Networks . . . 25
4.3 Correlation Filters . . . 26
4.3.1 MOSSE Correlation Filters . . . 26
4.3.2 Kernelized Correlation Filters . . . 26

4.3.3 CSR-DCF Tracker . . . 27

5 Proposed Tracking System **28**
5.1 Motivation . . . 28
5.2 Programmable Interface . . . 29
5.2.1 Observations, Attributes and Attribute Model . . . 29
5.2.2 Sensor Models and Predictive Model . . . 30
5.2.3 Detailed Overview . . . 30

6 Implementation **34**
6.1 Tracking Engine . . . 35
6.1.1 Track Tree Data Structure . . . 38
6.1.2 Solving Data Association . . . 39
6.2 Example Implementations . . . 44
6.2.1 Linear Motion Predictor . . . 45
6.2.2 Simple Camera Model and IoU . . . 45
6.2.3 Multi-Camera Model and Appearance Embeddings . . . 47

7 Evaluation **53**
7.1 Simulation . . . 53
7.2 Real-World Scenarios . . . 56
7.2.1 MOT Challenge 2020 . . . 56
7.2.2 Multi-Camera Road Junction Monitoring System . . . 57

8 Conclusion **60**

Bibliography **61**

A Contents of the included storage media **67**

Chapter 1

Introduction

In the area of intelligent systems, the (multiple) object tracking is a well known but non-trivial task. The goal of object tracking system is to maintain the identity of physical target throughout the time, while its location is a subject to change. Also, in general, the tracker is not given any prior information about the number of targets in the scene, which is usually a function of time anyway, thus it is responsible for determining this as well. Although the target detection is a standalone task, it is sometimes viewed as a part of the tracking system. However, the core component of the tracker is an algorithm, which is able to solve the data association problem, arising from the task of object tracking. This data association problem has often a combinatorial character, since in general case of multiple targets in the scene, globally optimal solutions are strongly preferred. Beyond the essential properties such as position and velocity, determination of other attributes of the target object sometimes takes the place (e.g. class of the object). While the fundamental theory as well as the first software solutions dates back to the early years of the second half of 20th century, there seems to be a recent ascend of the multi-object tracking applications. Beyond the classical applications such as military (e.g. missile defense), scientific (e.g. tracking the atmospheric phenomena), medical imaging (e.g. movement of proliferating cells), more ordinary applications are on the rise, such as face tracking or augmented reality. An interesting modern application of multi-object tracking in particular, is e.g. realtime traffic analysis[1], in which tracking of traffic objects using advanced sensors such as digital cameras or radars, instead of a simple induction coil in the road, can significantly help with adaptive traffic control.

In the following chapters, I provide an overview of basic theoretical background, methods, algorithms, and finally I propose a new multimodal system for multi-object tracking designed to be a general template rather than a specific implementation. The thesis is structured as follows. In chapter 2, general theory and a slightly more involved introduction to the object tracking task is given. The traditional approaches and methods are described, which even though being developed decades ago, have no newer alternatives that could be considered undoubtedly better and the fundamental theory in the area could be labeled as "mature". In chapter 3, I describe some examples of sensors that are usually used to provide an information about the outer world to the tracking system, including not only physical sensors but the preprocessing software as well. Because the modern object tracking is highly involved in the area of computer vision and many tracking algorithms focusing on the image data have been developed, chapter 4 will present some of those visual trackers. In chapter 5, I discuss the motivation, inspiration, overall design, and an

[1] https://datafromsky.com

implementable interface of the proposed multimodal multi-sensor tracking system. The implementable interface is designed to enable extending the core tracking engine for arbitrary input sensors and measured attributes, by implementing all the necessary operations and probabilistic equations. The tracking engine solves the data association problem following the theory in chapter 2, is fixed but configurable, and provides explicit mathematical guarantees about the solutions given that the input data and all the statistics associated with them are correct. In chapter 6, I describe in more detail the algorithm I've constructed for solving the combinatorial optimization. An algorithm builds on the previous work in various areas, but can be considered a novel approach in the area of multi-object tracking. Then, few particular implementations of the interface that I've programmed are described. Finally, in chapter 7, an evaluation of the tracking system is given. First, the system is evaluated in simulations, which provide an idealistic scenario when the probabilistic models are known exactly and the only source of error should be the uncertainty inherent to the observations. I show that with an increasing size of the temporal context in which multiple hypotheses are considered (MHT paradigm, see chapter 2), and with an increasing number of sensors, the overall performance of the tracker grows consistently. Then, it is shown that the single-sensor implementation of the proposed system exhibits a state-of-the-art performance in the realm of realtime pedestrian tracking on the real-world data from MOT Challenge 2020 benchmark[2]. Finally, I do show a multi-camera implementation employing appearance embeddings from a neural network, tracking through up to 4 cameras simultaneously on example videos from a road junction monitoring system. Although I was not provided with the ground-truth, I've implemented a visualization application, where the trajectories are projected from the physical plane back to the image plane of selected camera, and a visual evaluation is possible.

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

Chapter 2

Basics of Object Tracking

In this chapter, I provide a more detailed introduction into the object tracking in general. It is focused on the tracking in general, therefore details about the sensing and detection are omitted. Discussion about this is provided in chapter 3. Here, I will refer to the output of the two steps universally as an observation. These observations may be collected in different ways w.r.t. time, they may be measured in a regular intervals, or they may occur irregularly. I will follow the terminology from the literature [15], where the term “scan” is usually used for the set of observations present concurrently at some point in time. The object tracking task can be further divided into the narrow single object tracking (SOT) and the general multiple object tracking (MOT). Although it seems that there is an inclusive relationship between the two, it is incorrect. It is possible to decompose the MOT problem into multiple instances of SOT, but the global optimality of MOT is lost this way. Another drawback is that while the solutions for SOT can be quite robust (especially in visual trackers described in chapter 4), it is typically because all the computational resources are allocated for tracking only single object, instead of tens or hundreds in case of MOT. Therefore, the realtime performance is out of the reach for many such approaches. In the rest of this chapter, I will focus on a more general and more challenging MOT problem.

2.1 Conventional Pipeline

At every measurement scan from the sensor(s), each input observation is considered for inclusion in existing tracks, for initiation of the new track, or as a false alarm. The candidate observations for the true target are usually selected within its validation gate. This gate is a region in some measurement space (e.g. 3D position) around the expected true target location, within which the observations of it are admissible. Thus, it is chosen such that the probability of observing the true target outside this region is sufficiently low. This step may, given an accurate model, highly reduce the overall computational requirements of the system. After the candidate observations are assembled for each hypothesis, a data association conflicts are likely to occur and has to be solved. Closer look at those conflicts and their solutions as an essential part of a tracking system is provided in section 2.3. After the conflict situations are resolved (this solution may be deferred, as is described in section 2.4) by finding the best (ideally global) assignment, the track maintenance takes the place. In this step, new tracks are initiated from the unassigned observations, previously initiated targets may be confirmed, and targets which are likely to be not present anymore are deleted. It is a good practice to mark newly initiated tracks as tentative and confirm them after there is

enough evidence that those are actually targets and not just false alarms. A naive but good approach is to confirm a new target identity after N consecutive updates. Same goes for the deletion by choosing a proper time-to-live value. In specific applications however, the number of targets may be fixed and apriori known (e.g. tracking of football players). After the maintenance step, the filtering and prediction steps follow. Those two steps can highly improve the tracking accuracy and can be crucial in order to overcome missing observations and/or low measurement frequency. More detailed description of models used for this step is provided in section 2.2. The conventional tracking system operation described above can be summarized in the following steps [9], visualized in the figure 2.1.

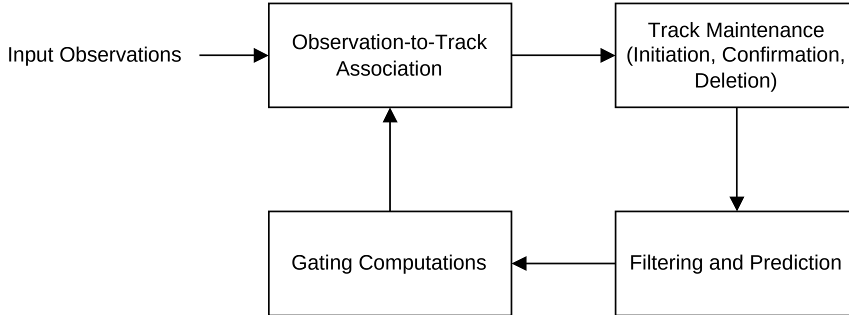


Figure 2.1: Conventional MOT pipeline

2.2 Motion Filtering and Prediction

In order to provide a good prediction about the expected future true target location, a model for filtering and prediction has to be employed. The filtering step in can be described as an informed update of the target location estimate based on its current probability density function and the observation which is assigned to it. This probability density typically expresses the likelihood of the observation z coming from the given target, which when maximized, leads to the well-known maximum likelihood estimate (MLE). In practice, the function used is always just an approximation of the true probability density, due to theoretical and computational reasons often a very simplistic one. While the filtration step in this context means a posterior update to the target probability density, the prediction step on the other hand can be seen as a prior update. The simplest prediction assuming no model (e.g. random walk) can be used, but in practical tracking scenarios the objects usually do not move in a Brownian motion style, but rather follow trajectories which could be predicted more accurately by more informed motion model. Design of the filtration and prediction model has a huge impact on the performance of the tracking system as a whole and is typically used in gating computations as well.

2.2.1 Kalman Filters

One of the best-known predictor-corrector models in the area of control theory, tracking, navigation, etc., is a Kalman filter [44]. The Kalman filter is an optimal Bayesian state estimator for linear discrete dynamical systems given the premise of Gaussian errors (for both predictions and measurements). Thanks to this simplification, the optimal estimates can be obtained by a recursive application of closed form linear algebra equations. The filtering and prediction for each target-measurement assignment is performed in two steps,

however both are a closed form equations and thus could be merged together. The prediction step can be expressed by the equations (2.1, 2.2), where x_k and P_k are target location mean and probability density covariance respectively ($\hat{x}_k$ and $\hat{P}_k$ denote the optimal posterior estimates), F_k is a transition model and Q_k is a process noise covariance (representing unknown changes beyond the transition model), all at scan k:

$$x_k = F_k \cdot \hat{x}_{k-1} \tag{2.1}$$

$$P_k = F_k \cdot \hat{P}_{k-1} \cdot F_k^{\mathsf{T}} + Q_k \tag{2.2}$$

In the filtering (or correction) step, the optimal posterior estimate given the observation is computed, maximizing the joint probability of that observation and the predicted target location statistics. It is efficiently done following the equations (2.3, 2.4, 2.5, 2.6), where S_k is the innovation covariance (expressing the likelihood of the observations), K_k is called Kalman gain, z_k is the measurement, H_k maps the state x_k into the measurement space, and R_k is the measurement noise (the error of the sensor), all at scan k:

$$S_k = H_k \cdot P_k \cdot H_k^{\mathsf{T}} + R_k \tag{2.3}$$

$$K_k = P_k \cdot H_k^{\mathsf{T}} \cdot S_k^{-1} \tag{2.4}$$

$$\hat{x}_k = x_k + K_k \cdot (z_k - H_k \cdot x_k) \tag{2.5}$$

$$\hat{P}_k = (I - K_k \cdot H_k) \cdot P_k \tag{2.6}$$

From those equations it can be noticed that the Kalman gain defines, loosely speaking, how strongly the predicted internal state should be updated based on the observation. The transition and measurement matrices are constructed according to the chosen motion and observation models. In the context of object tracking, the linear constant velocity model is usually a good approximation, but it may be quite problematic when targets do undergo strong maneuvers. One dimensional constant velocity model is defined by (2.7), where p is a position, t is time, and $\dot{p}$ is position derivative w.r.t. time (velocity):

$$x = \begin{bmatrix} p \\ \dot{p} \end{bmatrix}, \ F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \ H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.7}$$

A huge practical issue with the Kalman filter is that it is truly optimal only when all the assumptions hold and the covariances are known exactly. While the measurement noise can be often derived from the sensor specification, the process noise is frequently set in a trial-and-error manner and finding a proper setting for a given application is time and resource consuming. There has been some work done on methods for automatic estimation of those covariances. For example, in [57] an autocovariance least-squares method which solves the problem using a convex optimization has been proposed, having a solid mathematical foundation, but being restricted to covariances constant w.r.t. time. In [1], the adaptive approach has been introduced, where the covariances are estimated recursively by running an exponential average of measurement innovations and residuals, i.e. how do observations deviate from the predictions and how from filtered estimates, respectively. They've shown good results on dynamic state of synchronous machines estimation, but I've found their approach to be prone to divergence in object tracking scenarios due to potentially high hysteresis. In [3], a Bayesian approach has been proposed, where the distributions of both the state and parameters (the covariances) are calculated and the family of those parameter-dependent Kalman filters is referred to as a Field Kalman Filter (FKF). There

are also adaptive approaches, where the covariances are set dynamically based on the motion model. One popular example is a random acceleration model [65], where the process noise covariance Q_k is estimated using the assumption of a random acceleration of the target object with variance σ_a. Thus, the acceleration occurs randomly during each discrete step, which leads to a linear change in velocity and quadratic change in position (due to the integration) during this step. In a one dimensional case, this yields the expression (2.8) for Q_k:

$$Q_k = \sigma_a^2 \cdot \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{2}\Delta t^2 & \Delta t \end{bmatrix} = \sigma_a^2 \cdot \begin{bmatrix} \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 \\ \frac{1}{2}\Delta t^3 & \Delta t^2 \end{bmatrix} \tag{2.8}$$

It is a common technique to condition the value of σ_a on some heuristic, e.g. velocity of the object. Again, this can suffer from hysteresis. Despite those (and more) methods being available, the optimal decision of the Kalman filter covariances is still a challenging problem. However, because of the algebraic nature of the Kalman filter equation(s), it is not necessary to know their values - the ratio of the covariances is sufficient to define the result. Therefore, one can just experiment with his believes about this ratio rather than trying to find out the exact values, if the actual probability density function values are not required later on.

The linearity condition can be limiting, as many dynamical systems are non-linear. If Gaussian errors premise is still reasonable, the Kalman filter can be easily extended by the 1st order Taylor expansion around the current state estimate, i.e. local linearization. This extension is known under the abbreviation EKF [22]. Although the transition and measurement mappings f and h cannot be performed by matrix multiplication anymore, the equations for x_k, P_k and K_k are preserved and the Jacobian matrices of partial derivatives of f and h are used as F and H matrices. More precisely, the following matrices has to be computed and evaluated (2.9), where n is a hidden state dimensionality and m is a measurement dimensionality:

$$F_k = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}, \; H_k = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix} \tag{2.9}$$

Another way of extending the Kalman filter, even for a non-Gaussian uncertainty statistics, is known as unscented Kalman filter [41] (UKF). In this approach, no linearization is required. The basic idea is to use a set of σ-points, for which the sample mean and sample covariance are equal to x_k and P_k, respectively. These points can then be transformed with a non-linear mapping and the statistics can be re-estimated. It is important to note that even though the distribution does not have to be Gaussian, the mean and the covariance have to be sufficient statistics for it in order to achieve good estimates. This method differs from the Monte-Carlo type methods in the way that the σ-points are not drawn randomly, but according to a deterministic algorithm. The UKF outperforms EKF in highly non-linear systems, but for the cost of increased computational complexity.

2.2.2 Particle Filters

Particle filters [32] are a nonlinear and non-Gaussian alternative to Kalman filtering (and a generalization of the UKF). The particle filters are especially popular in SLAM (simulta-

neous localization and mapping) problems and robotics [55], and have found their way into the single object tracking as well, but they are not very used in multiple object tracking, mostly due to their huge computational complexity. Despite this drawback, particle filters are a powerful concept with considerable potential for object tracking. The general idea behind the particle filters is to use N random realizations called particles, to approximate an arbitrary probability distribution. As $N \to \infty$, the approximation is theoretically exact. The prediction step is performed for each particle denoted by $i = \{1, \dots, N\}$ according to the (possibly non-linear) transition function (2.10), where $w_{i,k}$ is a sample drawn from the process noise probability density function:

$$x^{-}_{i,k+1} = f_k\left(x^{+}_{i,k}, w_{i,k}\right) \tag{2.10}$$

Then, in the filtering step, the discrete probability mass function q (2.11) is evaluated for all the particles, where z_k is an observation from the scan k:

$$q_i = \frac{p\left(z_k \mid x^{-}_{i,k}\right)}{\sum_{j=1}^{N} p\left(z_k \mid x^{-}_{j,k}\right)} \tag{2.11}$$

The posterior particles $x^{+}_{i,k}$ are obtained by resampling the prior particles according to q, i.e. their importance. A successful application of the particle filter for SOT in video sequences can be found for example in [54].

2.2.3 Recurrent Neural Networks

A different approach to the filtering and prediction step is a recurrent neural network [20], which employs machine learning and learns to perform both filtering and prediction jointly. There is a strong parallel between recurrent neural networks and the Bayesian filters such as Kalman or particle filter, because they both recursively estimate a hidden state h_k, and an output state y_k based on their input x_k at the current discrete time k (may be scan in the case of object tracking). The key difference is that while the Bayesian filters do require an explicit model (usually designed by engineers), and they require explicit probability density functions for both prediction (prior) and filtering (posterior), the neural networks do not. They instead require a dataset of training data (the larger, the better) consisting of pairs of ground-truth locations and corresponding (noisy) observations. The recurrent neural network parameters are then optimized by the gradient descent method (or some improved version of it) in order to minimize a regression loss function, which is usually the mean squared error of the residuals.

In [33], the direct comparison of Kalman filter and LSTM based recurrent neural network has shown a superior performance of the LSTM both on a toy example and the main goal, which was facial landmarks localization and tracking. Despite this and other papers showing great results for the neural network based time series filters, it is not true that they can replace the Bayesian filters. Those two are a distinct approaches, both having their strong and their weak sides. The main downside of a recurrent neural network is that is it a black box. It is just a sequence of linear transformations interleaved with non-linearities, which are optimized to minimize the error on the training data. But there are usually a very few theoretical justifications and guarantees (if any) behind their architectures. The lack of interpretability makes them prone to unexpected behavior on situations not captured in the training set and also to be computationally inefficient due to possibly redundant parameters.

On the other hand, the Bayesian models have a very strong mathematical foundation and are directly interpretable. I suppose that a conjunction of those two paradigms may be an interesting future direction.

2.2.4 Interacting Multiple Model

Models mentioned above have one disadvantage - a single transition function for prediction. In object tracking (especially for ground targets), the complex and hardly predictable maneuvers do occur, and thus it is very complex (if even possible) to construct a transition function capable of accurately modeling all the possible patterns of motion. Interacting multiple models or IMM [29, 9] is an interesting way to tackle this challenge. Even in such scenarios, a simple motion model such as constant velocity Kalman filter (CEKF) may be an efficient approximation of the true dynamics during some interval in time, however, at some point a rapid maneuver may occur (object takes a sharp turn, decelerates too fast, etc.), which can lead to poor performance or even divergence of the filter. IMM solves this problem by running multiple filters in parallel, each designed and fine-tuned for a different motion pattern, e.g. a triplet of a constant velocity, constant acceleration and constant-speed turn models. Each model has its probability of being "the correct one", which is expressed by the probability vector μ. Each model is first updated using the standard Kalman equations (section 2.2.1). Then, the mixed state for j-th model is computed by equations (2.12, 2.13, 2.14), where $\pi^{i,j}$ is the prior probability of the transition from state i to state j, all at scan k:

$$\mu_k^{i,j} = \frac{1}{c} \cdot \pi^{i,j} \cdot \hat{\mu}_{k-1}^{i} \tag{2.12}$$

$$\hat{x}_k^{0j} = \sum_{i=1}^{N} \mu_k^{i,j} \cdot \hat{x}_k^{i} \tag{2.13}$$

$$\hat{P}_k^{0j} = \sum_{i=1}^{N} \mu_k^{i,j} \cdot \left[\hat{P}_k^{i} + \left(\hat{x}_k^{i} - \hat{x}_k^{0j} \right) \cdot \left(\hat{x}_k^{i} - \hat{x}_k^{0j} \right)^{\mathsf{T}} \right] \tag{2.14}$$

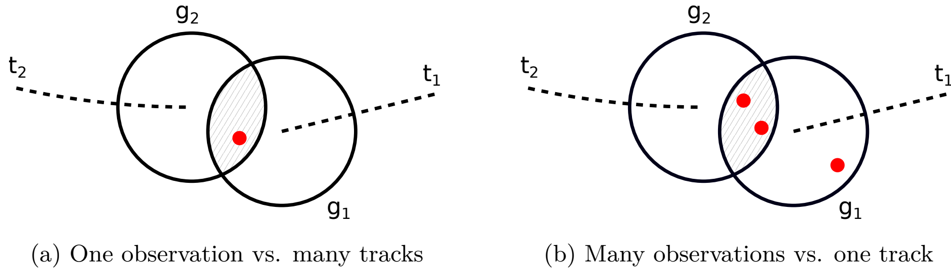
The posterior models probabilities $\hat{\mu}_k^i$ are then updated and plugged into the equations (2.13, 2.14) instead of term $\mu_k^{i,j}$ (omitting the superscript $0j$), to compute the combined state of the IMM as a whole. Values of $\hat{\mu}_k^i$ are updated simply to be proportional to the likelihoods of the observation z_k given individual models, which can be written by the equation (2.15), where S_k^{0i} is the innovation covariance, as is expressed in equation (2.3) from section 2.2.1, but using the mixed states x_k^{0i} and P_k^{0i}:

$$\hat{\mu}_k^{i} = \frac{1}{c} \mathcal{N} \left(z_k \,|\, x_k^{0i}, S_k^{0i} \right) \tag{2.15}$$

2.3 Data Association

The most important part of any multi-object tracking algorithm is the data association step. In this step, the measured observations at scan k are assigned to the (possibly new) targets, or discarded as false alarms. Also, observations for a confirmed existing target can be missing due to the sensor or detector failure, or due to the occlusion. Observations are also subject to measurement error. A tracking system is expected to perform the (near) optimal decisions, given such fuzzy information about the outer world. Therefore, it should

guarantee that, given perfect statistical models of all the uncertainties, the most likely solution is chosen. The problem is not very hard, when for every target, there is only one observation within its validation gate, or when there is only single target. When the observation falls into the validation gate of more than one target, or when there is more than one observation within the validation gate of a single target, the problem complexity increases dramatically. Mentioned situations are illustrated in the figure 2.2, where the red dots represent observations and circles illustrate the validation gates.



(a) One observation vs. many tracks

(b) Many observations vs. one track

Figure 2.2: Data association problems illustration

In order to find the optimal data association, given the probabilistic models, a simplifying assumptions are further made. More precisely, the common premise along traditional tracking algorithms is that single observations can not origin from more than one target, where a false alarm is usually viewed, for better theoretical consistency, as a special target as well. Often, even more strict assumption is made, that there is at most one observation per target (e.g. GNN in section 2.3.1).

2.3.1 Global Nearest Neighbors

Given a scoring metric (e.g. appearance similarity, geometric affinity, or statistics from the motion filter), the problem can be stated as a linear assignment problem. This approach is traditionally called global nearest neighbors (GNN). The linear assignment problem is a classical combinatorial optimization problem. The goal is to find the set of edges in a biparite graph with weighted edges, such that the sum of all the weights is being minimized. The two disjoint sets of the biparite graph are in this case the set of N targets present during the scan $k-1$, and the set of M observations from the scan k. Weights are the association scores and if those scores are negative log-likelihoods of the respective assignments, then the sum minimization is in fact maximization of the joint probability of the assignments, i.e. maximum likelihood estimate. The linear assignment problem happens to be a special case of the max-flow problem, which can be solved by Ford-Fulkerson algorithm [25]. Although the max-flow problem is NP-hard, a Kuhn-Munkres algorithm [47], sometimes referred to as a Hungarian algorithm, has been derived and mathematically shown to solve the problem in $O\left(n^3\right)$ time complexity using $N \times M$ matrix representation G of the biparite graph. From the implementation point of view, the algorithm is relatively simple. It can be summarized in 4 steps (denoting i-th row as $G_{i,*}$ and j-th column as $G_{*,j}$):

1. For $1 \leq i \leq N$, subtract $\min\left(G_{i,*}\right)$ from $G_{i,*}$
2. For $1 \leq j \leq M$, subtract $\min\left(G_{*,j}\right)$ from $G_{*,j}$
3. Try to cover all zeros with $\min\left(N, M\right)$ lines, return solution if successful

4. Create additional zeros and goto step 3

Steps 1 and 2 are pretty straight forward. In step 3, the covering procedure can be done by any greedy method. In step 4, creating additional zeros is done by finding the smallest coefficient not covered by lines in step 3, and subtracting this value from all uncovered coefficients. Also, it is added to those coefficients, which are covered twice (lines intersect). If successful, unique combinations of zero coefficients covering $\min(N, M)$ rows and columns represent the optimal assignment (there can be more equivalent solutions, but not really in real world situations). Sometimes, other algorithms such as auction algorithm [6], are used instead.

2.3.2 Joint Probabilistic Data Association

In case of GNN, one can notice that what happens is that the hypothesis that maximizes the probability of the scan is obtained. However, due to the imperfections in the scoring metric and as well as the inherent uncertainty in the problem itself, such solution might be wrong. Other alternative hypotheses should be considered also. In [26], this was addressed in solving a challenging passive sonar tracking with multiple sensors and targets, in which targets were not fully observable from a single sensor and there was very high clutter in the sensors. They have developed the joint probabilistic data association (JPDA) algorithm, which started a whole new paradigm in MOT. The basic idea originates from [4], where a probabilistic data association has been proposed for SOT. Instead of assigning the most probable single observation to the target, all observations within its gate are used to compute an expected observation. The innovation for the filtering step (e.g. Kalman filter) is aggregated from the measurements $z_{i,k}$ within the target validation gate by the equation (2.16), where β_i is the probability that the measurement $z_{i,k}$ is the correct one (β_0 is the probability of missing observation), and y_k is the predicted prior state:

$$d_k = \sum_{i=1}^{N} \beta_i \left(z_{i,k} - y_k\right) \tag{2.16}$$

In SOT, for any observation association, the only alternative hypothesis is that all the other observations are false alarms. In MOT, there can be more alternative hypotheses. JPDA modifies the basic idea for MOT, by considering all the feasible joint events and computing the β_i values as marginal probabilities. The event is feasible if and only if no more than one observation originates from one target. The probabilistic nature of the JPDA makes it relatively reasonable to relax the second constraint, i.e. one observation originates from at most one target. The joint probability for every feasible event can be evaluated. In [26], the Gaussian distribution was used for the measurement noise. For false alarms and target births, Poisson distribution in time and uniform distribution in space were used. However, an arbitrary distributions may be used as well, building on the same core idea. The value of β_i^j, where j denotes the target object, is computed using the equations (2.17, 2.18), where Ω is the set of all feasible events and $\psi_{i,j}(\omega)$ returns 1 if the observation i is assigned to the target j in ω, otherwise returns 0:

$$\beta_i^j = \sum_{\omega \in \Omega} P(\omega)\, \psi_{i,j}(\omega) \tag{2.17}$$

$$\beta_0^j = 1 - \sum_{i=1}^{N} \beta_i^j \tag{2.18}$$

Despite being a successful method for tracking and has a reasonable theoretical background, JPDA can suffer from coalescence of objects, i.e. targets that are very close together for an extended period of time tend to merge together, which is undesired.

2.4 Multiple Hypothesis Tracking

In GNN method (section 2.3.1), the most likely hypothesis is expanded at each scan. The weak point of this paradigm is that in many situations, it is very uncertain if the most likely hypothesis given only the past and the present information, is actually correct. In those difficult to solve scenarios, the tracking system is prone to make wrong decision, and because it has expanded only one hypothesis, unfortunately the incorrect one, it is not able to recover from that mistake. In JPDA method (section 2.3.2), this problem is partially addressed, but only indirectly, in the sense that the information from alternative hypotheses in the past is indirectly propagated by the current state of the motion filter. In multiple hypothesis tracking (MHT) paradigm, the solution to the data association step in the conventional pipeline is deferred as much as possible. All the possible hypotheses are expanded concurrently and the most likely one is chosen when it is required, e.g. to provide the user with the results. Therefore, the MHT truly and directly solves the stated problem. From a theoretical perspective, this formulation yields the exact Bayesian solution to the tracking problem. From a practical perspective, however, it is not so ideal, due to the "exponential explosion" of the space of possible hypotheses, which makes MHT computationally intractable without intensive pruning of the unlikely hypotheses.

2.4.1 Reid's Algorithm

The first algorithmic approach for multiple hypothesis multi-object tracking with variable number of targets, the Reid's algorithm, was proposed in [60]. Algorithm generates a set of data-association hypotheses for all possible origins of every measurement, thus being now called "measurement oriented". The hypothesis generation process can be illustrated by a tree, e.g. figure 2.3, where each level represents one measurement and there are two targets which can be assigned to the measurements, denoted by 1 and 2. Nodes 0 represent false alarms and nodes 3, 4, and 5 represent new identities initiated from the measurements.

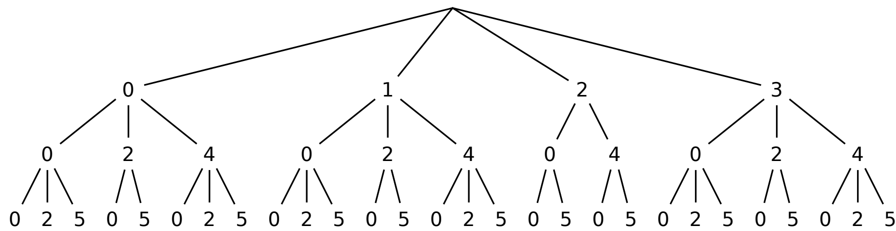


Figure 2.3: Hypothesis tree example

In [60], this tree was represented as a matrix in the computer (one row = one hypothesis), and subroutines for hypothesis reduction and clustering were described. As can be seen from the example above, where each leaf node corresponds to a single hypothesis (the path to the root is the actual hypothesis), the tree grows exponentially and makes the MHT extremely computationally and spatially demanding for larger problems.. The hypothesis reduction is done by pruning the tree, removing the unlikely hypotheses. Also, hypotheses

can be combined when they are similar, which means that the estimates of locations of all targets has to be sufficiently close, e.g. in terms of mean and covariance. Also, the number of tentative targets must be the same. For further reduction, a method using the k-best assignments Murty's algorithm [56] to find top k hypothesis was proposed in [17]. The Murty's algorithm is an efficient method for finding k best solutions to the linear assignment problem. It is based on the set theoretical analysis of the set of all feasible global assignments in the biparite graph. First, the optimal solution $a(1) \in A$ is found (e.g. using Kuhn-Munkres algorithm from section 2.3.1). Then, using the terminology from [17], a node N is defined as a non-empty subset of A such that some cells in the cost matrix G are enforced to be contained and some are excluded from each assignment in N. In order to induce a $(k+1)$-th assignment, given k-th one, the current node (A for $k=1$) is "partitioned" by its minimum assignment into list of nodes, solving at most $n-1$ linear assignment subproblems of sizes $2, 3, \ldots, n$, where n is the size of the current node. Partitioning nodes are carefully constructed in such way, that they are mutually disjoint, none of them contains $a(k)$, and their union with $a(k)$ results in the partitioned node. This way, k-best hypotheses can be find in an decreasing order based on their likelihood. The hypotheses can also be clustered. The observation is associated with a cluster if it falls within the validation gate of any target in that cluster. A new cluster is formed for observations which are not associated with any of the present clusters. Clusters are thus the collections of tracks that are (transitively) linked by common observations, and effectively make the large problem more manageable.

Very important contribution of [60] is the recursive probabilistic formula, evaluating the probability of the hypothesis up to the scan k. As usual, for the process and measurement noise, Gaussian statistics were used, and for the detection probability, Bernoulli process was a natural decision. However, for false alarms and target births, both temporal distribution (i.e. the probability of the given number of occurrences in the scan) and spatial distribution (i.e. the probability density of occurring at the given location) are required to properly evaluate the hypothesis. After pedantic combinatorial analysis, it has been shown that when Poisson process is plugged in for the temporal distribution, the hypothesis-dependent factorial terms cancel out. Also, when uniform density is plugged in for the spatial distribution, the dependence on the measurement volume is also eliminated. After simplifying all the constant terms into normalizing constant c, the equation becomes (2.19), where P_k is the probability of the given hypothesis, P_D is the probability of detection given the prior target presence, λ_f and λ_b are Poisson parameters for false alarms and target births, respectively, and N_τ, N_d, N_f, N_b are number of prior targets, number of successful observation-to-track associations, number of false alarms, and number of new targets, respectively, all at scan k (omitting k subscript for brevity):

$$P_k = \frac{1}{c} P_D^{N_d} \left(1 - P_D\right)^{N_\tau - N_d} \lambda_f^{N_f} \lambda_b^{N_b} \left(\prod_{i=1}^{N_d} \mathcal{N}\left(z_i \,|\, x_i, S_i\right) \right) P_{k-1} \tag{2.19}$$

The Gaussian distribution can be of course replaced by any other probabilistic model. It can be also noticed that although the uniform and Poisson distributions were used in the original paper, the equation can be further generalized such that both λ_f and λ_b can be functions of space and time, combining spatial and temporal density together, but one should be careful about the interpretation of the parameters in this case, because the temporal distribution must remain Poisson, in order to keep the term $e^{-\lambda}$ absorbed in c as a constant. This enables to build less ignorant priors than the uniform distribution, when

possible. Sometimes, target death probability is modeled explicitly in equation as well. More generalized form and rigorous justification on why the formula is actually correct from the Bayesian point of view, is provided in [16].

2.4.2 Track-Oriented Approach

Even when propagating only the k best hypotheses, and effective application of pruning and merging, the number of hypotheses can still grow significantly for large number of targets. An alternative approach is thus to discard global hypotheses from the previous scan and propagate only track hypotheses, on which extensive pruning should be also applied. The hypotheses are reassembled on each scan in order to provide the user with the best one. The track score is maintained for each track, computed by the recursive equation (2.20), borrowing the notation from equation (2.19):

$$L_k = L_{k-1} + \Delta L_k \tag{2.20}$$

$$\Delta L_k = \begin{cases} \ln(1 - P_D) & \text{no update} \\ \ln(P_D \cdot p(z \mid \theta)) - \ln(\lambda_f) & \text{update } \theta \text{ by } z \end{cases} \tag{2.21}$$

Logarithms are used in order to compress the numerical range, as the recursive products may become extremely large or small quite fast. The main trick of the track-oriented MHT is in multiplying the equation (2.19) by λ_f powered to $N_d + N_f + N_b$, which is constant over all global hypotheses at scan k, hence the P_k in equation (2.19) can be factorized into track scores defined above. The following equation (2.22) holds true as a consequence:

$$\ln(P_k) = \ln\left(\lambda_f^{N_d+N_f+N_b}\right) + \sum_{i=1}^{N_\tau} L_k^i = C + \sum_{i=1}^{N_\tau} L_k^i \tag{2.22}$$

The track score L_k is actually a log-likelihood ratio of hypothesis that an observation is assigned to the track against the alternative hypothesis that an observation is a false alarm, hence the numerical value itself can be used for track confirmation or deletion on its own. The sequential probability ratio test (SPRT) [67] can be employed for this task in a more statistically rigorous way, by thresholding the log-likelihood ratio score based on chosen α and β for type 1 and type 2 errors, respectively. Also, the (pseudo) physical dimension is cancelled - the track score is dimensionless. This makes it possible to numerically compare two track scores with different number of observation associations.

When implementing track-oriented MHT, it is convenient to maintain a track tree data structure, called also “family” in the literature [10]. This tree structure explicitly represents different possible track hypotheses for a single target, which means that all the leaves of the tree represent mutually incompatible hypotheses (only one can be “the correct one”). Track trees are build incrementally, on expanding different possible decisions about the observation-to-target associations, and the information provided by them makes the pruning, clustering and other hypothesis management operations very efficient. An example of the track tree structure, or better said, track forest in a general case, is demonstrated in the figure 2.4, where leaf nodes (small boxes) represent track scores for the corresponding track hypotheses, i.e. leaf-to-root paths. Empty nodes represent missing observations and the optimal global solution, maximizing the P_k (2.22), is indicated by the green color.

Beyond naive pruning techniques, such as deleting track hypotheses with scores less then some threshold, it is necessary to employ more advanced pruning methods in order

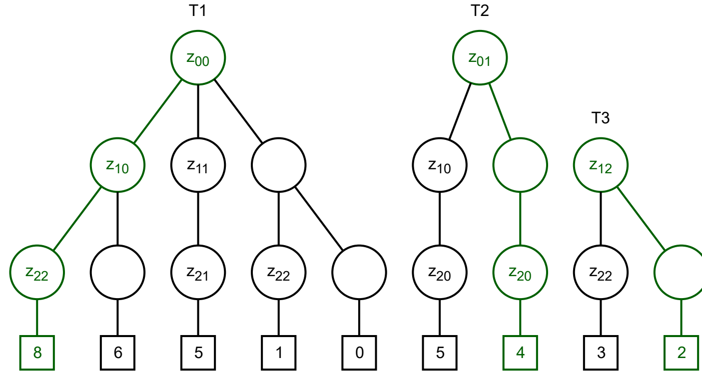


Figure 2.4: Track tree structure example

to prevent the exponential explosion of the trees. Instead of expanding k-best global hypotheses, as it is done for measurement oriented approach (section 2.4.1), a technique called n-scan pruning is performed. Here, the increasing complexity is handled by using a moving scan window (of fixed or adaptive depth), and the decision is made for all the nodes created n-scans in the past. An illustrative example is provided in the figure 2.5, where $n = 2$, i.e. decision is made at the root in this case.

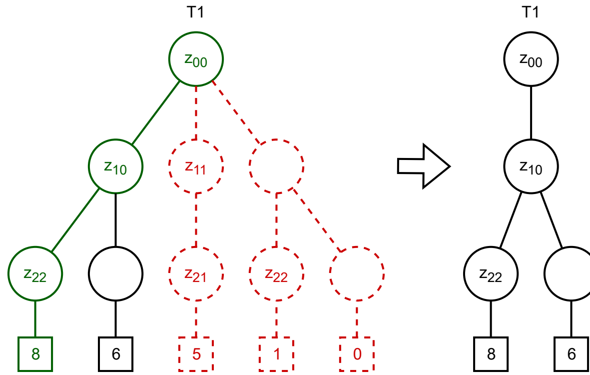


Figure 2.5: n-scan pruning on example from figure 2.4

Because tracks, which are incompatible, i.e. sharing tree or observation, cannot be together in the global hypothesis, it is non-trivial to reconstruct the optimal global hypothesis from track trees. In the illustrative example above (figure 2.4), finding the optimal assignment was relatively simple and could be even done in a brute-force manner. However, as the size of the problem increases (especially in MOT), more efficient methods are required. The problem itself is, unfortunately, NP-hard. Therefore, approximate algorithms has to be used in order to infer the optimal assignment. One such algorithm, which is a relatively classical one in the field, is generalized S-D assignment algorithm [19]. Here, the problem is stated as a linear integer program, where the the optimized criterion is (2.23), where $\rho\left(i_1, i_2, \ldots, i_S\right)$ is an indicator variable, which is 1 when track hypothesis is in the solution, and 0 otherwise, and $c\left(i_1, i_2, \ldots, i_S\right)$ is the negative track score:

$$\min_{\rho(i_1, i_2, \ldots, i_S)} \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_S=1}^{n_S} c\left(i_1, i_2, \ldots, i_S\right) \cdot \rho\left(i_1, i_2, \ldots, i_S\right) \tag{2.23}$$

Constraints are enforced in $\rho(i_1, i_2, \ldots, i_S)$. The main virtue of [19] is using a method known as Lagrangian relaxation in order to approximate the solution iteratively, in polynomial time. For all undecided associations, except the least recent one (i.e. scans at $k, k-1, \ldots, k-n+2$), a vector of Lagrangian multipliers (may be called adders in the log-domain) is used to relax the constraints at each level. On the least recent scan, the classical linear assignment problem is solved exactly, using the upper bound scores resulting from constraint relaxation. Then, the constraints are enforced on the way back to the most recent scan, by sequentially solving $(n-1)$ linear assignment problems, one at each level, as if it was a single hypothesis GNN approach (section 2.3.1). The key difference is, that the Lagrange multipliers change the cost matrices at each level in order to softly enforce the hard constraints. The procedure is repeated iteratively, updating the Lagrange multipliers based on the so-called subgradient, which is a measure of the constraint violation. For example, for 3 observations at scan k, when the 1st one is assigned to 2 targets, the 2nd one is assigned to 1 target, and the 3rd one is not assigned at all, during the relaxation phase, the subgradient vector will be $(-1, 0, 1)$. The algorithm is, however, not guaranteed to converge, and the only guarantee is that the solution will be feasible (no constraint violations).

Another and more modern alternative is to reduce the problem to a maximum weight independent set problem (MWISP) on a graph [66]. Independent set (IS) on a graph is a set of nodes, in which no two nodes are connected by an edge. MWIS is an IS on a graph with weighted nodes, such that the sum of the weights in the IS is maximized. It can also be viewed as a maximum aposteriori (MAP) assignment in a pairwise Markov random field (MRF), which is in fact equivalent to the MWIS in the corresponding graph. The main reason for this reduction is that MWISP is a classical combinatorial optimization problem, and thus much more focus and research has been invested into search for efficient and high quality approximations of this NP-hard task. Finding optimal solution for the tracking problem instance presented in the figure 2.4 is equivalent to finding MWIS in the graph presented in the figure 2.6.

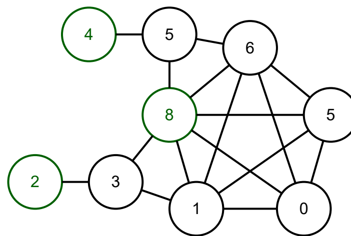


Figure 2.6: Example from figure 2.4 as a pairwise MRF

Experiments in [10] support the advantage of track-oriented MHT over measurement-oriented MHT, in terms of computational and spatial efficiency, while theoretically maintaining all the necessary information for near optimal deferred data association.

Chapter 3

Sensors

In chapter 2, I've described general tracking paradigms and algorithms, but the process of collecting the actual observations was abstracted out, mostly because it is often tightly bound to a particular application. However, the hardware and algorithms used for sensing and object detection, representing a connection of the tracking algorithms to the outer world, have extremely high impact on the overall abilities of the tracking system as a whole. In this chapter, examples of sensors commonly used in practice are briefly described.

3.1 Vision Sensors

Visual perception is often intuitively considered to be the most information rich mean of observation. Vision sensors, such as human visual system or digital camera with artificial intelligence based image processing, are able to perceive a very complex structural information about the world around them. Based on the image data, various properties within the environment, including geometrical, temporal, or physical, such as surface texture and color, can be inferred. While there are many physical properties that are hard or even impossible to be observed by the sight (e.g. sound), and specialized sensors designed to measure some quantities more directly (e.g. radar) can have much higher precision, the overall generality and usual availability of the visual sensors makes them very desirable. Especially in object tracking applications, visual sensors such as digital cameras, thermal (IR) cameras, or medical imaging devices, are often the inputs for the tracking system.

3.1.1 Image Capturing

The first step in the process of machine vision is to capture the physical light and convert it to discrete electrical signal. The result is a digital image. It can be viewed as a matrix of quantized light intensities at different positions within the image plane. These intensities may be scalars in a grayscale case, or vectors in case of a color image, where usually three spectral ranges are being measured as separate color channels, but the number of these color channels is generally unbounded. The conventional technologies are CCD (charge-coupled device) and CMOS (complementary metal oxide semiconductor) sensors[1], which both use the photovoltaic effect to convert the energy of a photon into an electrical charge, which is then converted into a digital signal with analog-digital (AD) converter.

[1] https://www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-camera-sensors-for-machine-vision-applications

3.1.2 Object Detection

When a digital image is captured, it is no more than a 2D signal with no explicit abstract meaning. In order to understand the semantic content of an image data and detect the object(s) of interest, the computer vision algorithms have to be employed. In the algorithms from chapter 4, this step is sometimes partially solved by the tracking algorithm itself, however, in general, the object detector can be viewed as a part of the vision sensor rather than of the tracking algorithm.

Because the topic of object detection in computer vision might deserve its own thesis, I will just briefly review the history and the technology used today for this problem. From its very beginning, the object detection was highly involved in the machine learning. The most successful early approaches include traditional classifiers, such as support vector machines [18], where the histograms of oriented spatial intensity gradients (HoG) of the candidate image patches were extracted and used as a features for classification, which was very successful in e.g. pedestrian detection. Another classical early approach is known as Haar cascade [48], where a hierarchical cascade of AdaBoost classifiers is used to efficiently search the image space with a sliding window. Together with the Haar features, efficiently evaluated using the concept of the integral image, Haar cascade detector is still extensively used today in devices, where the computational resources are limited. Today, the increased computational power and exploitation of the GP-GPU (and dedicated TPUs) parallelism enabled the use of deep convolutional neural networks [20], which led to great improvements in object detection, because robustness against appearance variability in such networks increases with an increasing depth (mostly due to an increasing size of the receptive field).

One of the first deep learning based object detectors, region-based convolutional neural network (R-CNN), was proposed in [31]. The selective search algorithm is used to find the most promising candidate patches, then these are cropped, resized to a fixed size, and classified with a standard CNN classifier one by one. Soon after, fast R-CNN [30] was implemented, where the shared convolutional feature maps are computed only once for the whole image and then, the candidate patches cropped from these feature maps are transformed into a fixed length vectors using a differentiable region-of-interest (RoI) pooling. Faster R-CNN [61] speeds up the detection even more by discarding the selective search part and training the neural network to generate the region proposals as well. Unfortunately, for all the R-CNN-type detectors, realtime performance is almost impossible to achieve even on the high-end hardware.

The breakthrough in the object detection occurred with the advent of single shot detectors, namely YOLO (you only look once) [59] and SSD (single-shot multibox detector) [49]. In YOLO, an image is divided into $S \times S$ grid and for each cell, class probabilities are predicted together with a bounding box in the form (x, y, w, h, c), where c is the confidence (intersection over union with any ground-truth box). Finally, non-maximum suppression is applied to filter out unlikely predictions. Although the intrinsic architecture of the CNN itself used in YOLO is secondary with respect to the core idea, in newer versions, specialized architectures were successfully developed in order to maximize the inference speed (e.g. YOLOv4 [11]). SSD algorithm works in a similar manner and both approaches are considered a standard for the realtime object detection nowadays, as they enable realtime processing even on today's mid-range hardware, such as NVIDIA GTX 1060 graphics card.

3.2 Emitting Sensors

Image capturing devices described in section 3.1.1 passively receive the light, which is either emitted or reflected from the surfaces of objects in the scene. On the other hand, by purposely emitting and receiving the radiation of a specific wavelength and power, not only the ability to sense becomes independent on the external light sources, but because the radiation generation process is in the control of the device, more specific and precise information can be obtained.

3.2.1 Radar

Radar (radio detection and ranging) [64] is an antenna based sensing device, which uses a low frequency bands of radiation (from 0.3 to 100 GHz). Usually, a computer-controlled antenna or an array of antennas is used to transmit and subsequently receive the reflected radio waves in a desired direction. When the radio wave reflects from an object of interest, it also changes its frequency relative, based on the mutual velocity of the object and the radar device. Because the frequency of radio waves is so low, Doppler effect can be exploited to measure this relative velocity directly with high precision. The equation for Doppler frequency shift [64] can be reformulated into expressing this velocity v of the observed object, given the transmission wavelength λ, angle between the antenna direction and the target object movement direction θ, and the measured frequency shift f_d (3.1):

$$f_d = \frac{2v\cos(\theta)}{\lambda} \Leftrightarrow v = \frac{\lambda f_d}{2\cos(\theta)} \tag{3.1}$$

Because θ is generally unknown, only the radial velocity component (i.e. $\cos(\theta) = 1$) can be measured directly using the radar. However, when an object is moving perpendicular to the antenna direction, even with known θ, the velocity cannot be determined at all. To spatially locate the object, frequency modulated waves with periodically linearly varying frequency are used. Because the modulation parameters are known, the distance of the reflecting object can be determined using the frequency shift, assuming the object is stationary. It is possible to measure both speed and location at the same time combining the techniques mentioned above. Radiation can also be emitted periodically in pulses, which is preferred for object detection on long distances (e.g. airport radar).

3.2.2 Lidar

For radar (section 3.2.1), the measurements of velocity are very accurate, but the location is estimated more roughly. The reason is a trade-off between the amount of uncertainty in the frequency domain and the amount of uncertainty in the time domain. In order to have accurate estimate of location in both time and space, high frequency waves have to be used. Lidar (light detection and ranging) [53] presents this kind of high frequency counterpart to a radar, emitting and receiving radiation in frequency bands usually ranging from the infrared, throughout the visible spectrum, and up to the ultraviolet frequency band. Laser beams are emitted towards an area of interest, effectively scanning the distances from the lidar at the reflecting surfaces. A point cloud approximating the surface mesh of the surrounding environment is created and has to be further processed.

Lidar can be useful for measuring exact distances and can provide depth information for images captured by a digital camera, when the relative positions of the two devices are known (e.g. autonomous vehicles). However, although there have been some relatively

successful attempts in the recent years, detecting objects in a point cloud is not an easy task. For example, in the current state-of-the art VoxelNet [70] neural network architecture, the problem is solved similarly as in YOLO from section 3.1.2 for 2D image detection. The space is subdivided into equally spaced voxels and per-voxel point-wise features are aggregated using a fully connected neural network. Then, series of 3D convolution blocks are applied to predict the class probabilities for each voxel, as well as one block of transposed convolutions in order to predict the 3D bounding boxes. While the results are not nearly as good as for 2D image detection, they are still quite impressing given the complexity of the task.

3.3 Collaborative Localization

In some cases, object tracking may be supported by collaboration of the tracked objects. Onboard odometry sensors and positioning systems (GPS) can provide useful and reliable information to the tracking system.

3.3.1 V2X - Vehicle to Everything

Vehicle-to-everything, known as V2X [39], is a general communication standard composed of multiple more specific systems, such as V2V (vehicle-to-vehicle), V2I (vehicle-to-infrastructure), or V2P (vehicle-to-pedestrian). V2X standard provides more information to navigation systems, allowing them to make more informed decision, and increases safety by collecting information from the peer vehicles and other traffic entities. One of potential uses of V2X standard is in realtime traffic analysis, where the collaborating vehicles can provide their onboard information to the tracking system using V2X communication, potentially improving the tracking confidence.

Chapter 4

Visual Trackers

Because of the great potential and extensive use of image capturing sensors in many applications (as mentioned in chapter 3), there is a lot of different algorithms specialized for tracking in the video sequences, and the visual tracking has long been studied in the area of computer vision. These visual trackers basically search for a visual pattern(s) in order to locate the target object. Changes in target appearance due to changes in pose, lighting or shape make the problem of visual tracking challenging. In a multi-target environment, an instance of such tracker has to be created for each target. While this is not usually an issue from a robustness point of view, the computational complexity is unacceptable for most realtime MOT applications (with very few exceptions such as MOSSE tracker in section 4.3.1). Nevertheless, although those algorithms as they are, have low applicability in realtime MOT, they present an important part in the topic of object tracking, and also provide a possible source of inspiration in implementing a multimodal multi-object tracking system.

4.1 Miscellaneous Algorithms

In this section, representative examples of miscellaneous visual tracking algorithms that do not share any significant common trait are provided.

4.1.1 Template Matching

A simple approach to visual pattern tracking is a traditional computer vision algorithm called template matching. Applied to object tracking task, the template is a prior target appearance snapshot. Then, for each input video frame, this template is being moved over the search region (possibly the whole image), and at each position the element-wise signal similarity metric is calculated between the template and the input frame. This metric is usually either cross-correlation, or a sum of absolute differences. The result is a similarity map, where the location of the maxima represents the most likely relative position of the target. This template can be taken from the most recent frame in which the target has been successfully found, or it may be the first occurrence of the target. In the latter case, template matching will likely fail if the target object undergoes deformations. For rigid transformations however, well-known Merlin-Fourier transform may be used. Although template matching in its base form is not very robust, many template adaptation methods were developed, and greatly improved its robustness for specific tracking tasks [45, 69].

4.1.2 Mean Shift

The mean shift is a well-known mathematical method for estimating the mode location of an arbitrary, but unknown probability density function, given only the data sample generated from it. The intuitive idea behind the method is that the density of the sampled points increases towards the mode. The more formal description of the method is slightly more complex, however, the application to object tracking is very straight forward. The target is defined by a window of a fixed size. Visual features of the target are extracted and the probability image is computed from the next input frame based on those features. These features are usually a hue histogram and the probability is evaluated using histogram backprojection. Then, the mean location within the current target window based on the probability image is computed. This can be more clearly expressed by the equation (4.1), where μ is the computed mean location of the target window W, and $P(x)$ is the probability image pixel value at the position x:

$$\mu = \frac{\sum_{x \in W} x P(x)}{\sum_{x \in W} P(x)} \tag{4.1}$$

The window is shifted towards μ, and the process is repeated iteratively until the convergence. When the change from previous μ is less than some threshold (e.g. 1 pixel), the algorithm terminates. In [13], an extension to this idea, called CamShift, was proposed for human face tracking, where the size and orientation of the target window are adapted using the probability image moments. The extra computations are negligible and algorithm is relatively robust to rigid and even non-rigid deformations of the target object.

4.1.3 Optical Flow

One of the most fundamental concepts in computer vision the optical flow. The core idea of the optical flow is that the color intensity I of an object remains constant given sufficiently small change in time, while the position may be subject to change due to the object and/or camera motion. This can be mathematically expressed by the equation (4.2), where v_x and v_y are the respective velocities of the target object relative to the image plane, and t denotes time:

$$\frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t} = 0 \tag{4.2}$$

In this basic formulation, the optical flow may be used to track brightness patterns, but can be generalized for other visual representations (e.g. census transform [34]). However, the equation is underdetermined because it has two unknowns. The optical flow determination is a challenging problem in computer vision, and while many algorithms have been developed, it is still a relatively actual topic. Optical flow can be divided into two categories - dense and sparse. For dense optical flow, a vector field assigning velocity to each pixel is computed. A common approach is Farnebäck's algorithm [23]), but also newer, convolutional neural network based algorithms, have shown quite impressing results [24]. For the sparse optical flow, velocity vectors have to be found only at some provided keypoints (should be corner-like) and the standard way to go in this case is Lucas-Kanade algorithm [50, 63].

Both dense and sparse optical flow can be very useful for object tracking, but the sparse one is preferred because it is computationally cheaper. In [42], a whole object tracking algorithm with the ability to detect its own failure based on the sparse optical flow was proposed - known as the median flow tracker. Given a target window, grid of

points is created. Those points are then tracked with optical flow. Then, they are tracked backward in order to evaluate consistency of the solution - error is the distance between the backtracked points and their prior locations, and median of these errors is found. Worst 50% of points are discarded as outliers. Finally, the box displacement is estimated using remaining points by finding the median of changes for each spatial dimension. The scale change is computed in a similar manner, but this time using the distance ratios between the pairs of points.

4.2 Tracking with Machine Learning

The machine learning has found its applications in a wide range of areas, with computer vision being one of the most successful ones. Following this trend, a whole class of trackers based on machine learning has emerged and turned out to be a promising approach in terms of robustness. In this section, I will mention few interesting examples of such tracking algorithms.

4.2.1 Multiple Instance Learning

While the attempts to use the machine learning algorithms to learn the appearance of the target date almost back to the beginnings of the visual tracking, such supervised learning is more challenging than usual due to the possibly inaccurate estimations of the target locations, causing drifts and failures. Poor quality of the training data impairs the ability of the classifier to separate the positive example from the negative ones (randomly selected patches far from the target location). On the other hand, using multiple patches with slightly perturbed positions around the predicted target location, all labeled as positive, confuses the classifier completely. The multiple instance learning (MIL) tracker [2] is an interesting solution to this problem.

In multiple instance learning, the main idea of the MIL tracker, learning the target appearance is based on the consideration, that, while the individual labels of the candidate patches located around the predicted target location are unknown, at least one of them actually contains the target object to such extent that it can be considered as positive. Thus, rather than assigning labels to the individual samples x_i, the whole bag of samples $X = \{x_1, x_2, \dots, x_n\}$ is assigned a single label. This way, the classification algorithm has the freedom to choose which candidates are good enough, instead of being misled by assigning positive labels to all the candidates, possibly creating contradictory training samples. The learning algorithm used in [2] is called gradient boosting [27], which is a classical machine learning algorithm, where multiple weak classifiers (usually decision trees) are trained and then combined into the final strong classifier. More precisely, in gradient boosting algorithm, a differentiable loss $\mathcal{L}$ has to be defined and then, the classifiers are trained one by one, each to predict the derivative of $\mathcal{L}$ w.r.t. to the output of the previous classifier. The first classifiers output is just a constant, such that the derivative of $\mathcal{L}$ w.r.t. it is zero, i.e. it minimizes $\mathcal{L}$. Also, the contribution of every classifier is weighted by a learning rate η, which lies between 0 and 1 in order to prevent the overfitting. This way, the model is trained in an online manner using the log-likelihood loss, where the likelihood of a label y given the bag of samples X is estimated by the equation (4.3), also called noisy OR, which implements the idea that if at least one x_i has high probability, the whole bag X will also

have high probability:

$$p(y \mid X) = 1 - \prod_i (1 - p(y \mid x_i)) \tag{4.3}$$

Despite using very simple appearance and motion models, the well known Haar wavelet features and a simple spatial distance threshold, the MIL tracker outperformed most of the visual trackers at the time. On the other side, it is poor at detecting its own failure.

4.2.2 Tracking-Learning-Detection

A different approach using the machine learning was proposed in [43] as tracking-learning-detection (TLD). This algorithm is focused on the long-term tracking, trying to solve the problem of temporal invisibility of the target. Many frame-to-frame trackers fail, when occlusions happen, and the re-identification is crucial to overcome such situations. As name suggests, there are two main components in the TLD algorithm - the tracker and the detector. The tracker is able to track in a classical frame-to-frame style, assuming small motions between the frames. The detector is able to search the whole frame and find the target object without any prior spatial knowledge. However, it has to be trained accordingly to the object of interest in order to perform well. The training is done with the support of two experts - the P-expert and the N-expert. The P-expert is basically a frame-to-frame tracker, and if the detector labels the predicted location as negative, it is added as a positive example to the training set. The N-expert assumes, that an object cannot be at multiple locations at the same time. Thus, all patches that do not overlap with the maximally confident patch according to the detector, are added as a negative examples to the training set. In the original paper, scanning window grid and the nearest neighbors classifier with relative normalized cross-correlation patch similarity metric were used, however, the essential idea could be generalized for different model as well. The final object state is assembled from both tracker and detector outputs - if none of them succeed, the object is declared invisible.

4.2.3 Deep Regression Networks

In recent years, the use of deep neural networks in computer vision has recorded a significant increase, and the area of visual tracking is not an exception. Although training those models in an online manner is almost impossible, if the processing speed is taken into account, such models are able to learn semantically rich and generic features offline. These can then be used to track previously unseen objects with a surprisingly good accuracy. In particular, one of the pioneering trackers in this path is GOTURN [36] (generic object tracking using regression networks). The architecture of this deep convolutional neural network [20] is the following. An image of the target from the previous frame and the current frame cropped by the search window are both fed into the series of convolutional layers with shared parameters. These layers capture a high-level features. Then the outputs are concatenated and fed into four fully connected layers in order to regress the final bounding box (x, y, w, h). Although this architecture may seem kind of uninformed in the sense that all the computation including the final bounding box regression is offloaded to the "black-box" neural network, the results are somewhat surprisingly good, especially given the fact that the neural network model is not fine-tuned online. Another but not so surprising observation is, that the performance of GOTURN increases with the size of the training dataset. The network architecture was mostly taken from [38], including hyperparameters,

and trained using the standard stochastic gradient descent algorithm. In order to motivate the network to prefer smaller changes, authors also augmented the training data by small perturbations in position and size, drawn from Laplace distribution (according to their empirical observations of the usual inter-frame displacements). The reported speed of GOTURN algorithm is approximately 100 FPS for a single object tracking on NVIDIA GTX 680 graphics card, thanks to the absence of online learning.

4.3 Correlation Filters

This section is devoted to the correlation filters, a very successful class of visual trackers, known for their great trade-off between speed and robustness.

4.3.1 MOSSE Correlation Filters

In the pioneering paper [12], the idea of an adaptive correlation filter being optimized to the target appearance using minimum output sum of squared errors (MOSSE) criterion was proposed. In the original paper, authors reported a processing speed of up to 669 frames-per-second on a 2.4 GHz 2 core processor, while achieving state-of-the-art robustness at the time, which made this tracker one of the few visual trackers applicable for realtime multi-object tracking. Basic idea is similar to template matching mentioned in section 4.1.1. The key difference is that the cross-correlation is computed in the Fourier domain, and the template is adapted as a filter in Fourier domain as well. The cross-correlation in the Fourier domain is just an element-wise multiplication (cross-correlation theorem), and therefore, the upper bound for the computational complexity is that of computing the Fourier transform, which can be done by efficient algorithms [14] in $O(P \log P)$ time, where P is the number of pixels in the search window. The property resulting from the cross-correlation theorem also enables efficient and simple learning of the correlation filter. The optimal filter is found by minimizing the sum of squared errors against the expected output - a well-defined peak at the true target location. The closed form solution for an optimal filter H^* given N target images is calculated by the equation (4.4), where F_i is the search window spectrum for the frame i, G_i is the corresponding expected peak spectrum, and $*$ denotes the complex conjugate operator:

$$H^* = \frac{\sum_{i=1}^{N} G_i \odot F_i^*}{\sum_{i=1}^{N} F_i \odot F_i^*} \tag{4.4}$$

The formula above gives the same weight to all samples, and therefore is not robust to large changes in scale, rotation, or non-rigid deformations. Therefore, the final algorithm proposed for a robust visual tracking calculates running exponential average, parameterized by a learning rate η, instead of the sums. This increases the robustness against target deformations dramatically. Failure is detected by peak-to-sidelobe-ratio (PSR), which indicates how strong the cross-correlation peak is.

4.3.2 Kernelized Correlation Filters

An alternative interpretation of the idea in [12] is that it is actually a linear regression classifier applied to each possible translation of the target window, efficiently exploiting the Fourier property of the cross-correlation, because the cross-correlation can be viewed as a dot-product at each possible translation, if one flattens the image patches into vectors. The idea proposed in [37] builds on top of this observation and the tremendous training

and inference efficiency of the MOSSE correlation filters, and presents a mathematical foundation for an extension to MOSSE, called kernelized correlation filter, where non-linear kernels may be used in order to perform a non-linear regression, which can have more discriminative power than the linear one. The main contribution of [37] is the exploitation of the classical kernel trick, heavily used in the support vector machine classifiers [18]. The basic idea is that the dot product of two vectors, non-linearly expanded into higher dimensional feature space, can be computed without actually instantiating them in that space. The kernelized filters increase classification capabilities in the same way the non-linear regression classifiers do, while maintaining the speed of linear correlation filter. The mathematical derivations and formulas are quite comprehensive, but the basic idea is similar to that in equation (4.4).

4.3.3 CSR-DCF Tracker

Today, the most complex and best performing correlation filters based tracker, is channel and spatial reliability discriminative correlation filter tracker, known as the CSRT [52]. Since [12], many extensions to the original algorithm were developed. Alternatively to [37], where the filter learning and inference equations were modified directly, the usage of more abstract feature spaces explicitly, while using the original MOSSE equations, became popular. Ranging from traditional visual features, such as histograms of oriented gradients (HoG), color names, or even deep neural network generated features, this simple extension to the algorithm has brought a dramatic increase in robustness, but for a possibly significant computational penalty. The idea was pushed even further, by using more feature channels at once and linearly combining the results efficiently in the Fourier domain as well, thanks to the superposition property of the Fourier transform. Nonetheless, due to different numeric scales of various feature spaces, and also their different discriminative power, using channel specific coefficients in weighted sum projecting this information into the final filter response may lead to better results.

The CSRT provides a robust method for finding these coefficients, called reliability weights. Even with a very simple features (grayscale, HoG, and color names), implementation in [52] achieved a state-of-the-art tracking performance. Another important point of CSRT is the spatial reliability map, which helps to prevent learning the background during the optimization step, which is one of the main problems of correlation filters. The color model for the foreground and background, originally proposed as HSV histograms, is build and incrementally updated over time. The color model is backprojected into the target window and also a spatial prior is constructed. Then, using conditional probabilities (Bayes rule), a Markov random field optimization from [21] is applied in order to minimize the noise in the final posterior probability image, which is then thresholded to obtain the segmentation mask. Filter learning, however, cannot be done using the closed form MOSSE equations anymore. Using Lagrange multipliers, an iterative procedure was designed. While being simple from the implementation perspective, it significantly increases the computational complexity of the algorithm. CSRT therefore owes to its robustness and is not applicable for realtime MOT on a common computer hardware.

Chapter 5

Proposed Tracking System

In this chapter, I discuss the main motivation and inspiration behind the proposed multimodal system for multi-object tracking in realtime. I also provide an overall overview of its design, especially its implementable interface (section 5.2).

5.1 Motivation

As the name suggests, the essential goal of the system is to simultaneously track multiple targets in some volume/area. The number of targets is apriori unknown, thus the system has to be able to determine this number as well. However, the attribute "multimodal" might not have such definite interpretation. Multimodal[1] is a general term that can be used in many contexts, but it can be literally understood as "many ways of doing something". This is exactly the essential principle behind the MOT system proposed in this thesis. The tracking system, which is not limited to a specific sensor, object detector, etc., in order to track objects, but a tracker, that is able to fuse data from different sensors, at different locations, and potentially measuring different properties, i.e. a heterogeneous system of sensors. Various combinations of sensors, statistical and physical models, high-level semantic features extracted from raw measurements, all with one system, being easily extensible, but at the same time, providing explicit mathematical guarantees about the fixed (but configurable) core algorithm. Therefore, one can focus merely on the models, that provide the core algorithm with the observations, all the required statistical properties associated with them, and also, a predictive model. A uniform interface is designed (section 5.2), which when properly implemented, is all that is needed to incorporate a given input device, into the tracking system. I've designed this interface trying to give as much flexibility to the user, as possible, making it possible and relatively easy to implement almost anything, that one could imagine in order to improve the tracking performance in a particular application.

One might naturally ask: "but doesn't this already exist somewhere?". The unfortunate truth is, at least to my best knowledge, that the prevalence of such flexible systems in the software landscape is quite limited. Trackers today are mostly implemented in an application specific manner. Nevertheless, there are few exceptions. Probably the most mature project, inclined in this direction, is a Matlab toolbox[2] for sensor fusion and tracking, which also supports C code generation. It provides a large variety of useful tools and algorithms for multi-object tracking and sensor fusion development for surveillance and autonomous

[1] https://www.dictionary.com/browse/multimodal
[2] https://www.mathworks.com/help/fusion/index.html?s_tid=CRUX_lftnav

systems. For example, a track-oriented MHT described in section 2.4.2 is implemented, using the classical S-D assignment algorithm [19]. After generating C code, it can be used out of the box, e.g. for tracking from radar data. But again, I did not find it as flexible, as I would like to, and also, generating C code from Matlab has its own limitations. Therefore, I've decided to implement the proposed MOT system from the ground up, in the C++ programming language, having the highest degree of flexibility possible, including being able to experiment with more modern algorithms for solving the combinatorial optimization problems, that arise in observation-to-track-association process (section 2.3 and 2.4). I've decided to use the track-oriented MHT paradigm, because it is considered the most general approach and has a unique ability to "recover" from errors potentially made in the past (at least theoretically). Although this ability is strongly reduced by the computational limitations of the current computer hardware, with high quality semantically rich features, such as visual affinity, or long term motion trends, a deferred decision-making based on the future data might solve highly ambiguous situations, in which a single-hypothesis tracker would make a mistake with high probability. Finally, by simply setting the size of a scan window to 1, the multiple-hypothesis pipeline is downgraded into the single-hypothesis one.

5.2 Programmable Interface

In this section, I will present the proposed interface in detail. But first, it may be interesting to mention the main source of conceptual inspiration for the design which was, maybe quite surprisingly, a successful standard API from a different, but not completely unrelated area of computer graphics, OpenGL[3]. In modern OpenGL, a general rendering pipeline is used, with well-defined stages, where only some of them, called shaders, are programmable via dedicated GLSL programming language. The programmer doesn't have to be concerned with essential tasks, such as assembling the geometry, frustum clipping, calling routines for triangle rasterization, etc. For example, in vertex shader, so-called vertex attributes are defined by the programmer, and are used in whatever way the programmer likes to, in order to achieve the desired results. In the system I've developed in this thesis, analogous approach is used, but for multi-object tracking.

5.2.1 Observations, Attributes and Attribute Model

Following the terminology, that I've used in chapter 2, let's denote the collection of observations from one sensor at some sample point in time a "scan". An observation is a collection of attributes, which are obtained by given input sensor. Observation is understood as an implementable interface, but similarly to GLSL vertex shader specification, where `gl_Position` is predefined, because it doesn't make any sense for a vertex of rendered 3D model to have no position, the observation class has this one attribute also present as compulsory, in its base class. It doesn't make any sense to track object position without measuring at least its position (possibly indirectly, of course). Then, for each specific sensor, specializations of attributes, which are provided by that particular device, such as visual descriptor of an image patch in which the object was detected, a direct velocity measurement from radar, or simply a detection confidence. Per observation measurement error statistics can be propagated as attributes as well.

In order to use those attributes to perform the task of object tracking, a model encapsulating the state of a tracked object has to be defined. I call this an attribute model.

[3] https://www.opengl.org/

For example, if one decides to use a Kalman filter for motion filtering and prediction (section 2.2) based on the position attribute, the attribute model will be simply a state of the Kalman filter, i.e. a mean vector x and a covariance matrix P. Of course, the programmer has to implement it (which obviously doesn't take too much work in this case). What the attribute model should be, and how it should be implemented, is merely on the programmer, which makes it very flexible, both from functionality and from computational efficiency perspective. However, the interface requires programmer to implement possibly non-trivial and, in fact, crucial method, being able to evaluate $p\,(z \mid \theta)$ term of the current track score increment ΔL, defined in equation (2.21), where θ are precisely the parameters of an attribute model (x and P in the case of Kalman filter). More detailed explanation is given in the comprehensive overview of the system interface in section 5.2.3.

5.2.2 Sensor Models and Predictive Model

For each sensor device, an implementation of a sensor model interface is required. Sensor model basically initializes and updates the attribute model instances with observations from the given sensor. Those observations must inherit from the base observation class, when it is necessary to extend the attribute set beyond the position attribute. I would recommend as a good practice to strongly tie such specialization to the sensor model (e.g.ås inner class). The sensor model also implements methods to evaluate the prior-to-update terms from the tracking equation (2.19), i.e. prior detection probability given the presence of the target, prior spatio-temporal target birth density, and prior spatio-temporal false alarm density. However, the generalized tracking equation that is actually used, along with detailed description, is provided in section 5.2.3. The sensor model has also the capability of determining the validation gates as well, based on the target state (attribute model), which is contained in the update method itself. The rationale for joining those two steps is that in the conventional pipeline (figure 2.1), the updates (observation-to-track-associations) follow the gating computations immediately, and usually some values computed in the gating step are the very same values that are required later in the update step. Therefore, instead of speculating about caching, the user programmer is completely free to implement both routines in one method as efficiently, as he can do. For example, when using a Kalman filter, Mahalanobis distance from the innovation distribution S, defined in equation (2.4), might be used for gating. It would be quite unfortunate to recompute S^{-1} in the update step again, instead of reusing the already calculated value.

For the motion prediction step, which is considered (and according to my experience really is) crucial for object tracking in general, a prediction model is required. This model is also connected to the attribute model, in the sense that it predicts the future positions based on the current state (e.g. x and P for a Kalman filter). Finally, because the prediction model is a fully implementable interface, as well as the attribute model is, there are no bounds on which attribute values could be predicted and which not. User is completely free to predict anything.

5.2.3 Detailed Overview

After introducing the motivation, basic constructs, ideas and rationales, this final section offers a detailed, but very clear overview of the proposed interface. The already outlined basic pipeline of the proposed tracking system is illustrated in the diagram 5.1. Orange boxes denote the implementable parts of the system (interface), and the blue circle in the middle is fixed, but configurable tracking engine, which is, as mentioned before, a "white

box" system, i.e. it is known how it works inside, what premises it is based on, and has explicit mathematical guarantees about its outputs when the assumption of validity of the implemented models (orange boxes) is satisfied.

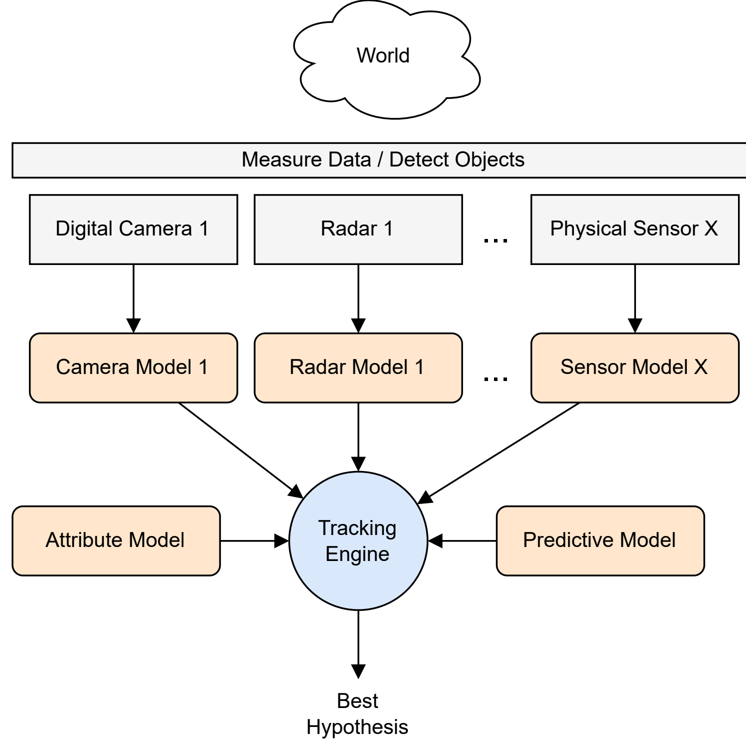


Figure 5.1: Illustration of the proposed system design

Before introducing a complete class diagram of the proposed implementable interface, it is necessary to recall the tracking equation (2.19) from [60]. Following its generalization from [16], the spatial probability density function doesn't have to be uniform. Therefore, terms λ_f and λ_b have to be explicitly multiplied by the corresponding spatial probability density functions $p_f(z_i)$ and $p_b(z_i)$, which should be properly normalized according to the tracking volume/area. However, I do argue, as I've already briefly mentioned in section 2.4.1, that both $p_f(z_i)$ and $p_b(z_i)$ could be absorbed into λ_f and λ_b terms, which reliefs the programmer from any constraints about them at all, except the one that they should preferably lead to the best tracking performance possible. Justification about the theoretical correctness is simply that for any continuous function $\lambda(z)$, there clearly exists a "true" λ' and a "true" properly normalized spatial probability density $p(z)$, such that $\lambda(z) = \lambda' p(z)$. Nevertheless, interpreting the true rate λ' also as a function of space is problematic. From an engineering point of view, however, it doesn't seem too important anyway. Thus, the tracking equation on which the proposed MHT tracker is based on becomes (5.1), using similar notation as in equation (2.19), but with the difference that S_τ, S_d, S_f, S_b are not counts, but the respective sets (z denotes observation, θ denotes state of a given target), and $S_\tau \setminus S_d$ is an abbreviation for $\{\theta \in S_\tau \,|\, *, \theta \notin S_d\}$:

$$P_k = \frac{1}{c}\left[\prod_{\theta \in S_\tau \setminus S_d} (1 - P_D(\theta)) \prod_{z \in S_b} \lambda_b(z) \prod_{z \in S_f} \lambda_f(z) \prod_{z, \theta \in S_d} P_D(z)\, p(z \,|\, \theta)\right] P_{k-1} \quad (5.1)$$

Finally, after using the trick explained in section 2.4.2 and applying logarithms for numerical stability, formula used for track score increment ΔL becomes (5.2):

$$\Delta L = \begin{cases} \ln\left(1 - P_D\left(\theta\right)\right) & \text{missing observation} \\ \ln\left(P_D\left(z\right)\right) - \ln\left(\lambda_f\left(z\right)\right) + \ln\left(p\left(z \mid \theta\right)\right) & \text{update } \theta \text{ by } z \\ \ln\left(P_D\left(z\right)\right) - \ln\left(\lambda_f\left(z\right)\right) + \ln\left(p\left(z \mid \theta\right)\right) + \ln\left(\lambda_b\left(z\right)\right) & \text{initialize } \theta \text{ by } z \end{cases} \tag{5.2}$$

More details about the actual implementation can be found in chapter 6. However, equation (5.2) is required to be known by the user, because he is responsible for evaluating all the terms by implementing the respective models. Now I will connect equation (5.2) with the provided class diagram 5.2.

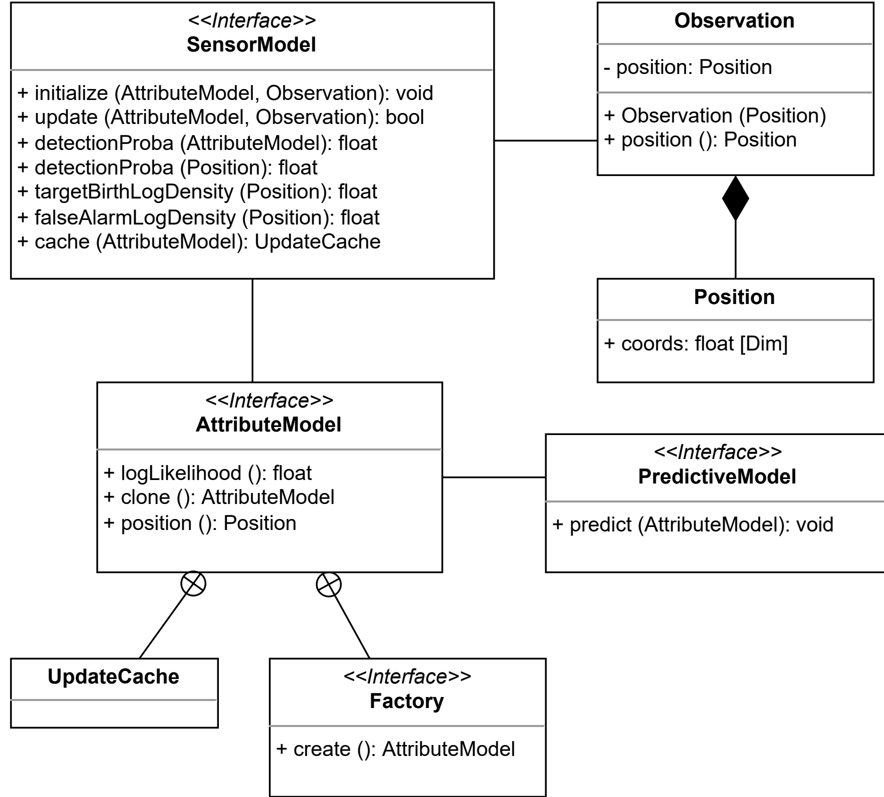


Figure 5.2: Class diagram of the implementable interface

I'll start with the most complex class, which is `SensorModel`. Method `initialize` is supposed to properly initialize an `AttributeModel` with the observation, from which a new target has been internally created. Thus, the method should evaluate $\ln\left(p\left(z \mid \theta\right)\right)$ term in (5.2) in when initializing an `AttributeModel` instance. For example, in case of a Kalman filter, this can be the log-probability density of the measurement noise R (determined by the `SensorModel`) at 0. Method `update` is supposed to properly update the state of an `AttributeModel` with the observation associated to it. Thus, the method has to evaluate $\ln\left(p\left(z \mid \theta\right)\right)$ term in (5.2) for given `AttributeModel`. As I've mentioned in section 5.2.2, it is actually a union of both gating and update step, therefore the method returns a Boolean value. It should return the value of `false`, if the observation falls out of the validation gate. Method `detectionProba` should return the prior probability of detecting a target in state θ, i.e. $P_D\left(\theta\right)$ term in (5.2). It is overloaded for both `Position` and `AttributeModel` as an

argument. The reason for the `Position` overload is that it is very reasonable to condition the value only on the attribute of position, while conditioning it on some other attributes might be disputable. However, the `AttributeModel` overload is kept in order to maintain the generality. An important use case is that an `AttributeModel` might internally use a different coordinate frame than the sensor device. This is especially the case in a multi-sensor environment. On missing observation, when the value of $\ln(1 - P_D(\theta))$ is required, passing the predicted position is ambiguous, since the `SensorModel` doesn't know if it is in the aggregated coordinate frame (e.g. UTM coordinates), or in the sensor coordinate frame (e.g. pixel coordinates in an image). The methods `targetBirthLogDensity` and `falseAlarmLogDensity` both take a `Position` as an input and should return, as their names suggest, $\ln(\lambda_b(z))$ and $\ln(\lambda_f(z))$ terms in (5.2), respectively. Finally, the method `cache` can be used to store any precomputed values for update per each `AttributeModel` instance. `AttributeModel` is, as was said in section 5.2.1, understood as a container for the state. Particular implementation has to be known by all instances of `SensorModel`, in order to properly update it. Methods `clone` and `position` are quite self-describing, but method `logLikelihood` deserves a few words. Although it should be clear that this method represents $\ln(p(z \mid \theta))$ term in (5.2), in fact it can be also $\ln(p(\theta \mid z))$, and the decision is up to the user. The reason why this is valid comes from the Bayes rule and a handy use of dummy tracks and observations. As was mentioned in chapter 2, for consistency, a dummy track representing "exogenous observations" (i.e. new target or false alarm) and a dummy observation representing "missing detection" can be introduced. In this uniform simplified notation, P_k from equation (5.1) can be expressed using equation (5.3), where z, θ denotes all the observation-to-track associations, including the dummy ones:

$$\frac{P_k}{P_{k-1}} \propto \prod_{z,\theta} P(z \mid \theta) = \prod_{z,\theta} \frac{P(\theta \mid z)\, P(z)}{P(\theta)} = \prod_{z,\theta} \frac{P(z)}{P(\theta)} \prod_{z,\theta} P(\theta \mid z) \tag{5.3}$$

From equation (5.3), it is evident that transforming the probabilities in equation (5.1) from $P(z \mid \theta)$ to $P(\theta \mid z)$ and vice versa is simply a multiplication by a constant, because the joint probabilities of all the states θ and observations z at scan k are constant. Another source of ambiguity is, how this quantity should be incorporated into the final value of $\ln(p(z \mid \theta))$ term in (5.2). In case of a motion filter, a probability density is preferred, since it is put against the λ_f hypothesis, which makes the final score numerically interpretable in the way it was described in section 2.4.2. However, when extending the model beyond a simple motion filtering, special care is necessary. For example, when a logistic regression classifier is used to predict the probability of $P(\theta \mid z)$ based on some higher level feature, such as object color, simple addition of $\ln(P(\theta \mid z))$, while being mathematically correct, can change the track score numerical interpretation significantly as it is always negative. Therefore, I recommend to divide the estimated probability by some baseline value, e.g. the one that gives the best classification accuracy in the case of logistic regression example. Then the score increases when the hard decision of the model would be to accept the hypothesis. It is also mathematically correct, following the argument about equation (2.22). `Factory` is an inner class of `AttributeModel` and has a purpose of creating instances of a particular implementation of `AttributeModel`. For `PredictiveModel`, a singleton method `predict` has to be implemented in order to predict the future state of an `AttributeModel` instance based on its current state. For example, in case of a Kalman filter, it is simply an application of the prediction equations (2.1, 2.2). In context of section 5.2.1, I assume `Position`, `Observation` and `UpdateCache` to be sufficiently self-describing.

Chapter 6

Implementation

In chapter 5, I've described the overall design of the implemented tracking system. Motivation, core ideas, chosen paradigm (TOMHT), class diagram of the implementable interface (figure 5.2), and also tracking equation (5.1, 5.2) used in the tracking engine were provided and explained. In this chapter, I describe the implementation details of the tracking engine part (blue circle in figure 5.1). As I've already mentioned, I chose the C++ programming language as a language of implementation. For prototyping during the development, data analysis, and preprocessing data for the tracker, I've used Python programming language, since Python has a very mature ecosystem of machine learning and data science frameworks implementing well-known models and algorithms, and in contrast to C++, the interface is much simpler and requires less work. Thus, for the sake of time, and because the data preprocessing step is not a part of the tracking system proposed in this thesis, I didn't bother with a dull work of rewriting machine learning models from Python frameworks into the C++ ones. For the tracking engine itself, together with the implementable interface, the C++14 standard template library was completely sufficient. For implementing motion filters, I've used a lightweight linear algebra library Eigen[1], which is known for its great performance, enabling many compile time optimizations thanks to an extensive use of the C++ metaprogramming. Finally, for basic image processing, I've used OpenCV[2] (version 3.4) library. For data analysis in Python, I've used NumPy[3], SciPy[4], and Matplotlib[5]. For machine (deep) learning in Python, in order to run pre-trained neural networks that I've used for high-level image processing, I've used frameworks in which particular models were originally written, i.e. PyTorch[6], Caffe, and TensorFlow[7]. For simulations, I've used a Python framework called Stone Soup's[8]. However, the main part of the system, the tracking engine, is implemented in pure C++ (2014 standard). Throughout this chapter, I will describe the data structures and algorithms used in the tracking engine, as well as some implementation specific details. Finally, in section 6.2, I'll provide example implementations of the implementable interface, described in section 5.2.

[1] https://eigen.tuxfamily.org
[2] https://opencv.org
[3] https://numpy.org
[4] https://scipy.org
[5] https://matplotlib.org/
[6] https://pytorch.org
[6] https://caffe.berkeleyvision.org
[7] https://www.tensorflow.org
[8] https://stonesoup.readthedocs.io/en/v0.1b9/

6.1 Tracking Engine

Tracking engine basically implements and extends the track-oriented MHT approach (section 2.4.2). In the main class, named `TrackingEngine`, the tracking itself is performed by two methods - `predict` and `update`. Tracker internally uses a discrete time and every call to `predict` is understood as a "tick". While the real world interpretation of the internal discrete timer of the tracker engine is merely up to the user, it usually makes only sense to understand the tick as a fixed interval in continuous real time axis. Conventional motion models, such as Kalman filter, may, without special care, become mathematically incorrect otherwise. The `update` method is called for each scan at given time k. However, if there are no data scans at time k, `predict` can be called again without any worries.

One of the most important goals of the implemented tracking system is the ability to aggregate data from multiple sensors. Although the MHT paradigm is implicitly accepted as applicable for a multi-sensor environment in the literature [10, 9], I consider it important to discuss explicitly, how exactly this should work, since I don't find it self-evident enough. First, I'll say how it is done from the view of the user programmer. When the number of sensors is more than one, hence there may be more than one data scan available at some time k, `update` method is simply called once for each scan. Moreover, the order in which those updates are performed (e.g. 1, 2, 3 vs. 3, 2, 1) doesn't matter. Now I'll explain why this is correct. Internally, the hypothesis tree can be build almost in the same way, as if the scans arrived consecutively in time. The only modification, that is necessary to make it mathematically correct, is simply to add $\Delta L = \ln\left(1 - P_D\left(\theta\right)\right)$ per each previously processed scan at the current time k. This goes directly from equation (5.2) and the whole aforementioned statement for is illustrated by figure 6.1 for a 3-sensor system example, where for simplicity, only one prior target is considered, and only one observation per scan is received. Observations have now triplet subscript kji, where k is for time, j is for sensor and i is for observation within the scan kj. New target initiated from observation z_{000} is omitted in order to save the space, because new targets from z_{010} and z_{020} are sufficient for the illustration purposes.

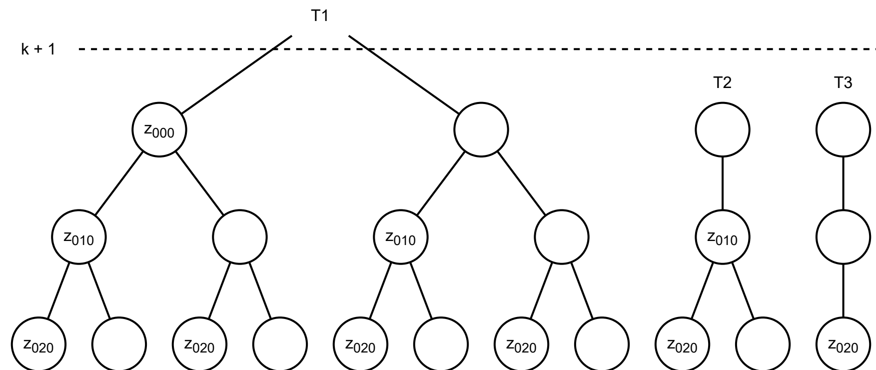


Figure 6.1: Sensor fusion hypothesis trees in 3-sensor system for one time "tick"

Also, in a multi-sensor environment, even more "aggressive" hypothesis pruning may be required in order to keep the computational complexity tracktable. However, it should be clear that there is yet something missing, in order to make this work, since it there are many permutations in which the actual updates of the actual attribute models could be executed. Therefore, it is desired, that individual models used to represent the state of the

target θ, exhibit a statistical independence across the successive updates. In other words, for a sequence of observations $z_1, z_2, \ldots, z_{k-n+1}$, a prior state θ_{k-n}, and a current state θ_k, the following factorization (6.1) must be valid:

$$P\left(\theta_k \mid z_1, z_2, \ldots, z_{k-n+1}, \theta_{k-n}\right) = \prod_{i=1}^{k-n+1} P\left(\theta_k \mid z_i, \theta_{k-n}\right) \tag{6.1}$$

Luckily, this property is exhibited by standard models such as Kalman filter. Even if in some model an auto-correlation occurs, it is unlikely to be significant. Expanding hypothesis for every possible permutation would lead to even greater possibility space explosion and almost certainly would not bring any practical improvement. An overall operation scheme for the methods `predict` and `update` can be seen in the diagram 6.2.

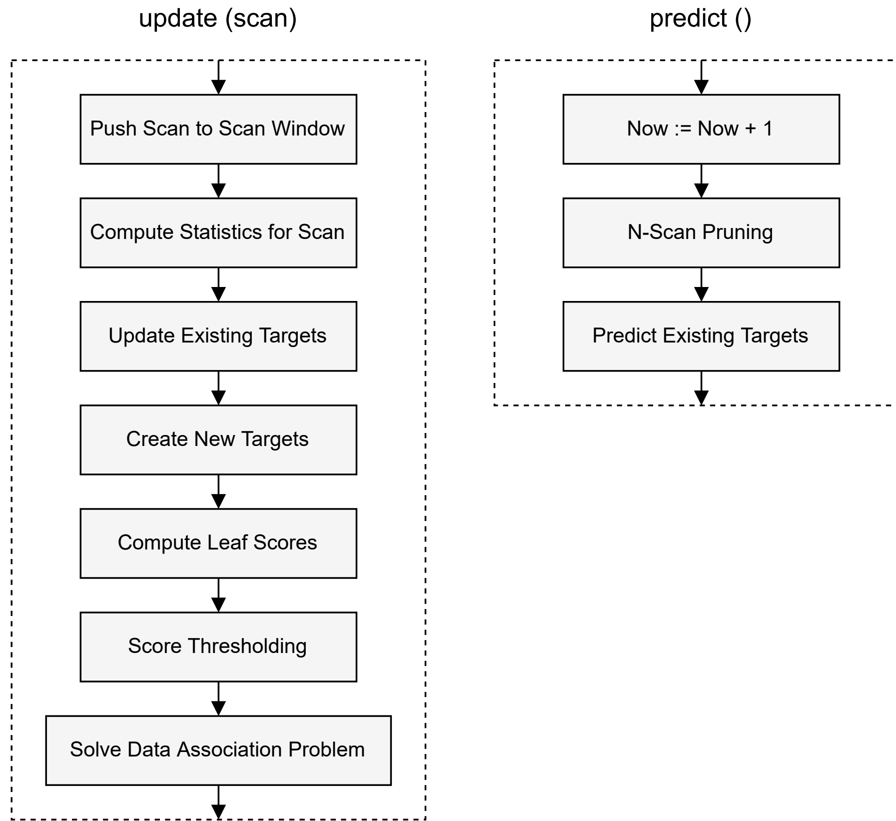


Figure 6.2: Operation pipelines of `update` and `predict`

For `update`, a moving scan window of size defined by user is maintained and each new data scan is "pushed" into is as the most-recent one and when the size is exceeded, the least-recent one is "popped". Thus, the moving scan window is basically a simple FIFO (first-in-first-out) structure. Before the updates, per-observation statistics described in section 5.2 and obtained by interface (diagram 5.2), are precomputed. Any values used for updates are precomputed and stored in `UpdateCache` instances per target hypothesis as well, when this is implemented by the user. Then, every prior target is updated based on the `update` method of the respective `SensorModel` instance. From observations, which are not associated with any target (or the associations are unlikely enough), new target hypotheses

are created, similarly by the method `initialize`. While it is again, not explicitly mentioned in the literature, I find it important to note that the track score L should be bounded by the scan window. Everything before the least-recent scan is considered as "closed", because definite decisions have been made. Due to a recursive nature of the track score, as written in equation (5.2), it is convenient to just keep adding ΔL_k indefinitely, hoping that the use of logarithms will "curb" the numerical range. But an aforementioned fact, that there are no alternative global hypotheses before the least-recent scan, could serve as an argument, why I recognize this to be incorrect to do. Because there is only one global hypothesis prior to the scan window, all global hypotheses within the scan window have, according to equation (5.1), probability $P_k \cdot P_0$, where P_0 is the probability of the singleton global hypothesis prior to the scan window. When implementing the system, I've also empirically found that this way, tracker was prune to becoming highly unstable. Therefore, I've assumed that it is truly necessary to sum the ΔL_k values solely within the moving scan window. In order to preserve the convenience of the recursive formula (5.2), every target in my implementation holds a floating point value, which I called a "root bias", and this value is simply subtracted from the leaf value of L_k. This value, denote it as β, is precisely defined as $\beta = L_{k-n-1}$, where $k - n$ denotes the least-recent scan.

After computing track scores, thresholding follows. Here, track hypotheses with scores below the user specified threshold are simply deleted. Finally, a data association problem is solved, which is discussed in more detail in section 6.1.2. Pipeline for the `predict` method in diagram 6.2 is fairly simple. Increment the internal discrete timer, perform the n-scan pruning routine (if scan window is full), and for each track hypothesis, call the `predict` method of `PredictiveModel` instance (diagram 5.2) associated with the tracker. The n-scan pruning operation is done only in the `predict` method because I've decided to move the scan window only in the time domain, as I consider it more natural. To summarize it all, I provide an illustrative example of a C++ client code for a 2-sensor system (figure 6.3).

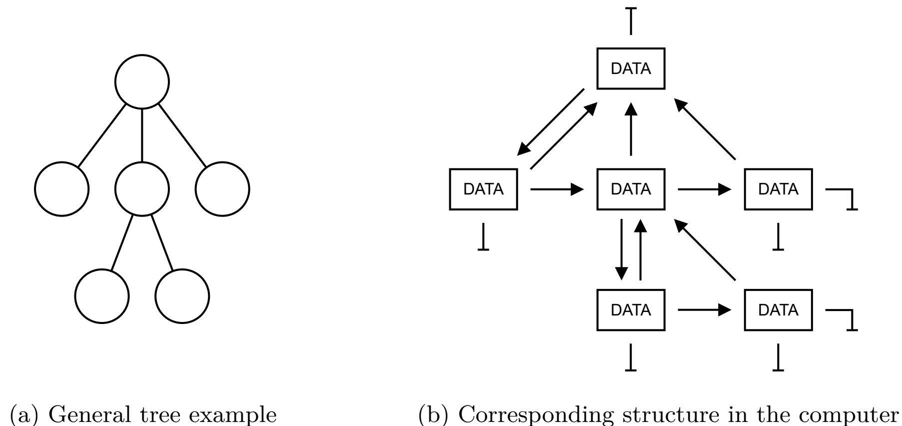
```cpp
TrackingEngine tracker(settings);
tracker.setAttributeModelFactory(factory);
tracker.addSensor("sensor1", sensorModel1);
tracker.addSensor("sensor2", sensorModel2);
tracker.setPredictor(predictor);

while (1) { // main-loop
    tracker.predict();
    std::shared_ptr<const Scan> scan1 = receive1(tracker.now());
    std::shared_ptr<const Scan> scan2 = receive2(tracker.now());
    tracker.update(scan1);
    tracker.update(scan2);
    // process the results here
}
```

Figure 6.3: Example of a client code in C++

6.1.1 Track Tree Data Structure

In the pioneering Reid's paper [60], the tree of global hypotheses was stored as a matrix, where each row represented one global hypothesis. For track-oriented MHT, a similar approach could be made. However, given the dynamic nature of the intensively pruned track trees, and the (at least spatial) inefficiency of storing the whole path back to the root for each leaf as a matrix row, "direct" implementation of the tree structure in a computer is to be considered. By this direct implementation, I mean a general tree structure implemented using e.g. raw memory pointers in C/C++. The only reasonable argument against using such data structure may be the problem of data locality. In matrix implementation, which normally simply an 2D array, data is located in a continuous block of memory. Traversals can leverage huge performance gains from efficiently utilizing the processor cache. However, on modifications, the entire structure has to be copied, which can become very costly. On the other hand, a direct implementation using dynamically allocated nodes and memory pointers to them, can suffer from inefficient utilization of the processor cache, due to data non-locality. Nevertheless, the memory pooling techniques may help significantly. After fair consideration, I've preferred the direct implementation, as I've found it to be encoding the structural relationships between the nodes much better.

When programming trees using nodes and their pointers to link them appropriately, it is very clear how to do so for binary trees. Same goes for their generalizations, i.e. n-ary trees. However, when a general tree has to be implemented, there is a variable number of children nodes, and therefore, children cannot be records in the node structure anymore. A dynamic data structure to store the children must be implemented. A naive way to do so is to use a dynamic array of node pointers. This is, however, not the best way to go. The most efficient way how to store the children nodes is to use a linked list instead. This is the variation which I've implemented. Each node stores a pointer to the parent, a pointer to the first child in the list of its children, and a pointer to its sibling. An illustrative example is given in the figure 6.4, where oriented edges (arrows) in the image on the right represent the memory pointers, and edges with a flat end represent `nullptr`.



(a) General tree example (b) Corresponding structure in the computer

Figure 6.4: General tree implementation

To be able to perform all the required tasks, only three algorithms operating on the track tree structure are required. First, a traversal algorithm has to be implemented. Thanks to the design of the structure (figure 6.4), it is not only possible to perform the depth-first-

search (DFS) traversal through the tree in a non-recursive manner, but it is possible to do so without any auxiliary stack structure. This makes the deletion operations (e.g. in n-scan pruning), as well as collecting the leaves, very efficient. The second algorithm is the detach operation, which simply splits the tree at the given node, removing the node from the list of children of its parent, and setting its parent pointer to `nullptr`. Finally, the third one is a non-default constructor, which takes a parent node pointer as an argument, and properly "splices" the newly created node with the parent tree. The mentioned algorithms are just a dull work with pointers, where a high level of mental concentration of the programmer is occasionally required, but essentially, all three are more or less trivial. Hence, I will skip the details about the exact form of those algorithms.

As I've mentioned earlier, the root bias β, which I've defined in equation (**??**), is maintained for each tree. Also, except formal attributes, such as root pointer and an integer representing the identity of a target, two buffers are used for leave nodes, in order to speed up the tracker in a multi-sensor scenario. Although after n-scan pruning, the changes in the track tree are generally unpredictable, and the leaves have to be reassembled using DFS traversal, expanding new hypotheses as well as simple pruning, in which only single leaf hypothesis is deleted at a time, the changes are tracktable. By keeping the leaves in a buffer between the individual `predict` calls, it is possible to avoid traversing the whole tree in order to collect its current leaves. Adding and removing pointers from a buffer, which I've implemented as a dynamic array, has a constant time complexity, as the removal can be done by swapping the removed value with the last element and just shrinking the array, which is nothing more than decrementing the value of the variable storing its size.

Finally, the node data in the tree is defined as a record of time tick k, at which it was created, an instance of an `AttributeModel` class, double precision floating point score value L_k, hypothesis as an enumerator, and an `AssociationSet` structure. Hypothesis enumerator is currently one of { `Observation`, `Prediction`, `MissingObservation` }, but could be easily extended for more advanced hypotheses. What is important in particular, is the mentioned `AssociationSet` attribute. It is basically a set of node pointers, that are associated with a common observation z, which makes them incompatible with each other and this information is used to construct the data association problem discussed in section 6.1.2. Although for general sets, data structures such as hash table or red-black binary tree are preferred thanks to their favourable asymptotic complexity of search and insertion operations, I've found that for my problem, where the expected sizes of sets are small, i.e. much less than a hundred usually, a dynamic array with naive sequential search performs the best.

6.1.2 Solving Data Association

Finally, hard decisions have to be made at some point, not only to provide the user with the definite result, but to incrementally reduce the space of possible hypotheses. For this purpose, a data association problem must be formally stated. As I've described in section 2.3, this problem can be stated as a combinatorial optimization. Precisely for TOMHT, there are two standard formulations, as explained in section 2.4.2. The first, more traditional one, is a direct optimization throughout the individual scans within the moving window, using the Lagrangian relaxation technique [19]. The second, more modern approach, and the one which I chose, is the reduction of the problem to the maximum weight independent set problem (MWISP), which is actually searching for a maximum aposteriori (MAP) assignment in a pairwise Markov random field (MRF) representing the posterior distribution over

the possible global hypotheses, factorized by the given collection of track trees, according to equations (5.1, 5.2). This dual representation is illustrated by figures 2.4 and 2.6. Because I do not assume the graphical models such as MRF to be a common knowledge, hence I'll provide a brief explanation for the sake of clarity of the rest of this section. More details can be found in the literature [8]. A pairwise Markov random field is a graphical representation of any probability distribution, that can be factorized into the following pattern (6.2), where Φ is called an edge potential and $x_1, x_2, x_3, \ldots$ denote all the nodes:

$$\ln\left(P\left(x_1, x_2, x_3, \ldots\right)\right) = C + \left[\Phi\left(x_1, x_2\right) + \Phi\left(x_1, x_3\right) + \Phi\left(x_2, x_3\right) + \ldots\right] \tag{6.2}$$

Variable nodes sharing an edge are conditionally dependent, since they share a common factor. In case of the probability distribution from track trees, a corresponding pairwise MRF is constructed in the following way. Leaf nodes from all trees, representing individual track hypotheses, are understood as Bernoulli distributed random variables, i.e. taking values either 0 or 1, depending on whether the given track hypothesis is in the global one or not. Nodes corresponding to those hypotheses, which are incompatible (sharing the same tree or a common observation), are connected by an edge. To encode the constraints, the edge potentials are defined as (6.3):

$$\Phi\left(u, v\right) \begin{cases} -\infty & \text{if } u = 1 \wedge v = 1 \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

To incorporate the track scores, an extra variable is defined for each node in the MRF, and is connected to it via an edge with potential defined as the respective track score L . It is evident, that those new variables can be made implicit with the notion of the node potential $\Phi\left(u\right)$, and doesn't have to be explicitly present in the corresponding graph.

To approximately infer the MAP assignment in the MRF, algorithm known as belief-propagation [8], or more precisely its version called max-product, can be utilized. Algorithm is proven to find the exact MAP assignment in polynomial time, when the MRF graph has a tree topology. Otherwise, it is not even guaranteed to converge. However, in practice, it often converges and the results are near-optimal. Inspired by this observation, it was proposed for TOMHT in [66]. Unfortunately, I've found that the lack of convergence guarantees is not only potentially dangerous, but in graphs constructed from complex tracking scenarios, it almost never converged. Inspired by the basic idea of a decentralized message passing algorithm, I've implemented the cavity expansion algorithm [28], which is sometimes referred to as "repaired belief-propagation". Inspired by the (back then) recent work in statistical physics, a notion of cavity, or bonus, was used. In [28], general decision networks with random weights were analyzed. A set of agents taking decisions from some categorical distribution (nodes), set of edges defining the interactions (i.e. statistical dependence) between the agents, and of course node and edge potentials Φ were used to define such general decision networks. Then, instead of trying to compute the optimal solution directly, a cavity B is approximated instead. In this context, B is defined for each variable node u and an action a by equation (6.4), where $x_1, x_2, \ldots$ denote all the other nodes:

$$B\left(u, a\right) = \sum_{x_1, x_2, \ldots} \ln\left(P\left(u = a, x_1, x_2, \ldots\right)\right) - \sum_{x_1, x_2, \ldots} \ln\left(P\left(u = 0, x_1, x_2, \ldots\right)\right) \tag{6.4}$$

After analyzing the computational tree of the max-product, and exhaustive mathematical derivations, a final recursive algorithm was constructed and proven to be always

converging, since the computational tree is finite. Algorithm finds the exact solution, when the recursion depth is unlimited, but the computational complexity is then exponential, as the problem is NP-hard. Moreover, it degrades to max-product when MRF is a tree. The main virtue of [28] is that they proved that when a given decision network exhibits a so-called correlation decay, the contribution of other nodes to the value of B is decreasing exponentially with the number of edges along the mutual path. Therefore, by setting a maximum recursion depth, e.g. 3, a good approximation of B can be obtained in polynomial time. It was proven with a comprehensive analysis that for potentials in MWISP network (and few other classical problems) distributed by normal, exponential, and also uniform probability density functions, the property of correlation decay holds true. For MWISP on the constructed MRF, the pseudocode provided in [28] can be simplified into the final pseudocode 1, which I've implemented in C++. Here, $G \setminus \{u\}$ is an abbreviation for removing a node u from graph G completely, with all the edges in which it was contained. I've implemented the graph as an adjacency list structure and all modifications of it are done using a vector of integers, and by incrementing or decrementing a coefficient at a corresponding index a node is either removed from, or returned to G. When testing my implementation on randomly generated graphs, I've found that maximum recursion depth $r = 3$ gives a good tradeoff between the speed and quality.

```
function cavity_expansion(u, r, G):
    if r = 0 then
        return 0
    end
    G_sub := G \ {u}
    B := Φ(G, u)
    forall v ∈ neighbors(G, u) do
        B_sub := cavity_expansion(v, r − 1, G_sub)
        B := B − max(0, B_sub)
        G_sub := G_sub \ {v}
    end
    return B
```

Algorithm 1: CE algorithm for MWISP

Because the solution is not exact when r is fixed, the cavity $B(u, 1)$ is not guaranteed to be > 0 when u is in the MWIS, and < 0 when it is not. Therefore, I've created a simple algorithm to reconstruct the near-optimal feasible solution based on the fact, that even if the cavity is not greater than zero when u is in the MWIS, it is still greater than the cavity of any v, that is not in the MWIS. The algorithm is defined in the pseudocode 2.

After I've implemented the final algorithm 2 and successfully used it to solve the data association problem arising in TOMHT, I've realized that the cavity expansion 1 (CE) part could be significantly improved in terms of computational complexity, exploiting the information that is contained in the track tree structure itself, hence is freely available as a side effect of its existence. This information is the tree membership of the individual leaf nodes. This information is not being exploited directly be the classical S-D assignment algorithm [19], neither in the MWISP formulation, solved for example by the CE. An alternative type of graphical model which is similar to but more explicit than MRF is known as factor graph [8]. In this model, the factors are expressed explicitly, and the factorization of the given distribution becomes (6.5), using a similar notation as in equation (6.2), but

```
function reconstruct(G, r):
    B := empty map
    forall u ∈ nodes(G) do
        B[u] := cavity_expansion(u, r, G);
    end
    A := ∅
    while ¬ empty(B) do
        u := arg min(B)
        delete B[u]
        A := A ∪ {u}
    end
    return A
```

Algorithm 2: Find MAP in pairwise MRF

$f(\dots)$ now represents the factor potentials explicitly:

$$\ln\left(P\left(x_1, x_2, x_3, \dots\right)\right) = C + \sum_{i=1}^{N} f_i\left(x_1, x_2, x_3, \dots\right) \tag{6.5}$$

Most efficient representation of any Markov random field (including the pairwise one) as a factor graph, is one where the individual factors correspond to the cliques of the minimum clique cover (MCC) of a particular MRF. But since finding this MCC is known to be NP-hard as well, it doesn't seem very helpful. However, track trees are themselves an explicit cliques. Instead of interconnecting all the leaves of a tree with each other, forming $\left(n^2 - n\right)/2$ edges, a single factor representing the tree membership can be defined instead. Even though track trees are obviously not guaranteed in any way to be a part of the MCC of the underlying graph, they are often large. Beyond the theoretical reasoning, I've also found this to be a significant speed-up in practice (up to 2×). An illustration of the different representations of the same probability distribution is shown in the figure 6.5.

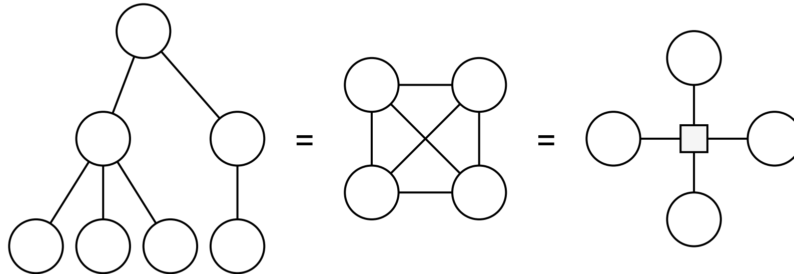


Figure 6.5: Trial representation - track tree, pairwise MRF and factor graph

After modifying the original CE algorithm for MWISP, based on the given observation, I've also empirically verified the implementation by comparing the outputs to the ones from the original CE, successfully. Finally, the CE algorithm, improved for the TOMHT problem is provided in the pseudocode 3. The implementation in C++ is in its nature similar to that of the pseudocode 1. The only difference is that the cliques resulting from tree membership are represented explicitly as lists of nodes, thus the adjacency list contains only inter-tree incompatibilities, and each node has a pointer to the clique of which it is member of.

```
function cavity_expansion(u, r, G):
    if r = 0 then
        return 0
    end
    μ := 0
    forall v ∈ clique(G, u) do
        G_sub := G \ clique(G, v)
        B_sub := cavity_expansion(v, r − 1, G_sub)
        μ := max(μ, B_sub)
    end
    B := Φ(G, u) − μ
    G_sub := G \ ({u} ∪ clique(G, u))
    forall v ∈ neighbors(G, u) \ clique(G, u) do
        B_sub := cavity_expansion(v, r − 1, G_sub)
        B := B − max(0, B_sub)
        G_sub := G_sub \ {v}
    end
    return B
```

Algorithm 3: Modified CE algorithm

In TOMHT algorithms, the size of a moving scan window used for the n-scan pruning routine is usually very small due to an exponential growth of the possibility space [10]. For example, the usual sizes used in radar tracking range from 3 to 5. Although such small windows are still helpful and often sufficient, in some scenarios, especially when strong inter-object occlusions happen, much larger windows may be required to solve an ambiguities that were hard to solve by a simple motion filter, but could be solved by some long-term features, such as appearance descriptor or a frequentist statistical model of some attribute. In order to maintain larger scan windows, able to contain even hypotheses older than a second (e.g. > 25 in case of a sensor with 25 Hz frequency), I've had to employ another pruning strategy. Initially I was thinking about some hierarchical approach, similar to the n-scan pruning in its nature, however, I was not able to construct any reasonable algorithm of that kind. I've finally reached for an inspiration in traditional Murty's k-best algorithm [56], being successfully used in the hypothesis-oriented MHT [17]. I've briefly described the original algorithm in section 2.4.1 as a way of obtaining the k-best solutions to the classical linear assignment problem, in an increasing order of cost. I realized that the set theoretical conclusions about the set of all feasible solutions of a given problem instance could be generalized to a much harder MWISP as well. After carefully modifying the original idea from [56] for my tracking problem, the final pseudocode 4 can be written, with a helper function for the so-called node partitioning. I've also verified the implementation empirically, against the brute-force method on randomly generated graphs. I call the resulting pruning method the n-best pruning. First, user-defined number of best global hypotheses n is obtained using the algorithm from pseudocode 4. Then, the leaves in the corresponding track trees, which are not present in any of those solutions, are simply deleted. This way, the computational complexity is strongly bounded as the maximum number of leaves of each tree is guaranteed not to exceed the user-defined value of n.

The CE algorithm used as a backbone for the combinatorial optimization problem of finding the best global hypothesis (or a set of n-best ones) of observation-to-track associa-

tions within the moving scan window has, beyond the mathematical guarantees about its convergence and the quality of the result, a very attractive property of being completely decentralized. This means that while in traditional approaches, a parallelization of the computations was inherently dependent on the clustering procedure, here the clustering step can even be removed. Therefore, although I've initially implemented this step, I've removed it in the final implementation as a possible source of unnecessary overhead. Maybe it could be a configurable part as well, because while an absence of this decomposition doesn't change the best global hypothesis, a list of the n-best hypotheses is potentially affected. I'll leave that for a possible future work. An important side note about the algorithms 1, 3, 2 and 4, proposed in this section, is that none of them works properly for negative weights. Therefore, it is crucial to transpose the weights adequately in such cases. However, this property becomes very obvious after understanding how the proposed algorithms work.

```
function partition(G, L, A, r):
    L := L \ A
    V := ∅
    forall u ∈ A do
        N := V
        forall v ∈ V do
            N := N ∪ neighbors(v)
        end
        G_sub := G \ (N ∪ {u})
        A_sub := reconstruct(G_sub, r)
        L := L ∪ {V ∪ A_sub}
        V := V ∪ {u}
    end
    return L
function kbest_assignments(G, k, r):
    S := ∅
    A := reconstruct(G, r)
    L := {A}
    for i ← 1 to k do
        L := partition(G, L, A, r)
        if L = ∅ then
            break
        end
        A := arg max (X ∈ L : Σ_{u∈X} Φ(u))
        S := S ∪ {A}
    end
    return S
```

Algorithm 4: Find k-best solutions

6.2 Example Implementations

In the rest of this chapter, I provide the example implementations of the implementable interface (diagram 5.2).

6.2.1 Linear Motion Predictor

First, I've implemented the `AttributeModel` interface as a wrapper for x and P of a simple Kalman filter for 2D position prediction and filtering, named `AttributeModelBasic`. Following this, I've implemented the `PredictiveModel` interface to perform a prediction step of the Kalman filter, named `LinearPredictor`. Model uses equations (2.1, 2.2) and a constant velocity model, as written in equation (2.7), but augmented for the extra dimension. `LinearPredictor` can be initialized directly by setting the process noise variances for position and velocity, using the random acceleration assumption, written in equation (2.8), or both. I've found that even though random acceleration model as a means of dynamic covariance adaptation suffers from hysteresis and introduces extra non-trivial parameters for tuning, a fixed one provides quite reasonable approximation of the underlying dynamics, taking into account a correlation between the position and its derivative. Also, it comes with only single parameter σ_a, instead of the two for position and velocity variances, or even the whole 4×4 matrix Q. `AttributeModelBasic` serves as a base class for all the other implementations of the `AttributeModel`, while `LinearPredictor` is the only implementation of `PredictiveModel`, used in combination with all the other implementations of the proposed interface.

6.2.2 Simple Camera Model and IoU

As the first implementation of the `SensorModel` interface, I've created a `SimpleCamera` class, where the detection probability, the target birth log-density, as well as the false alarm log-density, all used in equation (5.2), are just constant and provided by the user in the constructor. For updating the `AttributeModelBasic` instances, i.e. Kalman filtering step, equations (2.3, 2.4, 2.5, 2.6) are used and the measurement noise standard deviation as well as the prior velocity standard deviation are required in the constructor. The observation log-likelihood used in equation (5.2) is computed simply directly in the log-domain, by the equation (6.6), where $d = z - H \cdot x$ is the innovation:

$$\ln\left(p\left(z \,|\, \theta\right)\right) = -\frac{1}{2}\left[d^{\mathsf{T}} \cdot S^{-1} \cdot d + \ln\left(\det S\right) + 2\ln\left(2\pi\right)\right] \tag{6.6}$$

By implementing the `UpdateCache` inner interface class of the `AttributeModel`, values which do not depend on the input observations z, i.e. $H \cdot x, S^{-1}, K, P$ and the normalizer part of equation (6.6), are precomputed for each target hypothesis before the updates, which eliminates redundant computations in cases when more than one observation falls within the validation gate. The validation gate is implemented as thresholding the Mahalanobis distance term in (6.6) with the 95-percentile location of χ^2-distribution with 2 degrees of freedom. When observations with distance greater than this threshold are discarded, there is only 5% probability that a mistake was made, given the current filter state x and P.

In recent work focussed on realtime MOT in video sequences, e.g. SORT [7], tracking using the bounding boxes retrieved from visual object detectors, such as those described in section 3.1.2, is a well established approach, because those boxes provide useful information about the object size and partially about it shape as well. In [7], a bounding box parameterization using center, area and aspect ratio, is filtered with the standard Kalman filter. While this might seem a little bit "hacky" from a theoretical view, as the area and especially the aspect ratio obviously do not follow the assumptions made by the standard Kalman filter, in practice the filtering works quite well for pedestrian tracking. The problem is with the statistical interpretation, because although the filtering capability of the Kalman

filter depends solely on the ratio of the covariances (as I've mentioned in section 2.2.1), the actual distributions represented by x and P can be incorrect. Therefore, for evaluating the possible observation-to-track associations, the intersection-over-union (IoU) metric is used. This metric can be defined by equation (6.7), where A and B are axis-aligned bounding boxes and $A \cap B$ is again an axis-aligned bounding box and can be easily obtained using max and min operators:

$$\text{IoU}(A, B) = \frac{\text{area}(A \cap B)}{\text{area}(A) + \text{area}(B) - \text{area}(A \cap B)} \tag{6.7}$$

In SORT-like-trackers [7], single hypothesis paradigm is employed and the optimal assignment is found using the classical Kuhn-Munkres algorithm [47] on the cost matrix build from IoU scores between all the possible pairs of predicted boxes and detection boxes (observations). Gating is done by thresholding with a user defined value. The statistical interpretation of the aforementioned procedure is that the optimal assignment is actually a MLE of the respective joint distribution, where the IoU scores are distributed according to the exponential distribution. Because the rate parameter λ performs only scaling, it is actually assuming the family of all the possible exponential distributions. Because this is an uninformed assumption and the thresholding parameter is usually chosen with an "eye of an expert" rather than based on some data analysis, I've decided to build the model extending the `SimpleCamera` and `AttributeModelBasic` classes with the IoU metric in a more rigorous way. I've analyzed the MOT Challenge[9] public dataset, on which I've later performed few evaluations, discussed in chapter 7. However, in the analysis, I've tried to find an auto-distribution of the IoU metric as well as some baseline distribution. By auto-distribution in this context I mean a distribution of IoU between the detection at time k and the previous detection at time $k-\Delta$, translated according to the inertial from a motion filter. The "lag" factor Δ was set to 10 and the inertial was computed based on the simplest motion filter possible - a difference between position at $k - \Delta$ and $k - 2\Delta$. This way, the estimate should be an upper bound and should work well with any motion filter which is at least as good as the simple difference based one. For the baseline distribution, i.e. alternative hypothesis H_1, I chose a distribution over IoU with randomly selected detections with $\text{IoU} > 0$. After plotting the results, a clear separation is visible and also, the IoU auto-distribution, representing $p(z \mid \theta)$ in equation (5.2), is not exponential but actually a Γ-distribution. After obtaining the MLE, the estimated Γ-distribution fits almost perfectly the observed data. Results are shown in the figure 6.6. The final equation for updating the track score L based on the IoU is (6.8), where α, β, Δ are estimated parameters of the Γ-distribution, and $p(H_1)$ represents the baseline IoU distribution:

$$\ln(p(z \mid \theta)) = \ln(1 - \text{IoU} + \Delta)(\alpha - 1) - \beta(1 - \text{IoU} + \Delta) - \ln(p(H_1)) \tag{6.8}$$

Based on the figure (6.6), IoU threshold 1/2 was chosen. It is good to mention, that while the results from figure 6.6 are correct given the data, one cannot truly assign close to zero likelihood to the event of $\text{IoU} = 1$, since it is actually possible due to rounding errors, even though never seen in the analyzed dataset. Therefore, if IoU is greater than the mode location of the estimated Γ-distribution, the resulting log-likelihood ratio score is clipped to be strictly non-negative, preventing the score decrements caused by prior likelihood being close to zero. Also, the empirical baseline distribution happens to be zero for $\text{IoU} > 1/2$ as well, which obviously cannot be assumed. Therefore, I've set the $p(H_1)$ to be 5% of the mode density of the estimated Γ-distribution to reflect the prior uncertainty.

[9] https://motchallenge.net/

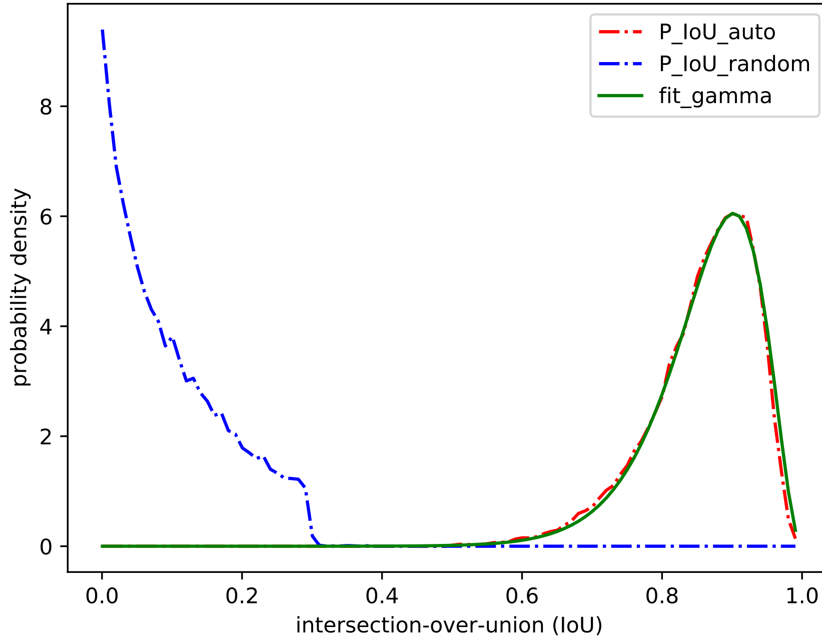


Figure 6.6: IoU distributions - baseline (blue), auto (red), and Γ fit for auto (green)

6.2.3 Multi-Camera Model and Appearance Embeddings

Finally, I've implemented a system capable of multi-object tracking in a multi-sensor environment. Again, the whole implementation is as simple as just implementing the respective interfaces, since the tracking engine proposed in chapter 5 is basically a library and can be used for any system of sensors. Because the only suitable real-world data for multi-sensor tracking I was able to acquire was from a multi-camera road junction monitoring system (results are discussed later in chapter 7), my implementation is just a homogeneous system, despite the name of this thesis. Creating a heterogeneous one is, however, in principle the same, except the necessity to implement the models for other types of sensors capable to map the observations to some common aggregate space and vice versa.

In this implementation, I've decided to geo-register the cameras as they are static, with only small motions caused e.g. by wind. The geo-registration was done by manually annotating appropriate points with respective world geodetic system (WGS84) coordinates (which can be found by online tool such as Google Earth[10]). Then, because the tracking area is sufficiently flat, a simplified model known as planar homography can be employed. First, the WGS84 coordinates are converted to universal transverse mercator (UTM) coordinate system, which locally approximates the earth surface by small planar facets resulting from a cylindrical projection of the geoid[11]. For this task, I've used a C++ implementation[12] available online. One potentially significant reason for preferring the geo-registration approach to creating a common aggregate space, as well as any other approach mapping the measurement space to the physical space (or its subspace in case of the planar homography

[10] https://earth.google.com/web
[11] mathematical approximation of the planet Earth
[12] https://github.com/ethz-asl/fw_qgc

from UTM) rather than some arbitrary one, is the invariance of the motion model parameters to the sensor location, pose and resolution. Also, this makes the simple models such as constant velocity model with random acceleration assumption (section 2.2.1) to perform much better, because the projection to the measurement space of a particular sensor, as a one of the main sources of the motion non-linearity, is removed.

The first step is again, implementing the `SensorModel` interface. For all the other specializations of multi-camera systems, potentially using higher level features, I've created the `MultiCamera` class. For motion prediction, simple `LinearPredictor` described in section 6.2.2 is used. However, the motion filtering step must be done using an extended Kalman filter, because of the non-linear projection from process space (UTM) to the measurement space (screen coordinates). While it is absolutely possible to use a linear mapping and just preprocess the input positions by transforming them to UTM beforehand, I've found this to be an extremely rough method and setting the measurement noise covariance matrix R to some fixed value in metres is very problematic, as it is non-trivial to map a (partially) known error covariance in the image plane to the real-world. It seems that the best way to do so is actually to project the real-world position to the image plane coordinates using the homography and then just use this homography locally linearized around the current estimate to compute the Kalman gain K. The innovation covariance S then reflects the non-linear effects of the perspective. Before I'll dive into the particular implementation of EKF for this task, it is important (and later useful) to write the projection equation of the well-known pinhole camera model (6.9), where K and M are known as intrinsic and extrinsic camera matrix, respectively:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = K \cdot M \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.9}$$

The non-linear effects of perspective occur after dividing the result by z'. However, in case of the UTM plane, the aforementioned planar homography directly follows from the equation (6.9), when z in real-world 3D coordinate space is set to 0. The planar homography can thus be a full-rank matrix of shape 3×3, which makes the whole projection very tractable since the inverse projection can be found analytically. Planar homography has 8 degrees of freedom, hence can be found exactly from 4 point pairs using direct linear transformation (DLT), which basically "unrolls" the corresponding individual projections into single system of 8 linear equations. This can be solved by standard methods. Also, the more point pairs available, the better, as the criterion of minimum of the squared residuals can be optimized. For finding the homography matrix P, I've used the OpenCV implementation. Now, after writing the linear projection followed by so-called "perspective divide" operation in order to get the final screen position in the form $\begin{bmatrix} x & z & 1 \end{bmatrix}$, a system of non-linear equations can be written. As mentioned in section 2.2.1, the EKF requires Jacobians of the non-linear mappings evaluated at the current filter estimate x. After a certain amount of mathematical analysis, partial derivatives for the Jacobian of $h(x)$ can be evaluated at x using equations (6.10, 6.11, 6.12, 6.13):

$$\partial h_1 / \partial x_1 = \alpha \left[(P_{11} P_{32} - P_{12} P_{31}) \, x_2 + (P_{11} P_{33} - P_{13} P_{31}) \right] \tag{6.10}$$

$$\partial h_1 / \partial x_2 = \alpha \left[(P_{12} P_{31} - P_{11} P_{32}) \, x_1 + (P_{12} P_{33} - P_{13} P_{32}) \right] \tag{6.11}$$

$$\partial h_2 / \partial x_1 = \alpha \left[(P_{21} P_{32} - P_{22} P_{31}) \, x_2 + (P_{21} P_{33} - P_{23} P_{31}) \right] \tag{6.12}$$

$$\partial h_2 / \partial x_2 = \alpha \left[(P_{22} P_{31} - P_{21} P_{32}) \, x_1 + (P_{22} P_{33} - P_{23} P_{32}) \right] \tag{6.13}$$

The coefficient α is shared between the equations above because it is a result of the perspective division, and is simply (6.14):

$$\alpha = (P_{31}x_1 + P_{32}x_2 + P_{33})^{-2} \tag{6.14}$$

To clarify the mathematical notation used in the equations above, it has to be explicitly stated that the Kalman filter state $\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$ in the context of equations (6.10, 6.11, 6.12, 6.13) is equivalent to $\begin{bmatrix} x & y & v_x & v_y \end{bmatrix}$ in the context of pinhole camera model equation (6.9), where v_x and v_y are the respective velocities. When implementing this in C++, it is good to note that most of the computations defined in equations (6.10, 6.11, 6.12, 6.13) can be pre-computed.

Another usage of the planar homography is to determine, if it is even possible to see the target object from a given camera, i.e. if it falls within its frustum[13]. When it doesn't, the `detectionProba` method of `SensorModel` instance should return 0, instead of absurdly penalizing the track hypothesis score. A naive approach would be to just project the predicted position in UTM to image plane and check, if it is within the image boundaries. But there is a catch. Because the planar homography has one degree of freedom less than the number of coefficients in the solution matrix P, all matrices in the form γP represent the same homography. When an object is behind the camera, its position will be projected within the image boundaries after the perspective divide. Therefore, I've implemented a decomposition of the homography matrix P in order to overcome this problem. As a bonus feature, physical 3D position of the camera is obtained, which may be exploited by some more advanced implementations. The decomposition is fairly simple and follows directly from equation (6.9). The rotation matrix R can be reconstructed using equations (6.15, 6.16), where P_i denotes i-th column of the homography matrix P:

$$r_1 = \frac{K^{-1} \cdot P_1}{||K^{-1} \cdot P_1||},\ r_2 = \frac{K^{-1} \cdot P_2}{||K^{-1} \cdot P_2||} \tag{6.15}$$

$$R = \begin{bmatrix} r_1 & r_2 & (r_1 \times r_2) \end{bmatrix} \tag{6.16}$$

The aforementioned issue that all matrices in the form γP represent the same transformation, is partially solved. Now, γ is either 1 or -1. However, to be able to tell if the object is behind the camera or not, the sign of γ must be known exactly. To find it, a linear system $R\,|\,T$ has to be solved, where T is the translation vector from equation (6.9) and in this case, it is equal to properly normalized 3-rd column of P. By properly I mean according to the denominators in equation (6.15), which should both be the same, yet there may be some small rounding error. The solution is an absolute real-world 3D position of the camera, except that the z coordinate may be negative. Assuming that all cameras are above the ground, the sign of γ is now the sign of the z coordinate. Of course, a similar procedure could be done if it was assumed, that all cameras are under the ground, no matter how absurd it may sound. It is necessary to mention, that the pinhole camera model is not correct for physical cameras based on optical lens, since those lenses introduce radial and tangential distortions. Those distortions are usually modeled using up to 14 coefficient, which have to be obtained by calibrating the camera. I've used the algorithms implemented in OpenCV to obtain the undistorted images, in order to make the pinhole camera model and geo-registration applicable.

Although using the detection boxes centers as positions when tracking in video sequences is a common practice and works well for single camera, when trying to track in an aggregate

[13] 3D region which is visible on the screen

space from multiple cameras bound by planar homography, large errors (in units of meters) do occur. This is because the planar homography projects the world plane (UTM in this case) to the image plane and vice versa, assuming the object center being at $z = 0$, which is obviously not correct, especially for taller objects. An illustration of this problem is provided in figure 6.7.

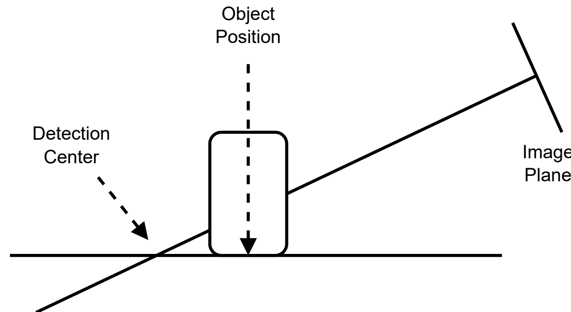


Figure 6.7: Ground position problem with planar homography

It is a highly non-trivial to reconstruct the 3D structure of an object based on its 2D projection in an image plane, and generally requires some kind of "visual intelligence". To make this example implementation work, I've used a deep convolutional neural network trained for the task of vehicle center ground position prediction. Because this task is worth of a half of a Masters's thesis itself, I've used an existing, already trained model written in the Caffe framework. The model I've used is proprietary, however, I was allowed to run the inference on my data. The network was trained on a synthetic dataset obtained from Grand Theft Auto V computer game using ScriptHook plugin[14]. The prediction is done by first resizing the detection patch using a "letterbox" transform with side length of 224 pixels. Then this input image is passed to the standard ResNet-18 [35] architecture front-end, and finally, the terminal fully connected layers perform binary classification per pixel (corresponds to 16×16 pixel grid in the input image). The pixel with the highest logit score is chosen as the center ground position. Also, a real numbered Δ vector is predicted as well, in order to refine the position rounded due to the pooling operations.

In relatively recent works on video tracking, an extensive usage of image descriptors inferred by deep learning models, often called "embeddings" in this context, has harvested a significant success, e.g. in [68]. When the spatio-temporal information, or the way in which it is modeled, is not reliable enough, exploiting rich visual information can increase the stability of the tracking algorithms and reduce the number of identity switches. Therefore, I've implemented also the `MultiCameraAppearance` class, which inherits from the `MultiCamera` class described above, and extends it with an appearance embedding model. The backend neural network I've used is from [46], trained on a proprietary dataset. While having again the standard ResNet-18 [35] architecture front-end, the training procedure described in [46] differs from the conventional approach, when the optimization is done based on some contrastive loss function, e.g. the well-known triplet-margin loss. Here, a large model with 512-dimensional embeddings is trained in a standard way first. Then, a new model with highly reduced dimensionality compared to the large one (512 to 64) is trained based on the large model and the so-called relaxed contrastive loss, which is described in [46] in more detail. Because the Euclidean distance was minimized during

[14] https://github.com/crosire/scripthookvdotnet

the training, I've had to find a probabilistic expression which would enable to incorporate it to the tracking score equation (5.2). First, the tracker was run on videos from multi-camera road junction monitoring system, described in chapter 7, without the appearance embeddings. However, the appearance embeddings were collected from tracks, and for each target, a mean normalized appearance vector was computed. Empirical distributions of the squared Euclidean distances are shown in figure 6.8, where the red one can be interpreted as "positives", i.e. distances of embeddings from the mean embedding of the respective target, and the blue one can be interpreted as "negatives", i.e. distances of embeddings from the mean embeddings of other targets. Using the Bayes rule, a conditional probability $P(\theta \mid z)$ can be computed from the estimated distributions, which can then be directly used in formula (5.2). In order to analytically approximate the distribution I've empirically found, a second order polynomial in the logit space was used. The empirical distribution (red) as well as the final regression (green) are shown in figure 6.9. Final formula used by the `MultiCameraAppearance` class is (6.17, 6.18), where μ is mean embedding of the target, maintained via exponential moving average (EMA), and a, b, c are estimated from the data:

$$P(\theta \mid z) = -\ln\left(1 + \exp\left(-aL_2^4 - bL_2^2 - c\right)\right) + \ln(1/2) \tag{6.17}$$

$$L_2^2 = (\mu - z)^{\mathsf{T}} \cdot (\mu - z) \tag{6.18}$$

Mean embeddings μ are stored as 64-dimensional vectors of 32-bit floats in the respective instances of `AttributeModelAppearance`, which inherits from `AttributeModelBasic`.

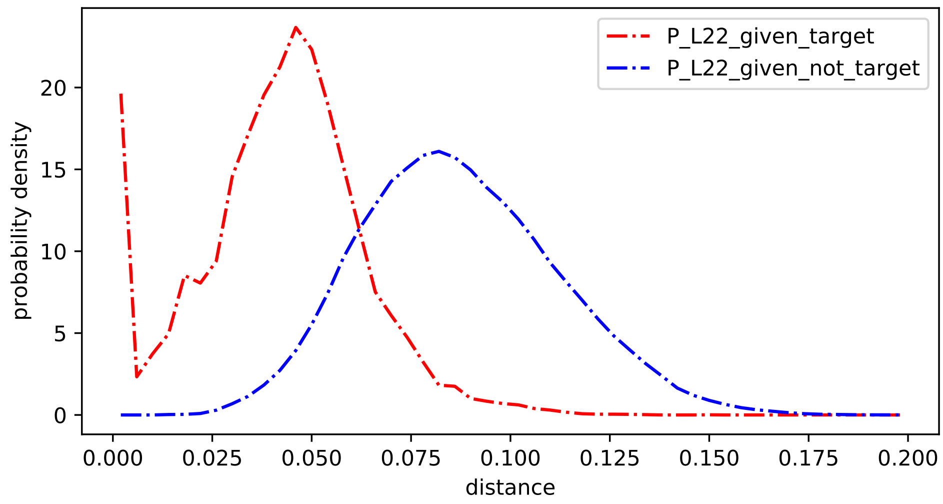


Figure 6.8: Appearance embeddings distance distributions

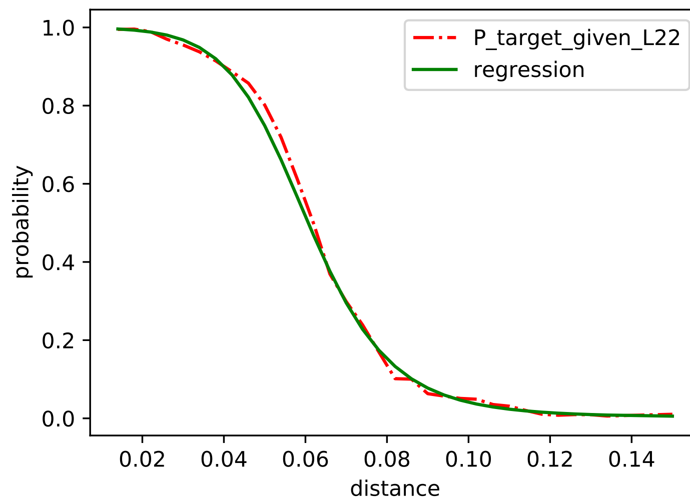


Figure 6.9: Conditional probability $P(\theta \mid z)$ used in appearance based track scoring

Chapter 7

Evaluation

In this chapter, I provide a brief evaluation of the implemented system(s).

7.1 Simulation

First, I've tested the implementation on simulated 2D MOT scenarios. The main advantage of using a simulated dataset is that the statistical parameters are known exactly, as well as the ground-truth trajectories. This is an idealistic situation, when the only sources of error should be either the real statistical limitations resulting from the uncertainty inherent to the observations, or the imperfections of the core tracking algorithm itself, i.e. the one which solves the combinatorial optimizations that arise in MOT. In my case, this is the `TrackingEngine` class with all its components. The evaluation of the tracking performance, especially in MOT, is a hard problem itself and is still an open debate, how to compress this fairly complex information into a single scalar value, which will in return accurately reflect how "good" the tracker is. One of the traditional metrics for evaluating the performance of multi-object trackers is known as optimal subpattern assignment (OSPA) [62]. This metric operates on the finite sets of tracks and known ground-truths and evaluates two error components - localization and cardinality error. Localization error reflects how bad the tracker is in estimating the position of the truth target. Cardinality error on the other hand, accounts for the target misses and false targets. However, in a relatively recent work, generalized OSPA [58] was proposed, and I've decided to use it instead, since it was presented as a better indicator than the original OSPA, while being based on similar ideas. Detailed mathematical content of [58] is slightly comprehensive, but the GOSPA formula, simplified by plugging in the parameters which I've used, can be written as an optimization problem (7.1), where X is the set of truths, Y is the set of tracks (output from the tracker), Γ is the set of all possible track-to-truth assignments, and $d(\dots)$ is an Euclidean distance:

$$\min_{\gamma\in\Gamma}\left[\sum_{i,j\in\gamma} d\left(x_i, y_i\right) + \frac{1}{2}\left(|X| + |Y| - 2|\gamma|\right)\right] \tag{7.1}$$

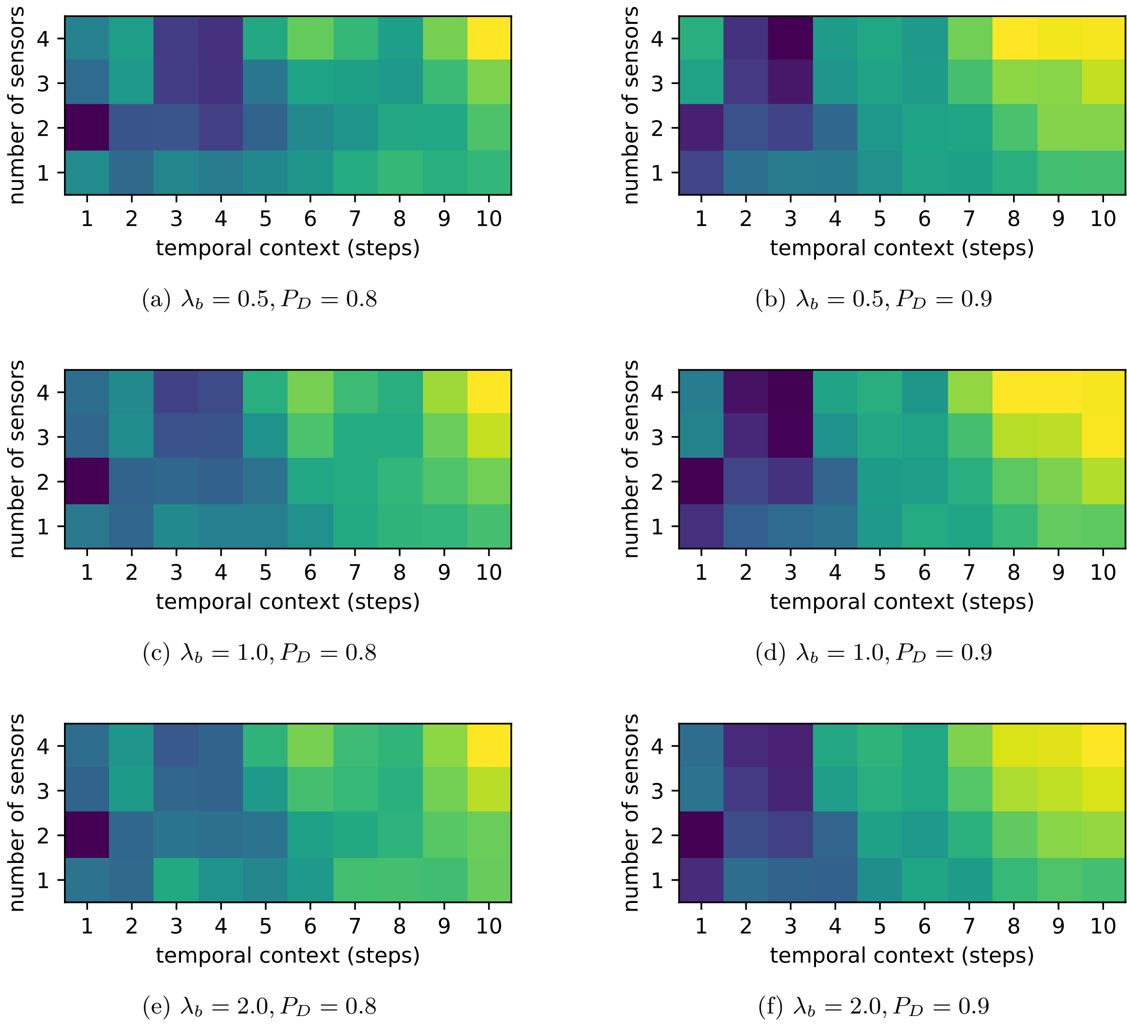
As I've mentioned in the beginning of chapter 6, I've used a Python framework named Stone Soup's for simulation generation and evaluation. Because there is no reason, why the tracker would perform worse than any other naive implementation of the standard single hypothesis MOT pipeline in such simulated scenarios with perfect information, I'll omit any comparisons here. In order to ground the performance of the proposed system, I provide

comparison with SORT tracker [7] on a real-world dataset in section 7.2.1. What I was interested in here, however, is how an increasing depth of the track-trees, i.e. the temporal context in which the decisions are deferred, and an increasing number of sensors impact the final GOSPA distances relative to the naive single-sensor single-hypothesis case. I've performed the following simulations. In all scenarios, the size of the tracking volume was set to 1000×1000 units, time duration was set to 400 steps, measurement noise variance was set to 1, variance σ_a^2 for the random acceleration model, as defined in equation (2.8), and the prior velocity variance, were both set to 0.09, false alarm rate was set to 0.0001 and death probability was set to 0.01. Then, the simulated ground-truths were generated from the Cartesian product of the following two sets of parameter values for birth rate λ_b and detection probability P_D (i.e. 12 simulated ground-truths):

- $\lambda_b \in \{0.5, 1.0, 2.0\}$
- $P_D \in \{0.6, 0.7, 0.8, 0.9\}$

On these ground-truth simulations, tracker was run with various settings. From all 12 ground-truths, 4 sets of detections were simulated based on the P_D parameter. Then, the tracking was performed using only one sensor, two of them, three, and finally all four. Also, the size of the tracking window (in the time domain) was progressively increased from 1 up to 10. In all settings, the maximum number of hypotheses per-target was capped to 20. Based on the death probability it can be noticed, that the average numbers of targets concurrently present in the scene for different values of $\lambda_b \in \{0.5, 1.0, 2.0\}$ are $\{50, 100, 200\}$.

After running all the experiments described above, I've plotted the respective GOSPA distances as a function of temporal context size (horizontal axis) and a number of sensors (vertical axis). To visualize the numerical values of the GOSPA distances, a color bar ranging from dark blue (higher GOSPA, worse performance) to bright yellow (lower GOSPA, better performance) is used. The results are shown in figure 7.1, but only those for $P_D \in \{0.8, 0.9\}$ are provided, because lower detection probabilities led to very poor performance in general and the GOSPA distances were very noisy. On the other hand, when $P_D \geq 0.8$, it seems that there is a consistent trend of increasing performance with increasing both the temporal context size and the number of sensors. This trend was expected and seems to be stronger, when λ_b is higher, which was also expected. Low target birth rate doesn't introduce as many data association conflicts as the high one does. Therefore, even a single-hypothesis single-sensor tracker performs very well and there is almost zero added value in using more sensors or building deeper hypothesis trees. Also, a substantial decrease in tracks-to-targets ratio (TTR) with increasing temporal context size can be noticed in figure 7.2, where TTR is marginalized over all experiments. This indicates a reduction in track fragmentation when the hard decisions are deferred for longer and more candidate hypotheses are expanded.



(a) $\lambda_b = 0.5, P_D = 0.8$

(b) $\lambda_b = 0.5, P_D = 0.9$

(c) $\lambda_b = 1.0, P_D = 0.8$

(d) $\lambda_b = 1.0, P_D = 0.9$

(e) $\lambda_b = 2.0, P_D = 0.8$

(f) $\lambda_b = 2.0, P_D = 0.9$

Figure 7.1: GOSPA scores from the simulations

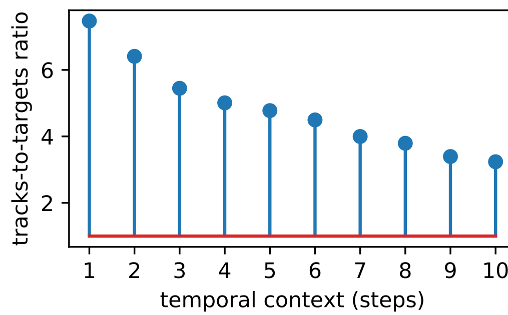


Figure 7.2: Decreasing tracks-to-targets ratio

7.2 Real-World Scenarios

7.2.1 MOT Challenge 2020

In order to measure the real-world performance of the implemented system, I've evaluated it on the MOT Challenge 2020 dataset[1] for single camera pedestrian tracking. I will refer to this benchmark shortly as MOT20. I've compared the performance with the SORT [7] tracking algorithm, which is still considered a state-of-the-art for realtime MOT in video sequences. I've used the set of public object detections available, preventing the bias which may result from using better/worse object detector. Together with MOT20, a Python script for evaluation [40] with various metrics is provided with the most informative ones being multiple object tracking accuracy (MOTA) [5], higher order tracking accuracy (HOTA) [51], and a well-known F1 score for identification, which is simply a harmonic mean between the precision and recall of target identification. All the mentioned metrics require track-to-target assignments, which are found using the IoU metric. Before running the tracker, noise covariance parameters R and σ_a^2 of the Kalman filter had to be set. Since the proposed system doesn't rely on the IoU and other domain specific heuristics, proper values for those parameters are required in order to maximize the performance. I've analyzed the ground-truth trajectories in the training split and estimated the random acceleration variance σ_a^2 by numerically computing the second derivative of position for both x and y axis and evaluating the second moment of the resulting sample of values. Then, I've estimated the measurement error by computing the second moment of the Euclidean distances between the ground-truth boxes and their respective nearset raw detection boxes. I've also tried an alternative approach based on the equation (7.2) from [1], using notation from section 2.2.1, with ϵ_k being the residual from filtering step:

$$R_k = E\left[\epsilon_k \epsilon_k^\intercal\right] + H_k \cdot P_k \cdot H_k^\intercal \tag{7.2}$$

I've initially set the R to be diagonal matrix representing high (likely suboptimal) variance. Then, I've run the tracking algorithm few times and interestingly enough, the matrix (after orthogonalizing it with eigenvalues) converged to the one found by directly comparing ground-truths with raw detections, which indicates that the random acceleration parameter σ_a^2 was probably well estimated as well. Other parameters, such as P_D, λ_b or λ_f, were either estimated directly from the dateset (e.g. in case of λ_b, this is fairly trivial), or set to some reasonable value by hand. Scan window depth, or as I've called it rather in section 7.1, temporal context, was set to 25 frames (i.e. one second on MOT20 videos), and the maximum number of hypotheses per target was capped to 10. Finally, I've run two implementations of the tracker on MOT20 benchmark, the simple one tracking solely based on the position, and the one augmented with IoU model, as described in section 6.2.2. Results are shown in table 7.1. The frames-pre-second (FPS) values were measured on an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz machine using only single processor core. All metrics present in the table (except FPS) can be interpreted as different types of tracking accuracy, therefore the higher, the better. From table 7.1, a quite interesting observation can be made, that the IoU model definitely does not significantly improve the overall performance, contrary to what I've expected. It even seems that it performs slightly worse, since in both HOTA and IDF1, the simple position based tracker wins by a small margin. However, if the ratios between 3-rd and 2-nd row, i.e. $\sim 1.024, \sim 1.000$ and ~ 0.991 are

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

Tracker	MOTA	HOTA	IDF1	FPS
SORT	45.794	32.420	34.058	31.900
Proposed (position)	54.484	38.771	42.034	427.068
Proposed (position + IoU)	55.795	38.765	41.653	346.741

Table 7.1: Evaluation on MOT Challenge 2020

multiplied together, the overall improvement is ~ 1.015, which at least is slightly greater than one. On the other hand, because of the extra computations in the IoU model, the position only tracking is clearly faster in terms of FPS. I do provide an FPS value for the SORT tracker as well, but it must be mentioned that it is extremely unfair comparison, since I've used the original research implementation, which is unfortunately written in Python. Despite using Numpy, which is written in C, a pure C/C++ implementation of SORT algorithm would probably be the fastest one. Nevertheless, both implementations of the proposed tracking system win in terms of tracking accuracy by a significant margin. At the same time, I do believe that with an exhaustive fine-tuning, the SORT tracker is capable of achieving similar performance on the MOT20 benchmark. What it is almost certainly not capable of, is to generalize into being a multimodal multi-sensor tracker, easily extensible for various types of input data and tracking scenarios.

7.2.2 Multi-Camera Road Junction Monitoring System

Finally, in order to see the real-world performance of the proposed multimodal multi-object tracking system in full deployment, I've tested the second implementation of the system, i.e. the multi-camera model with appearance embeddings described in section 6.2.3, on few videos from up to 4 cameras covering a road junction. As I've mentioned, the visual embedding model was "calibrated" on this scene. Because the embedding vectors are normalized, I believe the estimated distributions are quite general, yet there may be some variability in case of extremely different lighting conditions. For the motion model, however, the `LinearPredictor` class (see section 6.2.1) implementing simple constant velocity model with random acceleration assumption is a very rough approximation of the true dynamics of the objects of interest (mostly vehicles in this case), so the parameters should be set appropriately for the target scene. Because I didn't have any ground-truth data, in contrast to section 7.2.1, I've had to set the σ_a^2 by hand to some reasonable value reflecting the expected acceleration of an average vehicle. For R, similar approach as in section 7.2.1 using equation (7.2) was performed, but in this case, the normalization by detection bounding box size was employed. The reason is that the images are resized to a fixed size using a "letterbox" transform (i.e. based on the major box dimension) for ground position prediction using a neural network. It is obvious, that the prediction error should grow linearly with the scale of an object. However, I do provide a 2D histogram of empirical distribution of estimated $\sqrt{\det R}$ (without normalization) and the major box dimension (i.e. object scale) in figure 7.3. A positive linear correlation is visible.

After setting all the necessary parameters, I've evaluated the tracker by visual inspection, since I was not provided with any ground-truth data and manually annotating the scene would be too time-consuming for me alone. However, I've created a demo application, where the trajectories are projected from the UTM physical space back to a chosen

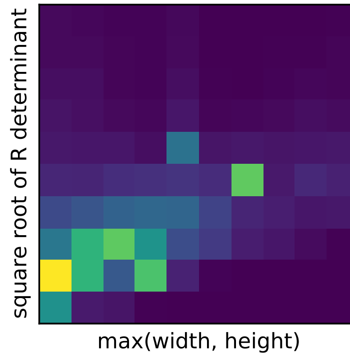
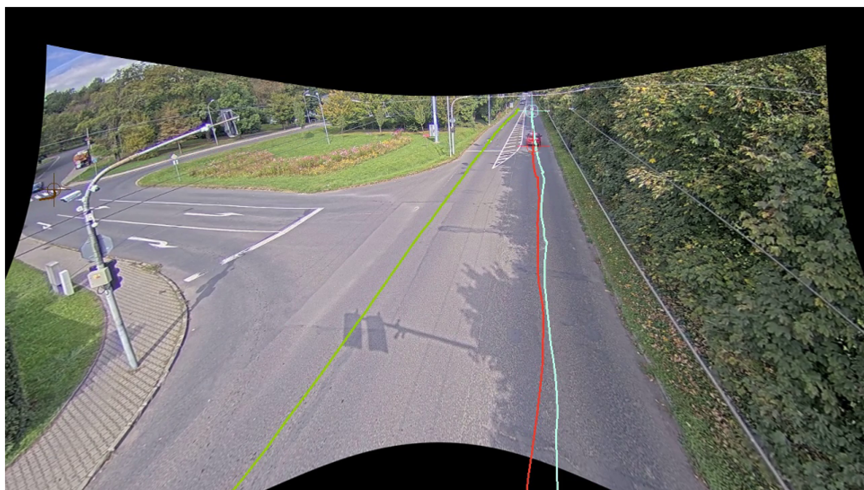


Figure 7.3: 2D histogram of object size and $\sqrt{\det R}$

"master" camera, and visualized using pseudo-randomly generated colors per each target identity. Although the evaluation lacks an objective numerical metric, a visual inspection is often the best way to evaluate the tracking performance since it doesn't compress the complex information into few numbers, and the most undesired failures such as identity switches, track fragmentation, etc., can be almost always captured by a human (except for extremely crowded scenes). For object detection, YOLOv5[2] neural network architecture (essentially the same as described in section 3.1.2) trained on a proprietary dataset aimed for traffic objects detection with 93.29 % mean average precision (mAP) was used. The tracking algorithm operates on frequency of hundreds and occasionally even thousands FPS, even for 4 cameras and 1s temporal context on an Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz machine. However, the models necessary to obtain the input data do require a modern GPU to achieve realtime processing end-to-end. All three data preprocessing models, i.e. detector, ground position predictor, and appearance embedding extractor, run in realtime on an NVIDIA GTX 1060 GPU separately, but to implement a realtime tracking system in end-to-end manner, more computationally capable GPU or a dedicated accelerator hardware is required. Nevertheless, the tracking algorithm itself doesn't seem to be a computational bottleneck in general.

[2] https://github.com/ultralytics/yolov5

(a) Camera 1



(b) Camera 2

(c) Camera 3

Figure 7.4: Example tracking visualization from 3 cameras at the same time

Chapter 8

Conclusion

In this thesis, I've designed and implemented a general extendable tracker, with fixed configurable tracking engine and an implementable interface, enabling user programmer to extend it for any system of sensors and set of measured attributes. I've implemented few basic specializations of the system and evaluated them both on simulated and real-world data. For the multi-sensor part, I've implemented only a homogeneous system of multiple digital cameras, but the overall design makes it possible to easily implement any system of sensors, including the heterogeneous ones. Therefore, I do argue that the implemented system deserves the name "multimodal", despite the lack of existing real-world specialization combining multiple kinds of sensors together. This may be a subject for possible future work. Also, because of the overall complexity of the task, I didn't have the time to implement some more advanced motion filters, e.g. IMM described in section 2.2.4. I believe that using motion filters which model the target maneuvers more reasonably than the random acceleration assumption does, might bring significant improvements in tracking performance for real-world scenarios. Also, I strongly suggest, that using a hierarchical approach in high frequency sensors, such as digital cameras, might bring large improvements as well. More precisely, by hierarchical approach in this context I mean for example the following. Instead of tracking with TOMHT using very deep scan window in order to capture sufficient temporal context, a single-sensor single-hypothesis "pre-tracking" could be performed. Then, the observations may be aggregated within some short temporal window from multiple positions belonging to the same trajectory, preferably augmented by velocity or even higher order position derivatives. Then, TOMHT on much lower frequency (e.g. 4 Hz) could be run on those aggregate observations, maintaining larger temporal context using smaller scan window. This can be viewed as "relaxed" tracklet level fusion, because the target identity labels are removed. Also, the motion derivatives, which are almost impossible to obtain from static observations (without using computer vision related techniques such as optical flow, see chapter 4), can be very helpful in building the validation gates for prior targets. I believe this approach could bring an order of magnitude improvement in tracking quality and speed, but unfortunately, I was not able to implement and test it out within the scope of this thesis due to the lack time. The tracking engine itself (without the particular implementations) is written in pure C++ programming language and together with basic implementations such as Kalman filter, it is easily capable of realtime processing even in challenging crowded scenes and/or when processing data from multiple sensors.

Bibliography

[1] Akhlaghi, S., Zhou, N. and Huang, Z. *Adaptive Adjustment of Noise Covariance in Kalman Filter for Dynamic State Estimation*. 2017.

[2] Babenko, B., Yang, M.-H. and Belongie, S. Visual tracking with online Multiple Instance Learning. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, p. 983–990. DOI: 10.1109/CVPR.2009.5206737.

[3] Bania, P. and Baranowski, J. Field Kalman Filter and its approximation. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. 2016, p. 2875–2880. DOI: 10.1109/CDC.2016.7798697.

[4] Bar Shalom, Y. and Tse, E. T. S. Tracking in a cluttered environment with probabilistic data association. In:. 1975.

[5] Bernardin, K. and Stiefelhagen, R. Evaluating multiple object tracking performance: The CLEAR MOT metrics. *EURASIP Journal on Image and Video Processing*. january 2008, vol. 2008. DOI: 10.1155/2008/246309.

[6] Bertsekas, D. P. *A distributed algorithm for the assignment problem*. 1979.

[7] Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B. Simple Online and Realtime Tracking. *CoRR*. 2016, abs/1602.00763. Available at: http://arxiv.org/abs/1602.00763.

[8] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. 738 p. Information science and statistics. ISBN 9780387310732. Available at: http://www.library.wisc.edu/selectedtocs/bg0137.pdf.

[9] Blackman, S. S. *Design and analysis of modern tracking systems*. Boston, Mass. ; London: Artech House, 1999. Artech House radar library. ISBN 1580530060.

[10] Blackman, S. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*. 2004, vol. 19, no. 1, p. 5–18. DOI: 10.1109/MAES.2004.1263228.

[11] Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020.

[12] Bolme, D., Beveridge, J., Draper, B. and Lui, Y. Visual object tracking using adaptive correlation filters. In:. June 2010, p. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[13] Bradski, G. R. Real time face and object tracking as a component of a perceptual user interface. *Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV'98 (Cat. No.98EX201).* 1998, p. 214–219.

[14] Brigham, E. O. and Morrow, R. E. The fast Fourier transform. *IEEE Spectrum.* 1967, vol. 4, p. 63–70.

[15] Challa, S., Morelande, M. R., Mušicki, D. and Evans, R. J. Introduction to object tracking. In: *Fundamentals of Object Tracking.* Cambridge University Press, 2011. DOI: 10.1017/CBO9780511975837.002.

[16] Coraluppi, S. P. Multiple-Hypothesis and Graph-Based Approaches to Multi-Target Tracking. In:. 2016.

[17] Cox, I. J. and Hingorani, S. L. An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and Its Evaluation for the Purpose of Visual Tracking. *IEEE Trans. Pattern Anal. Mach. Intell.* 1996, vol. 18, p. 138–150.

[18] Cristianini, N. and Shawe Taylor, J. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. In:. 2000.

[19] Deb, S., Yeddanapudi, M., Pattipati, K. and Bar Shalom, Y. A generalized S-D assignment algorithm for multisensor-multitarget state estimation. *IEEE Transactions on Aerospace and Electronic Systems.* 1997, vol. 33, no. 2, p. 523–538. DOI: 10.1109/7.575891.

[20] Di, W., Bhardwaj, A. and Wei, J. *Deep Learning Essentials: Your Hands-on Guide to the Fundamentals of Deep Learning and Neural Network Modeling.* Packt Publishing, 2018. ISBN 1785880365.

[21] Diplaros, A., Vlassis, N. and Gevers, T. A Spatially Constrained Generative Model and an EM Algorithm for Image Segmentation. *IEEE Transactions on Neural Networks.* 2007, vol. 18, no. 3, p. 798–808. DOI: 10.1109/TNN.2007.891190.

[22] Einicke, G. and White, L. Robust extended Kalman filtering. *Signal Processing, IEEE Transactions on.* october 1999, vol. 47, p. 2596 – 2599. DOI: 10.1109/78.782219.

[23] Farnebäck, G. Two-Frame Motion Estimation Based on Polynomial Expansion. In: *SCIA.* 2003.

[24] Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazirbaş, C. et al. *FlowNet: Learning Optical Flow with Convolutional Networks.* 2015.

[25] Ford, L. and Fulkerson, D. Maximal flow through a network. *Can J Math.* 1956, vol. 8, p. 399–404.

[26] Fortmann, T. E., Bar Shalom, Y. and Scheffe, M. Sonar tracking of multiple targets using joint probabilistic data association. *IEEE Journal of Oceanic Engineering.* 1983, vol. 8, p. 173–184.

[27] Friedman, J. H. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*. Institute of Mathematical Statistics. 2001, vol. 29, no. 5, p. 1189 – 1232. DOI: 10.1214/aos/1013203451. Available at: https://doi.org/10.1214/aos/1013203451.

[28] Gamarnik, D., Goldberg, D. and Weber, T. *Correlation Decay in Random Decision Networks*. arXiv, 2009. DOI: 10.48550/ARXIV.0912.0338. Available at: https://arxiv.org/abs/0912.0338.

[29] Genovese, A. The interacting multiple model algorithm for accurate state estimation of maneuvering targets. *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*. october 2001, vol. 22, p. 614–623.

[30] Girshick, R. *Fast R-CNN*. 2015.

[31] Girshick, R., Donahue, J., Darrell, T. and Malik, J. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014.

[32] Gordon, N. J., Salmond, D. and Smith, A. F. M. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In:. 1993.

[33] Gu, J., Yang, X., Mello, S. D. and Kautz, J. Dynamic Facial Analysis: From Bayesian Filtering to Recurrent Neural Network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, p. 1531–1540.

[34] Hafner, D., Demetz, O. and Weickert, J. Why Is the Census Transform Good for Robust Optic Flow Computation? In: *SSVM*. 2013.

[35] He, K., Zhang, X., Ren, S. and Sun, J. *Deep Residual Learning for Image Recognition*. arXiv, 2015. DOI: 10.48550/ARXIV.1512.03385. Available at: https://arxiv.org/abs/1512.03385.

[36] Held, D., Thrun, S. and Savarese, S. *Learning to Track at 100 FPS with Deep Regression Networks*. 2016.

[37] Henriques, J. F., Caseiro, R., Martins, P. and Batista, J. High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Institute of Electrical and Electronics Engineers (IEEE). Mar 2015, vol. 37, no. 3, p. 583–596. DOI: 10.1109/tpami.2014.2345390. ISSN 2160-9292. Available at: http://dx.doi.org/10.1109/TPAMI.2014.2345390.

[38] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J. et al. *Caffe: Convolutional Architecture for Fast Feature Embedding*. 2014.

[39] Jiang, D. and Delgrossi, L. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In: *VTC Spring 2008 - IEEE Vehicular Technology Conference*. 2008, p. 2036–2040. DOI: 10.1109/VETECS.2008.458.

[40] Jonathon Luiten, A. H. *TrackEval* [https://github.com/JonathonLuiten/TrackEval]. 2020.

[41] Julier, S. J. and Uhlmann, J. K. New extension of the Kalman filter to nonlinear systems. In: Kadar, I., ed. *Signal Processing, Sensor Fusion, and Target Recognition VI*. July 1997, vol. 3068, p. 182–193. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. DOI: 10.1117/12.280797.

[42] Kalal, Z., Mikolajczyk, K. and Matas, J. Forward-Backward Error: Automatic Detection of Tracking Failures. *2010 20th International Conference on Pattern Recognition*. 2010, p. 2756–2759.

[43] Kalal, Z., Mikolajczyk, K. and Matas, J. Tracking-Learning-Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* USA: IEEE Computer Society. jul 2012, vol. 34, no. 7, p. 1409–1422. DOI: 10.1109/TPAMI.2011.239. ISSN 0162-8828. Available at: https://doi.org/10.1109/TPAMI.2011.239.

[44] Kalman, R. E. and Others. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*. 1960, vol. 82, no. 1, p. 35–45.

[45] Kaneko, T. and Hori, O. Template update criterion for template matching of image sequences. In:. February 2002, vol. 2, p. 1 – 5 vol.2. DOI: 10.1109/ICPR.2002.1048221. ISBN 0-7695-1695-X.

[46] Kim, S., Kim, D., Cho, M. and Kwak, S. *Embedding Transfer with Label Relaxation for Improved Metric Learning*. arXiv, 2021. DOI: 10.48550/ARXIV.2103.14908. Available at: https://arxiv.org/abs/2103.14908.

[47] Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955, vol. 2, 1-2, p. 83–97. DOI: https://doi.org/10.1002/nav.3800020109. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[48] Lienhart, R. and Maydt, J. An extended set of Haar-like features for rapid object detection. In: *Proceedings. International Conference on Image Processing*. 2002, vol. 1, p. I–I. DOI: 10.1109/ICIP.2002.1038171.

[49] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*. Springer International Publishing. 2016, p. 21–37. ISSN 1611-3349.

[50] Lucas, B. D. and Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. In: *IJCAI*. 1981.

[51] Luiten, J., Osep, A., Dendorfer, P., Torr, P., Geiger, A. et al. HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking. *International Journal of Computer Vision*. Springer. 2020, p. 1–31.

[52] Lukežič, A., Vojíř, T., Zajc, L. Čehovin, Matas, J. and Kristan, M. Discriminative Correlation Filter Tracker with Channel and Spatial Reliability. *International Journal of Computer Vision*. Springer Science and Business Media LLC. Jan 2018, vol. 126, no. 7, p. 671–688. DOI: 10.1007/s11263-017-1061-3. ISSN 1573-1405. Available at: http://dx.doi.org/10.1007/s11263-017-1061-3.

[53] Mehendale, N. and Neoge, S. Review on Lidar Technology. *SSRN Electronic Journal*. january 2020. DOI: 10.2139/ssrn.3604309.

[54] Mihaylova, L. S., Brasnett, P., Canagarajah, N. and Bull, D. R. Object Tracking by Particle Filtering Techniques in Video Sequences. In:. 2007.

[55] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B. FastSLAM: a factored solution to the simultaneous localization and mapping problem. In: *AAAI/IAAI*. 2002.

[56] Murty, K. G. Letter to the Editor - An Algorithm for Ranking all the Assignments in Order of Increasing Cost. *Oper. Res.* 1968, vol. 16, p. 682–687.

[57] Odelson, B. J., Rajamani, M. R. and Rawlings, J. B. A New Autocovariance Least-Squares Method for Estimating Noise Covariances. USA: Pergamon Press, Inc. feb 2006, vol. 42, no. 2, p. 303–308. DOI: 10.1016/j.automatica.2005.09.006. ISSN 0005-1098. Available at: https://doi.org/10.1016/j.automatica.2005.09.006.

[58] Rahmathullah, A. S., Garcia Fernandez, A. F. and Svensson, L. Generalized optimal sub-pattern assignment metric. In: *2017 20th International Conference on Information Fusion (Fusion)*. IEEE, Jul 2017. DOI: 10.23919/icif.2017.8009645. Available at: https://doi.org/10.23919%2Ficif.2017.8009645.

[59] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. *You Only Look Once: Unified, Real-Time Object Detection*. 2016.

[60] Reid, D. B. An algorithm for tracking multiple targets. In:. 1978.

[61] Ren, S., He, K., Girshick, R. and Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016.

[62] Schuhmacher, D., Vo, B.-T. and Vo, B.-N. A Consistent Metric for Performance Evaluation of Multi-Object Filters. *Signal Processing, IEEE Transactions on*. september 2008, vol. 56, p. 3447 – 3457. DOI: 10.1109/TSP.2008.920469.

[63] Shi, J. and Tomasi, C. Good features to track. *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, p. 593–600.

[64] Skolnik, M. I. 36 - Radar. In: Middleton, W. M. and Van Valkenburg, M. E., ed. *Reference Data for Engineers (Ninth Edition)*. Ninth Editionth ed. Woburn: Newnes, 2002, p. 36–1–36–22. DOI: https://doi.org/10.1016/B978-075067291-7/50038-8. ISBN 978-0-7506-7291-7. Available at: https://www.sciencedirect.com/science/article/pii/B9780750672917500388.

[65] Sudano, J. J. The /spl alpha/-/spl beta/-/spl eta/-/spl theta/ tracker with a random acceleration process noise. *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference. NAECON 2000. Engineering Tomorrow (Cat. No.00CH37093)*. 2000, p. 165–171.

[66] Sun, J., Li, Q., Zhang, X. and Sun, W. An Efficient Implementation of Track-Oriented Multiple Hypothesis Tracker Using Graphical Model Approaches. *Mathematical Problems in Engineering*. september 2017, vol. 2017, p. 1–11. DOI: 10.1155/2017/8061561.

[67] Wald, A. Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics*. Institute of Mathematical Statistics. 1945, vol. 16, no. 2, p. 117 – 186. DOI: 10.1214/aoms/1177731118. Available at: https://doi.org/10.1214/aoms/1177731118.

[68] Wojke, N. and Bewley, A. Deep Cosine Metric Learning for Person Re-identification. In: IEEE. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018, p. 748–756. DOI: 10.1109/WACV.2018.00087.

[69] ysekikawa, Y. S. and Suzuki, K. Fast Eigen Matching Accelerating Matching and Learning of Eigenspace method. In:. 2016.

[70] Zhou, Y. and Tuzel, O. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. 2017.

Appendix A

Contents of the included storage media

- **demo_video.mp4** – demonstration video
- **example_data/** – example synchronized geo-registered videos from 4 cameras
- **masters_thesis_src/** – source code for the text of this thesis
- **scripts/** – Python scripts for data analysis, preprocessing, and simulations
- **tracker/** – tracking engine, interface and example implementations written in C++
- **tracker_apps/** – demonstration tracking applications