Multi-Target Tracking using Random Finite Set based Bayesian Filtering in a Heterogeneous Platform

Master Thesis

Author: Uzair Sharif
Carried out at: Robotics & Embedded Systems, Department of Informatics, TUM
Advisor: M.Sc. Biao Hu
Supervisor: Dr. Kai Huang & Prof. Dr.-Ing. Walter Stechele
Submission date: 06 October, 2016
Confirmation

Herewith I confirm that I independently prepared this work. No further references or auxiliary means except those declared in this document have been used.

Munich; 06 October, 2016

....................

Uzair Sharif
Abstract

Advanced Driver Assistance Systems (ADAS) prevalent in modern automotive vehicles rely on their ability to recognize relevant traffic participants such as other vehicles, pedestrians etc. along with other environment information like road-signs, road lane-trackings etc. Due to intensive computational budget requirements of ADAS intelligence applications, research efforts, at TUM Robotics and Embedded Institute, are underway to come up with innovative compute platforms comprising a heterogeneous mix of various technologies, like multi-core CPUs, GPUs and FPGAs, in order to tackle this computational complexity.

An important role of modern ADAS systems is to be able to distinguish pedestrians and track their motion to make intelligent driving decisions so as to detect dangerous situations, in terms of safety, ahead of time. The associated computer vision problem that needs to be solved requires the detection and tracking of multiple targets (pedestrians) from a moving camera platform. The aim of this Master thesis work is to come up with an efficient and fully functional pedestrian-tracking system implementation which can be run under real-time constraints offering decent tracking accuracy. For this purpose, recently proposed RFS based filtering techniques like PHD,LMB filters are chosen to tackle the underlying multi-target tracking problem after a thorough overview of the published literature in this field. These tracking algorithms are implemented following a highly modular system design approach. An extensive evaluation analysis is then carried out to ensure the fulfillment of sufficient accuracy requirements. This is followed by extensive profiling analysis to spot the potential bottlenecks in terms of execution performance which are then targeted to come up with an OpenCL accelerated application. Video-throughput improvements from roughly 15fps to 100fps (6x) are observed on average while processing typical MOT benchmark videos. Moreover, the worst-case frame processing yielded an 18x advantage from nearly 2fps to 36fps, thereby, comfortably meeting the real-time constraints.
# Contents

Abstract iii

List of Figures vii

List of Tables ix

List of Abbreviations xi

1. Introduction 1
   1.1. Motivation ........................................ 1
   1.2. Problem Statement .................................. 2
   1.3. Thesis Structure ................................... 3

2. Multi-Target Tracking 5
   2.1. The target tracking problem ...................... 5
      2.1.1. Objectives .................................... 5
      2.1.2. Challenges ................................... 6
   2.2. Solution to tracking problem .................... 7
      2.2.1. Bayesian Inference ............................ 7
   2.3. Single Target Tracking ............................ 9
      2.3.1. Kalman Filter ................................ 10
      2.3.2. Recursive Bayesian filter approximations .... 12
   2.4. Multiple Target Tracking ......................... 15
      2.4.1. Global Nearest Neighbour Filter .............. 16
      2.4.2. Joint Probabilistic Data Association Filter ... 16
      2.4.3. Multiple Hypothesis Tracking Filter ........... 16
      2.4.4. Random Finite Set based Filtering ............ 17

3. RFS based Multi-Target Filtering 19
   3.1. Motivation ......................................... 19
   3.2. Random Finite Set formulation .................... 20
      3.2.1. A Random Finite Set .......................... 20
      3.2.2. Multi-Target Bayes Filter .................... 21
      3.2.3. RFS Filters .................................. 23
   3.3. The LMB Filter .................................... 30
      3.3.1. LMB Filter Prediction ......................... 30
      3.3.2. LMB Filter Update ............................ 31
## 4. System Design & Implementation

4.1. System Design ........................................... 33
   4.1.1. Modules ........................................... 33
   4.1.2. Interfaces ........................................... 35
   4.1.3. System Upgrades .................................... 35

4.2. System Implementation .................................... 36
   4.2.1. GM-PHD Tracker .................................... 36

4.3. SMC-LMB Tracker ........................................ 38
   4.3.1. Filter Initialization ................................ 38
   4.3.2. Birth Model ........................................ 40
   4.3.3. Predict LMB ........................................ 40
   4.3.4. Predict-LMB to Predict-$\delta$GLMB ................. 40
   4.3.5. Update LMB intermediate ............................ 40
   4.3.6. $\delta$-GLMB Update ................................ 41
   4.3.7. Update LMB ........................................ 41
   4.3.8. Track Management .................................. 41
   4.3.9. State Estimation ................................... 41

4.4. OpenCL Acceleration ...................................... 41

## 5. Evaluation and Analysis ................................ 43

5.1. Evaluation Metrics ..................................... 43
   5.1.1. Tracking Accuracy .................................. 43
   5.1.2. Execution Performance .............................. 43

5.2. Tracking Accuracy Analysis .............................. 44
   5.2.1. Simulation Analysis ................................ 44
   5.2.2. MOT Dataset Analysis ............................... 52
   5.2.3. Influence of Detector Accuracy ...................... 54
   5.2.4. Guide to set LMB Tracker parameters ............... 56

5.3. Execution Performance Analysis ......................... 57
   5.3.1. Sequential (C++) Implementation ................... 57
   5.3.2. OpenCL Acceleration ............................... 58

## 6. Conclusion .............................................. 63

6.1. Summary ................................................. 63
   6.2. Future Work .......................................... 63

## A. Fundamentals ........................................... 67

A.1. Bayes Recursive Filter for Object Tracking ............. 67

## Bibliography .................................................. 69
## List of Figures

1.1. A brief subset of MTT Applications ........................................ 1
1.2. High-level overview of an MTT context ................................. 2
2.1. The tracking problem ......................................................... 6
4.1. Block diagram of Pedestrian Tracking System .......................... 33
4.2. Block diagram of GM-PHD filter recursion .............................. 37
4.3. Block diagram of SMC-LMB Tracker Recursion ......................... 39
5.1. Simulation scenario .......................................................... 45
5.2. Tracking accuracy of GM-PHD Tracker ................................. 46
5.3. Tracking accuracy of SMC-LMB Tracker ............................... 47
5.4. LMB Tracker accuracy: clutter density .................................. 49
5.5. LMB Tracker accuracy: sensor imperfections .......................... 50
5.6. LMB Tracker accuracy: state-space model deviation ................. 51
5.7. LMB Tracker accuracy: tracker parameters ............................ 53
5.8. LMB Tracker accuracy: MOT dataset videos ........................... 55
5.9. LMB Tracker accuracy: HOG Detector .................................. 56
5.10. C++ LMB Tracker execution performance ............................... 59
5.11. OpenCL accelerated LMB Tracker execution performance ............ 60
5.12. Comparison between C++ and OpenCL LMB implementations ....... 61
5.13. Scalability of C++ and OpenCL LMB implementations ............... 62
List of Tables

5.1. LMB Tracker configurations ........................................ 52
5.2. MOT Dataset videos .................................................. 54
5.3. LMB Tracker parameters ............................................. 56
List of Abbreviations

ADAS  Advanced Driving Assistance System
CBMeMBer  Cardinality Balanced Multi-target Multi-Bernoulli filter
COTS  Commercial Off The Shelf
CPHD  Cardinalized Probability Hypothesis
CPU  Central Processing Unit
DSP  Digital Signal Processor
ECU  Electronic Control Unit
EKF  Extended Kalman Filter
FISST  Finite Set Statistics
FPGA  Field Programmable Gate Arrays
GLMB  Generalized Labeled Multi-Bernoulli
GM-PHD  Gaussian Mixture - Probability Hypothesis Density
GNN  Global Nearest Neighbour
GPU  Graphics Processing Unit
HMM  Hidden Markov Model
LMB  Labeled Multi-Bernoulli
MeMBer  Multi-target Multi-Bernoulli filter
MHT  Multiple Hypothesis Tracking
MTT  Multi-Target Tracking
NN  Nearest Neighbour
OSPA  Optimal Sub-pattern Assignment
PDA  Probabilistic Data Association
PHD  Probability Hypothesis Density
RFS  Random Finite Set
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computing</td>
</tr>
<tr>
<td>SIR</td>
<td>Sequential Importance Resampling</td>
</tr>
<tr>
<td>SMC</td>
<td>Sequential Monte Carlo</td>
</tr>
<tr>
<td>STT</td>
<td>Single Target Tracking</td>
</tr>
<tr>
<td>TUM</td>
<td>Technische Universitaet Muenchen</td>
</tr>
<tr>
<td>UKF</td>
<td>Unscented Kalman Filter</td>
</tr>
<tr>
<td>UT</td>
<td>Unscented Transform</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. Motivation

The Multi-Target Tracking (MTT), in general, has hitherto been a heavily researched and extensively explored topic within Machine Vision. Trackers, incorporating the use of such techniques, are fundamental to all sorts of surveillance, guidance and obstacle avoidance systems. With the ever increasing advancement of computational infrastructure, and wide-spread availability of high quality sensing technologies, MTT research continues to gather momentum and is poised to play a key role as a vital building block in many futuristic technologies like autonomous driving, cognitive robotics, internet-of-things etc.

![Figure 1.1: A brief subset of MTT Applications](image)

Formally, MTT can essentially be posed as a statistical problem of jointly estimating the number of targets along with their individual states or trajectories within an environment, from an observation of noisy corrupted or false sensor measurements, with the passage of time. Therefore, a typical MTT context is expected to comprise of multiple objects or targets to be tracked; a sensor or multiple sensors which measure some aspect of the considered targets; and most importantly a
signal processor that runs suitable algorithms for solving the MTT problem as shown graphically in Figure 1.2.

![Figure 1.2: High-level overview of an MTT context](image)

It is specifically the design of this signal-processor module that has garnered much attention and research focus world-wide during the past seven decades. The holy grail, so to speak, being sought by these researchers within their quest, can broadly be expressed as coming up with a functional highly efficient implementation characterized by negligible target’s state estimation error while being computationally tractable. While this presented a classical engineering trade-off between accuracy and execution performance in the past, the emergence of advanced compute platforms, comprising a heterogeneous mix of commercial off the shelf (COTS) hardware offerings like RISC CPUs, FPGAs, GPUs, DSPs etc. coupled with ever improving software design flow to harness their compute potential, have enabled the researchers to seek the best of both worlds today.

This thesis work mainly concerns with this research problem in general, and aims to make a valuable contribution in this regard; whereby providing an MTT solution that could well be used, in future, by a higher level application algorithm in carrying out its functionality.

1.2. Problem Statement

In the context of TU9 research projects [2], the Institute of Robotics & Embedded Systems at TUM [1] has been involved in dealing with the challenge of providing high-performant ECUs as an enabling technology applicable in the automotive field, relying on a heterogeneous system with a multi-core CPU, FPGA, GPU etc. Prior to the start of this thesis, the research group had
1.3. Thesis Structure

This thesis document is mainly divided into three parts. The first part serves to introduce the reader to the fundamentals of object tracking in general. The second part details the specific methodology being carried out to fulfill the thesis objectives along with its implementation details. Finally, the third part provides an extensive evaluation and analysis of the proposed algorithm in terms of its tracking accuracy and execution performance in various pedestrian tracking scenarios. Specifically:

• chapter 2 introduces the bare fundamentals of mature statistical theory of Bayesian Inference to solve dynamic estimation problems. This is followed by a brief overview of the pre-dominant techniques being used to tackle the multi-target tracking problem in general.
• chapter 3 introduces the recently proposed Random Finite Set based formulation of the optimal Recursive Bayes filter. It then provides detailed theoretical background of the GM-PHD and the SMC-LMB trackers whose design and implementation is the main focal point of this thesis work.

• chapter 4 presents the overall modular system design approach to tackle the pedestrian tracking problem in the automotive context. This is followed by a description of the various implementation aspects in coming up with a functional implementation of the GM-PHD, SMC-LMB tracker algorithms.

• chapter 5 presents an extensive analysis that is carried out to evaluate the efficiency of the tracker implementation both in terms of its accuracy as well as its execution performance.

• chapter 6 concludes the chapter providing insights regarding possible research-related extensions to this thesis work.
2. Multi-Target Tracking

This chapter lays down the bare fundamentals needed in order to understand much of the advanced algorithms and techniques which are at the fore-front of the current MTT research. Starting off with a mathematical formulation of a simpler single target tracking (STT) scenario, it extends those ideas to the much harder problem of multi-target tracking. This chapter concludes by providing Bayesian filtering as a theoretically optimal solution to the MTT problem.

2.1. The target tracking problem

Target tracking refers to the problem of using sensor measurements to determine the location, path and characteristics of the targets of interest within the surveillance environment. In layman’s terms, this corresponds to following such targets of interest, within the environment, over the passage of time thus yielding target trajectories or tracks.

A target is any moving object in that environment whose state (generally its kinematic state) is of interest. A sensor can be any measuring device such as radar, sonar, camera, lidar, microphone, ultrasound or any other sensor that can be used to collect information about those targets in the environment. The sensor provides these measurements continually over the passage of time. Each of these time instances are referred to as scans. These scans may however also contain measurements of other objects within the environment that are not of interest. Such unwanted measurements in these scans are referred to as clutter. In each time-step, within the scan, each measurement has to be classified either as being a false-alarm or clutter, the start of a new track, or as the continuation of an existing track. This phenomenon in literature is compactly referred to as data association.

2.1.1. Objectives

The typical objectives of target tracking are the determination of number of objects, their identities, and their states such as kinematic attributes like position, velocity or in some cases their specific features with the passage of time. A typical example of target tracking is the radar tracking of aircrafts in its surveillance zone. The tracking problem in this context attempts to determine the number of aircrafts; their types like military, commercial or recreational; their unique identities; and their motion attributes, like their 3D positions and speeds, using the measurements obtained from the radar regularly spaced in time.

In a nut-shell, the tracking algorithm is designed to solve the data association problem within each scan. Fundamentally, it accomplishes this by using a two-step process: a) making target’s state predictions based on past history and b) associating each of this prediction with each new measurement in order to classify it as per data association, meanwhile updating current tracks.
2. Multi-Target Tracking

This process is visualized graphically as in Figure 2.1. As shown here, the current scan consists of five new measurements regarding the already tracked target. The tracking algorithm, having already made a prediction of the target’s position on previous scan, will update its prediction as per the new measurements for a better estimation of target track.

2.1.2. Challenges

One may argue over using solely continuous object detections for the sake of tracking objects in an environment thereby avoiding the use of tracking algorithms altogether. However, there are a number of sources of uncertainty, that arise in this methodology, which render such a primitive approach to tracking useless in most scenarios. These sources of uncertainties, then also act as the main challenges that need to be overcome by the tracker to be of any advantage. A brief overview of such challenges follows:

- Object motion is often subject to random disturbances, for example objects may appear or disappear randomly within the scan sensors due to occlusions or missed detections or simply because of being moved out of the environment. Moreover they could exhibit complex motion characteristics which are hard to model.

- Apart from model/process and sensor’s measurement noises, the multi-target tracker has to contend with much more complex sources of uncertainty, such as measurement origin uncertainty, false alarms, missed detections, target birth/death etc.
• Due to the combinatorial nature of the data association problem inherent in MTT, the computational complexity of tracking algorithm could become hard to meet, by conventional computer technologies, even with moderate number of targets.

• In a multi-sensor settings, a multi-target tracker has to deal with measurements from multiple heterogeneous sensors such as radar, infrared, camera etc. and is expected to process them using some sort of data-fusion techniques in reasonable amount of time.

2.2. Solution to tracking problem

As mentioned in chapter 1, the tracking problem encountered in almost all of tracking scenarios of interest can be formulated as a state-estimation problem within the realm of estimation theory. State-space analysis can then be carried out to effectively solve this problem while overcoming its various challenges. As will be shown later in this section, the recursive Bayesian filter provides a theoretically optimal solution. In essence, all of the mainstream tracking algorithms, proposed so far, can be seen as an approximation to this solution.

2.2.1. Bayesian Inference

The Bayesian Inference is a well-developed probabilistic and statistical theory which fundamentally relies on the use of Bayes’ theorem to solve large variety of complex engineering problems. Fundamentally, the Bayesian paradigm is a theoretical framework for reasoning under uncertainty. The tracking problem lends itself naturally to be solved by these Bayesian estimation techniques.

In most target tracking scenarios, measurements from the sensors are received in a sequential manner over time. At each sensor scan, new estimates of the target are derived utilizing the information obtained from current scan’s measurements. These latest estimates are updated in light of new information received via future scans and so on. The recursive application of Bayes’ theorem is an appropriate framework for handling such sequential form of measurement data and the associated uncertainties.

Bayes’ theorem

Bayes’ theorem is the encapsulation of a philosophy to consistently reconcile past and current information by exploiting the conditional probability concepts of probability theory. Formally, given two related probabilistic events $x$ and $y$, the conditional probability of event $x$ given observation of event $y$ is given by

$$p(x|y) = \frac{p(x,y)}{p(y)} \quad (2.1)$$

or equivalently by

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \quad (2.2)$$

Various aspects of the object tracking problem e.g. the number of objects and their states, can be
modeled as events \( x \) and many types of sensor outputs e.g. radar returns or infrared images etc. as events \( y \). Recursive application of Bayes’ theorem can then be carried out to obtain the conditional probabilities \( p(x|y) \) as a stochastic answer to the tracking problem.

### Application to target tracking

In target tracking, the complete probabilistic knowledge of the individual target states can be described by the joint pdf \( p(X^k) = p(X_k, X_{k-1}, X_{k-2}, ..., X_0) \) where \( X_k \) denotes a generic target state that could represent single or multiple target state, or the number of objects, or the identity of target/s or a combination of these at \( t_k \) time-step or equivalently the k-th scan. Let \( z^k = (z_k, z_{k-1}, z_{k-2}, ..., z_1) \) be the sensor outputs, with \( z_t \) being the measurement at \( t_t \). When a physical relationship governing \( z^k \) and the target states \( X^k \) is known alongwith the stochastic knowledge of target states in the form of \( p(X^k) \), then the Bayes’ theorem (2.2) provides a fundamentally sound method to update the knowledge of these states as given by:

\[
p(X^k | z^k) = \frac{p(z^k | X^k)p(X^k)}{p(z^k)} \tag{2.3}
\]

In the context of Bayesian estimation, the conditional probability distribution \( p(X^k | z^k) \) is referred to as the posterior distribution of the generic target state while the term \( p(X^k) \) as its prior distribution. The term \( p(z^k | X^k) \) is the likelihood function, which expresses the probability that the observed sequence of measurements is \( z^k \) given that the underlying target states were \( X^k \). Finally the term \( p(z^k) \) is the normalization factor as it helps to make the posterior distribution a valid probability measure within \((0, 1)\) interval.

As expressed by (2.3), the application of Bayes’ theorem in the context of target tracking leads to a process where the updated knowledge regarding the target’s states can be obtained by taking the functional product of the likelihood function with the prior distribution within the state-space followed by re-normalization with the normalizing factor. Theoretically, this provides a solution for computing the probabilistic distribution target states of interest utilizing all the measurements information up till this scan. However, the measurements are received in a sequence over time. To use the information as soon as it is received to make better estimates for a given scan, the distribution of the target’s states can rather be updated at each time-step. This leads to a recursive form of Bayesian estimation, where the posterior distribution \( p(X^k | z^k) \) is obtained at \( t_k \), using the current measurement \( z_k \) and the posterior \( p(X^{k-1} | z^{k-1}) \) at \( t_{k-1} \). The recursive form of the Bayes estimator is given by:

\[
p(X^k | z^k) = \frac{p(z_k | X_k)p(X_k | X_{k-1})p(X^{k-1} | z^{k-1})}{p(z_k | z^{k-1})} \tag{2.4}
\]

Due to computational constraints, in most tracking scenarios, it is deemed sufficient enough to compute only the state conditional density \( p(X_k | z^k) \) instead of the whole posterior in estimating targets states. Recalling \( X^k \) to be \( (X_k, X_{k-1}, X_{k-2}, ..., X_0) \), this state conditional density is simply the marginal density of the joint posterior, and so can be expressed via (see Appendix A):

\[
p(X_k | z^k) = \frac{p(z_k | X_k) \int_{X_{k-1}} p(X_k | X_{k-1}) p(X_{k-1} | z^{k-1}) dX_{k-1}}{p(z_k | z^{k-1})} \tag{2.5}
\]
(2.5) is the fundamental target state density which most target tracking algorithms compute or seek to approximate. The integral in (2.5) is the *Chapman-Kolmogorov equation* which can be formulated as:

\[ p(X_k|z^{k-1}) = \int_{X_{k-1}} p(X_k|X_{k-1})p(X_{k-1}|z^{k-1})dX_{k-1} \]  

\[ (2.6) \]

So in a nut-shell, the optimal recursive Bayesian estimation can be carried out via a two-step process for any \( t_k \) as:

- First the predicted state density \( p(X_k|z^{k-1}) \) is computed using (2.6)
- This predicted state is then corrected by the likelihood factor \( p(z_k|X_k) \) using current measurements \( z_k \), followed by normalization as shown in (2.5)

### 2.3. Single Target Tracking

In this section, we apply the optimal recursive Bayes filtering derived in previous section to STT problem. This problem is much simpler as compared to the general multi-target tracking problem but serves to establish the fundamental ideas which extend further to multi-target tracking scenarios.

#### Target dynamics equation

Let \( x_k \in \mathbb{R}^{n_x} \) denote the single target’s state at \( t_k \). To model the various uncertainties, this state is expressed as a random vector having \( n_x \) components. Target dynamics are usually modeled using a stochastic difference equation of the form:

\[ x_k = g(x_{k-1}, v_k) \]  

\[ (2.7) \]

where \( g(\cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_v} \mapsto \mathbb{R}^{n_x} \) is assumed to be a twice continuously differentiable function of its arguments. The disturbance \( v_k \) is a random noise input to the system which is used to account for modeling errors, also commonly referred to as the process noise. For most tracking problems of interest, the target motion model implies this process noise to be additive. So target dynamics equation becomes:

\[ x_k = f(x_{k-1}) + v_k \]  

\[ (2.8) \]

where \( f(x_{k-1}) \) transforms \( x_{k-1} \) as per Hidden Markov Models (HMM). Under such an additive noise assumption and Markovian dynamics, the *state-transition density* \( p(x_k|x_{k-1}) \) can be expressed as:

\[ p(x_k|x_{k-1}) = p_{v_k}(g^{-1}(x_k, x_{k-1})) | \nabla_{x_k} g^{-1}(x_k, x_{k-1}) | = p_{v_k}(x_k - f(x_{k-1})) \]  

\[ (2.9) \]
Sensor measurement equation

Let \( z_k \in \mathbb{R}^{n_z} \) denote the observed measurement at \( t_k \). Most of the sensors used for object tracking can be adequately described using sensor models of the form:

\[
    z_k = I(x_k, w_k)
\]  

where \( I(\cdot) : \mathbb{R}^{n_z} \times \mathbb{R}^{n_w} \rightarrow \mathbb{R}^{n_z} \) is assumed to be a twice continuously differentiable function of its arguments. The random variable \( w_k \) models the sensor measurement error. Like the process noise, most tracking algorithms impose an additive assumption on the measurement noise, yielding the measurement equation of the form:

\[
    z_k = h(x_k) + w_k
\]

Prediction and Filtering

Using (2.9) and (2.12), the prediction and filtering steps for recursive Bayesian filter can be re-expressed as:

- **Prediction:**

\[
    p(x_k | z_{k-1}) = \int p_{v_k}(x_k - f(x_{k-1})) p(x_{k-1} | z_{k-1}) dx_{k-1}
\]  

- **Filtering/Update:**

\[
    p(x_k | z_k) = \frac{p_{w_k}(z_k - h(x_k)) p(x_k | z_k)}{p(z_k | z_{k-1})} \int p_{w_k}(z_k - h(x_k)) p(x_k | z_k) dx_k
\]

2.3.1. Kalman Filter

The Kalman Filter [12] is the most popular representative of a discrete Bayesian estimator and is extensively being used in the context of STT. Mathematically, Kalman filter can be seen as the specialization of the generic Bayesian filter introduced in previous sections. Kalman filtering gives the optimal solution to the STT problem, under these conditions:

- The object dynamics (2.8) and measurement equations (2.11) are linear of the form:

\[
    x_k = F_k x_k + v_k \quad (2.16)
\]

\[
    z_k = H_k x_k + w_k \quad (2.17)
\]

- \( v_k \) and \( w_k \) are white, uncorrelated, Gaussian noise sequences with zero mean and some covariance matrices given by \( Q_k, R_k \) respectively.
• The posterior density of the target state \( p(x_{k-1}|z_{k-1}) \) at any \( t_{k-1} \) is Gaussian with mean \( \hat{x}_{k-1|k-1} \) and covariance \( P_{k-1|k-1} \).

Under these assumptions, we can derive the Kalman filter using the analysis provided in this section. Since \( p_{v_k} \) is a Gaussian density with zero mean and covariance \( Q_k \), the state transition density from (2.9) becomes:

\[
p(x_k|x_{k-1}) = N(x_k; F_k x_{k-1}, Q_k) \tag{2.18}
\]

As per the third assumption, \( p(x_{k-1}|z_{k-1}) = N(x_k; \hat{x}_{k-1|k-1}, P_{k-1|k-1}) \), so putting these results into (2.6), we get:

\[
p(x_k|z_{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{k-1})dx_{k-1} = \int N(x_k; F_k x_{k-1|k-1}, Q_k)N(x_k; \hat{x}_{k-1|k-1}, P_{k-1|k-1})dx_{k-1} \tag{2.19}
\]

To solve this integral, we can make use of the Gaussian Product Theorem to get:

\[
p(x_k|z_{k-1}) = N(x_k; \hat{x}_{k|k-1}, P_{k|k-1}) \tag{2.20}
\]

\[
\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} \tag{2.21}
\]

\[
P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{2.22}
\]

This equation-set, aptly referred to as the Kalman prediction step, provides the equations to compute the Kalman predicted state density. These equations also show that if the posterior density at \( t_{k-1} \) is Gaussian and the targets are taken to be linear/Gaussian, then the computed predicted state density is also Gaussian.

Now, we can carry out a similar analysis to (2.12) and measurement prediction density as:

\[
p(z_k|\hat{z}_{k}^{k-1}) = \int p(z_k|x_k)p(x_k|\hat{z}_{k}^{k-1})dx_k = \int N(z_k; H_k x_k, R_k)N(x_k; \hat{x}_{k|k-1}, P_{k|k-1})dx_k \tag{2.23}
\]

\[
p(z_k|\hat{z}_{k}^{k-1}) = N(z_k; \hat{z}_{k|k-1}, S_k) \tag{2.24}
\]

\[
\hat{z}_{k|k-1} = H_k \hat{x}_{k|k-1} \tag{2.25}
\]

\[
S_k = H_k P_{k|k-1} H_k^T + R_k \tag{2.26}
\]

Finally using these state-prediction and measurement-prediction densities, we can easily derive the posterior conditional state density. Using (2.5), we get:

\[
p(x_k|z_k) = \frac{N(z_k; H_k \hat{x}_{k|k-1}, R_k)N(x_k; \hat{x}_{k|k-1}, P_{k|k-1})}{N(z_k; H_k \hat{x}_{k|k-1}, S_k)} \tag{2.27}
\]

\[
p(x_k|z_k) = N(x_k; \hat{x}_{k|k}, P_{k|k}) \tag{2.29}
\]

\[
K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{2.27}
\]

\[
\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - \hat{z}_{k|k-1}) \tag{2.29}
\]

\[
P_{k|k} = P_{k|k-1} - K_k H_k P_{k|k-1} \tag{2.29}
\]
2. Multi-Target Tracking

where we have again made use of the Gaussian Product Theorem to yield Kalman update equations. Again we notice that under the Kalman filter assumptions, we find that the posterior density is also Gaussian. This is extremely convenient since it means that, given a Gaussian prior density at \( t_0 \) and assuming linear/Gaussian target dynamics and sensor measurements, the posterior density for all \( t_k \) can be represented compactly by just a mean vector and a covariance matrix.

2.3.1.1. Nearest Neighbour Filter

The Nearest Neighbour (NN) filter [3] is the simplest approach to STT in clutter. Given the mean and covariance of the prior density, the NN filter first performs the standard Kalman prediction step to obtain the prediction density. For the data association, this filter simply chooses the measurement closest to the predicted measurement as the true measurement and uses this within the Kalman filtering step to compute the posterior density.

Although, the NN filtering scheme is intuitively appealing and simple to implement, it performs poorly in practice for common tracking scenarios. Especially, with low probability of detection, the NN filter becomes very susceptible to track loss and exhibits poor tracking performance.

2.3.1.2. Probabilistic Data Association Filter

The Probabilistic Data Association (PDA) filter [3] is an important representative of the soft-decision class of tracking algorithms. It contrasts with the NN filter by incorporating all the received measurements within the data association step so as to avoid incorrect measurement to target association. Like the NN filter, PDA filter first computes the standard Kalman prediction density using the prior density. For data association, it calculates an association probability for each of the measurement, as if it is being generated by the target being tracked. Average of all measurements, weighted according to their association probabilities, are then used within the Kalman filtering step to compute the posterior density. To reduce computational requirements, the process of measurement gating or validation is usually carried out to identify the subset of measurements that are somewhat closer to the predicted while discarding the rest of them. This results in less computations in the data association stage.

The PDA filter shows reasonable performance in practice and is commonly used in single tracking problems in diverse range of applications. Because of its wide-spread use, various research efforts have focused on further improving its performance and these endeavors have led to the innovation of several variants of the PDA filter such as the Integrated PDA (IPDA) filter [20], the Multipath PDA filter (MPDA) [23] filters etc.

2.3.2. Recursive Bayesian filter approximations

As mentioned in the previous section, under linear-Gaussian state-space formulation with additive process/measurement noise, the Kalman filter provides a compact analytic optimal Bayesian estimation solution to the tracking problem. However, if any of these assumptions do not hold, then barring special cases, the exact computation of the posterior density becomes impossible due to the integrals in Bayesian prediction/filtering steps. The solution then amounts to developing
2.3. Single Target Tracking

an accurate and computationally feasible approximations to the exact Bayesian recursion. A brief overview of some of these techniques follows.

2.3.2.1. Kalman Filter extensions

In many target tracking applications, it is reasonable to assume Gaussian process and measurement noises. Furthermore, for computational reasons, it is desirable and reasonably accurate to adopt a Gaussian approximation to the posterior distribution since it can then be expressed solely by a mean vector and a covariance matrix as mentioned earlier. However, the assumption of linear dynamic and measurement equations generally violates. The key idea then has been to approximate such non-linear characteristics. This approach leads to different variants of Kalman filter like Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF) etc.

The main gist behind the EKF is to analytically approximate the non-linear functions $f(x_{k-1}), h(x_k)$ within the Bayesian formulation via Taylor series expansion. Generally, the EKF uses such approximation to zeroth and first-order terms. Higher order EKFs may be obtained by retaining more terms of the expansion, however they only tend to provide performance benefits when the measurement noise is small. Interested readers are directed to [32] for a thorough review of such filters. Here we present the EKF recursion steps only:

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + K_k (z_k - \hat{z}_{k|k-1})$$ (2.30)
$$P_{k|k-1} = P_{k-1|k-1} - K_k H_k P_{k|k-1}$$ (2.31)

where

$$\hat{z}_{k|k-1} = h(x_{k|k-1})$$ (2.32)
$$\delta_k = H_k P_{k|k-1} H_k^T + R_k$$ (2.33)
$$K_k = P_{k-1|k-1} H_k^T S_k^{-1}$$ (2.34)
$$H_k = \nabla_x h(x) \big|_{x=\hat{x}_{k|k-1}}$$ (2.35)
$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1})$$ (2.36)
$$P_{k|k-1} = F_k F_k^T P_{k-1|k-1} + Q_k$$ (2.37)
$$F_k = \nabla_x f(x) \big|_{x=\hat{x}_{k-1|k-1}}$$ (2.38)

The UKF can be seen as an alternative to the EKF which shares its computational complexity, while avoiding the Jacobian computations, but achieves greater tracking accuracy in many scenarios. The UKF is based on the Unscented Transform (UT) which helps in approximating the moments of a non-linearly transformed random variable. For the sake of brevity, using [31], here we just present the filtering equations for UKF. Before applying the recursion, a set of sigma points $(X_{k-1}^1, X_{k-1}^2, ..., X_{k-1}^s)$ and weights $(w^1, w^2, ..., w^s)$ is determined to best match the mean $\hat{x}_{k-1|k-1}$ and covariance matrix $P_{k-1|k-1}$ which are then used to compute the prediction density as
shown:

\[
X_k^i = f(X_{k-1}^i) 
\]

\[
\hat{x}_{k|k-1} = \sum_{i=1}^{s} w^i X_k^i 
\]

\[
P_{k|k-1} = Q_k + \sum_{i=1}^{s} w^i (X_k^i - \hat{x}_{k|k-1})(X_k^i - \hat{x}_{k|k-1})^T 
\]

Now again, another set of sigma points \((X_1^1, X_2^1, \ldots, X_s^1)\) and weights \((w_1, w_2, \ldots, w_s)\) is determined to best match the new predicted mean \(\hat{x}_{k|k-1}\) and new predicted covariance matrix \(P_{k|k-1}\). Using these sigma points, the UKF recursive equations are:

\[
\hat{x}_{k|k} = \hat{x}_{k|k-1} + \psi_k S_k^{-1} (Z_k - \hat{Z}_{k|k-1}) 
\]

\[
P_{k|k} = P_{k|k-1} - \psi_k S_k^{-1} \psi_k^T 
\]

where

\[
Z_k^i = h(X_k^i) 
\]

\[
\hat{Z}_{k|k-1} = \sum_{i=1}^{s} w^i Z_k^i 
\]

\[
S_k = R_k + \sum_{i=1}^{s} w^i (Z_k^i - \hat{Z}_{k|k-1})(Z_k^i - \hat{Z}_{k|k-1})^T 
\]

\[
\psi_k = \sum_{i=1}^{s} w^i (X_k^i - \hat{x}_{k|k-1})(Z_k^i - \hat{Z}_{k|k-1})^T 
\]

### 2.3.2.2. Particle Filter

Although the EKF and UKF are computationally efficient approximations to optimal Bayesian filtering, their accuracy is somewhat limited by the validity of these approximations within the specific tracking scenarios. If sufficient computational resources are available, increased accuracy can be obtained by attempting a numerical approximation to the posterior conditional state density instead of an analytic approximation. Moreover, the accuracy of such numerical approximations is agnostic to other assumptions being satisfied concerning the additive Gaussian measurement/process noise and the Gaussian prior density.

**Particle Filter**, as a specialized Sequential Monte-Carlo (SMC) technique, is one such approach that seek a discrete approximation to the posterior density, via a set of weighted samples. These samples or particles \(x^i\) can be regarded as a point-mass approximation to the posterior density as:

\[
p(x) \approx \frac{1}{N} \sum_{i=1}^{N} \delta(x - x^i) 
\]

The discrete support within this approximation is chosen stochastically rather than deterministically which makes the filter implementation much simpler as no special rules
need to be engineered for determining these points for applying the recursion within each scan. The sampling procedure within particle filtering will automatically move this discrete set of sample points to the region of state-space of interest in representing the state pdf.

Particle filters implement the prediction/filtering steps of the Bayesian recursion directly using a genetic type mutation-selection particle algorithm. Each of the state density within the recursion is represented by a set of \((x^i_k, w^i)\) samples within the state-space. Particle filters are most commonly formulated as Sequential Importance Resampling (SIR) methods. These SIR methods involve drawing samples from an importance density \(q(\cdot)\) which is specifically chosen to be easily sample-able. These samples are then used to update the current sample-set representing the posterior density. Due to appearance of weight degeneracy issues in this sampling approach, whereby few samples/particles become significantly weighted while majority of samples have negligible weights, a resampling stage is introduced at the end of recursion. This goal of this resampling step is to replace the particles with negligible weights by particles with higher weights in a stochastic manner.

Though their exists numerous designs of the particle filter in literature, we present the simplest Bootstrap particle filter for STT to illustrate basic working principles of particle filtering. This filter is characterized by an importance density of \(q(x_k, t) = w^i_{k-1} p_{v_k}(x_k - f(x^i_{k-1}))\). Following algorithm briefly expresses this filter recursion:

```
Algorithm 1 Bootstrap Particle Filter algorithm
1: for i=1,2,...,n do
2: Draw a mixture index \(t^i\) such that \(Pr(t^i == l) = w^i_{k-1}\)
3: Draw \(v^i_k \sim p_{v_k}(\cdot)\) and compute the sample object state \(x^i_k = f(x^i_{k-1}) + v^i_k\)
4: Compute the weight update \(e^i_k = p_{w_k}(z_k - h(x^i_k))\)
5: end for
6: Compute the updated weights: \(w^i_k = \frac{w^i_{k-1} e^i_k}{\sum_{j=1}^{n} w^j_{k-1} e^j_k}\)
7: Compute a state estimate: \(\hat{x}_k|k = \sum_{i=1}^{n} w^i_k x^i_k\)
```

2.4. Multiple Target Tracking

In the multi-target tracking problem, the number of targets to be tracked are unknown apriori and stochastically varies with time. At the sensor, a random number of measurements is received due to detection uncertainty and false alarms. Consequently, standard Bayesian filtering techniques as already introduced in this chapter are not directly applicable, since it is not known which of the received measurements, if any, should be used to update which target state, if any, at each sensor scan.

Conventional solutions have tried to address this problem by logically building on the existing base of single tracking target filters as described in previous section, thus resulting in comparatively more advanced yet still intuitively appealing adaptations. In essence, these
approaches are primarily based on estimating the data associations followed by the application of standard Kalman filtering recursion. Comprehensive treatments of these techniques can be found in [3, 5, 6]. A brief overview of such techniques follows:

2.4.1. Global Nearest Neighbour Filter

The Global Nearest Neighbour (GNN) filter is an extension of the NN filter to the problem of MTT. After computing the state prediction density for each of the currently tracked targets, the GNN tackles the data association problem by solving an optimization problem. This problem is designed to search for a joint association of received measurements to target tracks while minimizing the overall cost, e.g. the individual measurement likelihood factor, under the constraint that a measurement can only be associated to at most one target. Like the NN filter, the GNN filter then simply uses this association within the update step of the recursion to compute the posterior densities of all the targets’ states. Naturally, the GNN filter also suffers from NN limitations and consequently performs poorly in most MTT problems of interest, but from an implementation point of view, it provides the benefits of being intuitive and quite simple to carry out.

2.4.2. Joint Probabilistic Data Association Filter

The Joint Probabilistic Data Association (JPDA) filter [3] is an extension of the PDA filter to the MTT scenario in which all the targets to be tracked are known in advance and their tally remains fixed throughout the surveillance. It can deal better with multiple targets in clutter and dense target scenarios than GNN filter because it does not perform hard data associations. JPDA recursion propagates the targets’ states in a similar manner to PDA filter differing only in the computation of the association probabilities.

The JPDA filter uses joint association events and joint association probabilities in order to avoid conflicting measurement to track assignments in the presence of multiple targets. Then each track is updated with its own weighted innovation consisting of fractions of all measurements. As can be readily seen, the complexity of the calculation for joint association probabilities grows exponentially with the number of targets and the number of measurements. This makes JPDA algorithm generally infeasible practically and several approximation approaches have been proposed that aims to overcome its computation complexity.

2.4.3. Multiple Hypothesis Tracking Filter

The Multiple Hypothesis Tracking (MHT) filter [5] is a deferred decision approach to data association based MTT. The MHT filter mitigates association uncertainty at the current time-step by searching over all previous time-steps for all possible combinations of measurements to target associations that are likely to constitute target tracks/trajectories. Such an exhaustive association of all received measurements (past and present) to either a single track or clutter is referred to as a hypothesis.

The fundamental idea of MHT is to delay difficult data association decisions until more information is received. This is accomplished by maintaining a set of different association hypotheses in a ranked fashion as per their association probabilities. When a new set of
measurement arrives, a new set of hypotheses is correspondingly created from these existing hypotheses and their posterior probabilities are updated using Bayes' rule. Thus, the basic idea in MHT is to propagate the hypotheses with high posterior probabilities, and then seek the hypothesis with the highest posterior probability, at each time-step, to carry out the Bayesian filtering step which can simply be accomplished by employing a standard Kalman filter to estimate individual target states. Note that in the generation of new hypotheses, a measurement can be assigned either to clutter, an existing track or a completely new track. Hence, this allows MHT filter to accommodate unknown and time-varying number of tracks, while tracking. This overcomes the limitations found in the GNN or the JPDA filter.

The combinatorial nature of MHT is its biggest limitations since the total number of possible hypotheses increases exponentially with time resulting in quick exhaustion of memory and computational resources. To overcome this limitation in practice, traditional implementations do measurement gating as well as some sort of heuristic pruning/merging of these hypotheses to reduce computational requirements. Moreover, stronger approximations, in which only selected hypotheses are calculated and retained, are often employed within these approaches to fight the inherent complexity issues.

2.4.4. Random Finite Set based Filtering

The Random Finite Set (RFS) approach to MTT is an emerging and promising alternative to the traditional association-based methods like JPDA, MHT filters [11]. Pioneered by Mahler [17, 15], as finite-set statistics (FISST), the RFS based MTT filtering can be considered as the first systematic and rigorous approach to Bayesian state-estimation while explicitly avoiding the need of cumbersome association of measurements with targets or tracks. This thesis work explores the use of RFS based techniques to tackle the pedestrian tracking problem in the context of automotive driving. A detailed overview of such techniques follows in chapter 3.
3. RFS based Multi-Target Filtering

The Random Finite Set (RFS) based filtering enables the use of optimal Bayesian estimation framework for MTT scenarios by introducing the concepts of a multi-target state/measurement expressed via random finite sets. This chapter reviews the fundamentals of this RFS theory, and presents a mathematically sound RFS formulation of Bayesian Multi-target filtering; following it up with the introduction of RFS based filters like PHD, CPHD, GLMB and the LMB filters.

3.1. Motivation

As described in chapter 2, the recursive Bayesian estimation provides an optimal solution to STT problem. However, the problem arises if one wants to extend the Bayesian framework to track multiple targets as this requires the extension of these Bayesian approaches to cope with target birth, target death, clutter and missing observations. Although the multi-target state can be seen as a concatenation of individual single-target states, each modeled via a random vector, to carry out the Bayesian estimation, such an analysis quickly becomes computationally infeasible due to the increase of the state dimensionality with the increasing number of targets. The algorithm design then resorts to using sub-optimal approximations which lead to degrading accuracy in tracking performance.

On the other end of the spectrum, the measurement-to-track association based approaches like JPDA or MHT, generally impose Gaussian/linear state-space models to enable the use of Kalman filtering for their recursions. Relying on Kalman filters instead of Particle filters, allow these algorithms to balance out the increase in computational complexity arising as a result of their inherent association calculations. However, this also forms a major limitation of their operation in terms of their tracking accuracy when applied to general MTT scenarios involving non-Gaussian/non-linear characteristics.

The main motivation behind using the random finite set formulation to tackle MTT is in its ability to model the multi-target state as well as the multi-target measurement by Finite Set Statistical (FISST) quantities which act as solid mathematical tools in enabling the generalization of the standard Bayes filter from STT to MTT problems thereby avoiding the above mentioned issues. Specifically, the RFS approach represents the multi-target state as a finite-set of individual single target states which allows to elegantly pose the MTT problem as a dynamic multi-target state estimation problem analogous to STT. Furthermore, RFS based approach provides these capabilities without requiring any explicit measurement-to-track based association analysis prevalent in JPDA or MHT algorithms, thus it becomes very attractive from a practical point of view with relatively high computational efficiency.
3. RFS based Multi-Target Filtering

3.2. Random Finite Set formulation

3.2.1. A Random Finite Set

A Random Finite Set $X$ on state-space $X$ is defined as a random variable taking (unordered) finite-set values in $\mathcal{F}(X)$ (the collection of all finite subsets of $X$). While a random vector is expressed as a random constituent point within the state-space, the RFS is rather a point cloud on the state-space where the number of such constituent points, at any time, are random. Moreover each of these points is itself random, distinct and unordered. Mathematically, this can simply be represented as a mapping from a sample-space $\Omega$ representing all possible possibilities of MTT, to such finite-sets:

$$X : \Omega \mapsto \mathcal{F}(X) \quad (3.1)$$

Probability Density

Like any random variable, an RFS is completely described by its probability density. For any region $S \subseteq X$, the FISST density of RFS $X$ is defined to be a non-negative function $p(X)$ given as:

$$\Pr(X \subseteq S) = \int_S p(X) \delta X \quad (3.2)$$

where the above integral is a FISST set integral [17] given by:

$$\int_S p(X) \delta X = \sum_{i=0}^{\infty} \frac{1}{i!} \int_{S^i} p(\{x_1, x_2, ..., x_i\}) dx_1 dx_2 \cdots dx_i \quad (3.3)$$

Intensity Function

The Intensity Function of an RFS is its probabilistic first moment. This is also referred to as the Probabilistic Hypothesis Density (PHD). It is defined to be a non-negative function $v$ on $X$ whose integral over any region $S \subseteq X$ gives the expected number of elements of its RFS that are in $S$:

$$\mathbb{E}[|X \cap S|] = \int_S v(x) dx \quad (3.4)$$

As seen from its definition above, the local maxima of the RFS’s PHD are points in $X$ with the highest local concentration of expected number of elements of $X$ RFS.

Poisson RFS

An RFS $X$ on $X$ with a given intensity function $v$ is said to be a Poisson RFS if its cardinality ($|X|$) is Poisson distributed with mean $\bar{N} = \int v(x) dx$; and for any finite cardinality, the elements $x$ of $X$ are independently and identically distributed (i.i.d) according to the pdf $v(.) / \bar{N}$. The Poisson RFSs are completely characterized by their PHD functions solely. The pdf of such a RFS is given compactly by:

$$p(X) = \exp^{-\bar{N}} v^X \quad (3.5)$$

where $v^X : \mathcal{F}(X) \mapsto \mathbb{R}, v^X = \prod_{x \in X} h(x)$ with $h^\phi = 1$ by convention.
Bernoulli RFS

A **Bernoulli RFS** on $X$ has a probability $1 - r$ of being empty, $r$ of being a singleton whose only element is distributed according to a probability density $p(.)$ on $X$. The cardinality distribution of a Bernoulli RFS is a Bernoulli distribution parametrized by $r$. The pdf of such a RFS is given as:

$$p(X) = \begin{cases} 
(1 - r), & X = \emptyset \\
r p(x), & X = \{x\}
\end{cases}$$

Multi-Bernoulli RFS

A **multi-Bernoulli RFS** on $X$ is a union of a fixed number of independent Bernoulli RFSs $X^i$ with existence probability $r^i \in (0, 1)$ and probability density $p^i$ on $X$. A multi-Bernoulli RFS is thus completely described by the multi-Bernoulli parameter set $\{(r^i, p^i)\}_{i=1}^{M}$. The mean cardinality of such an RFS is $\sum_{i=1}^{M} r^i$. Moreover, its pdf is given by:

$$p(X = \emptyset) = \prod_{j=1}^{M} (1 - r^j) \quad (3.6)$$

$$p(X = \{x_1, x_2, ..., x_n\}) = n! p(\emptyset) \sum_{\{i_1, i_2, ..., i_n\} \in \mathcal{T}_n(\{1, 2, ..., M\})} \prod_{j=1}^{n} \frac{r^{i_j} p^{i_j}(x_j)}{1 - r^{i_j}} \quad (3.7)$$

where $\mathcal{T}_n(X)$ denotes the collection of finite subsets of $X$ with exactly $n$ elements.

### 3.2.2. Multi-Target Bayes Filter

In an MTT scenario, suppose that, at time $t_{k-1}$, there were $M(k-1)$ targets having states $(x_{k-1}^1, x_{k-1}^2, ..., x_{k-1}^{M(k-1)})$ with each $x_{k-1}^i \in X$. At $t_k$, some of these targets may die; the surviving targets evolve to their new states; and new targets may appear. This results into $M(k)$ targets having new states as $(x_k^1, x_k^2, ..., x_k^{M(k)})$. Similarly, at the sensor, suppose that $N(k)$ measurements $(z_k^1, z_k^2, ..., z_k^{N(k)})$ where each $z_k^i \in Z$ are received at $t_k$. The origins of these measurements are not known, and thus the order in which they appear in this representation bears no significance. Generally, only some of these measurements are actually generated by targets. Even in the ideal case where the sensor observes all targets and receives no clutter, STT filtering techniques are not applicable since there is no information about which targets generated which observation as referred as the data-association problem earlier.

Since there is no specific ordering on the respective collections of target states and measurements at $t_k$, they can be aptly represented as finite-sets, as shown below:

$$X_k = \{x_k^1, x_k^2, ..., x_k^{M(k)}\} \in \mathcal{T}(X) \quad (3.8)$$

$$Z_k = \{z_k^1, z_k^2, ..., z_k^{N(k)}\} \in \mathcal{T}(Z) \quad (3.9)$$

The key idea behind RFS formulation is to refer to these sets $X_k$ and $Z_k$ as the **multi-target state** and **multi-target measurement** respectively. Using these concepts, the MTT problem can then be
3. RFS based Multi-Target Filtering

posed as a Bayesian estimation problem with $\mathcal{F}(X)$ state-space and $\mathcal{F}(Z)$ observation-space.

In a STT context, uncertainty within the system is characterized by modeling the state $x_k$ and measurement $z_k$ as random vectors. Analogously, in the MTT context, such uncertainties are characterized by modeling multi-target state $X_k$ and multi-target measurements $Z_k$ as RFSs. As introduced in earlier section, these RFSs can be completely described by a discrete pdf expressing the RFS’s cardinality; and a family of joint probability densities expressing the joint probabilities of its elements on the state-space $X$.

Multi-Target State

For a given multi-target state $X_{k-1}$ at $t_{k-1}$, each $x_{k-1} \in X_{k-1}$ either continues to exist at $t_k$ with survival probability $p_{S,k}(x_{k-1})$ or dies with probability $(1 - p_{S,k}(x_{k-1}))$. Consequently, for a given state $x_{k-1}$ at $t_{k-1}$, its behavior at the next time-step $t_k$ is modeled as being a RFS $S_{k|k-1}(x_{k-1})$ that can take on either $\{x_k\}$ when the target survives, or $\phi$ when the target dies. A new target born at $t_k$ is similarly modeled by an RFS $\Gamma_k$.

The multi-target state $X_k$ at $t_k$ can be written as:

$$X_k = \bigcup_{\zeta \in X_{k-1}} S_{k|k-1}(\zeta) \cup \Gamma_k \tag{3.10}$$

Multi-Target Measurement

The RFS measurement model, which accounts for detection uncertainty and clutter, can be extracted in a similar fashion. A given target $x_k \in X_k$ at $t_k$ is either detected with detection probability $p_{D,k}(x_k)$ or missed with probability $(1 - p_{D,k}(x_k))$. Consequently, at each $t_k$, each state $x_k$ generates a measurement RFS $\Theta_k(x_k)$ that can take on either $\{z_k\}$ when the target is detected, or $\phi$ when the target is missed by the sensor. In addition to these target originated measurements, the sensor also receives a set of clutter measurements modeled via $K_k$ RFS. Thus given a multi-target state $X_k$ at $t_k$, the multi-target measurement $Z_k$ can be written as:

$$Z_k = \bigcup_{x \in X_k} \Theta_k(x) \cup K_k \tag{3.11}$$

Multi-Target Bayesian Recursion

In a similar fashion to STT Bayes filtering, the multi-target state-transition densities $p(X_k|X_{k-1})$ and the multi-likelihood function $(p(Z_k|X_k))$ can be derived from the underlying physical models of targets and sensors using FISST techniques. Assuming their availability, the multi-target Bayes filter propagates the multi-target posterior state-conditional density $p(X_k|Z_k)$ via the familiar prediction-update mechanism as follows:

$$p(X_k|Z_{k-1}) = \int p(X_k|X)p(X_{k-1}|Z_{k-1})\delta X \tag{3.12}$$

$$p(X_k|Z_k) = \frac{p(Z_k|X_k)p(X_k|Z_{k-1})}{\int p(Z_k|X_k)p(X_k|Z_{k-1})\delta X} \tag{3.13}$$
where the integrals in the recursion are FISST set-integrals as introduced earlier.

### 3.2.3. RFS Filters

The recursion in (3.12),(3.13) involves multiple integrals on \( \mathcal{F}(X) \), which can get computationally intractable even with moderate number of targets. SMC implementations can be found in literature approximating these integrals, but still they can pose extreme computational requirements due to the combinatorial nature of the densities. To alleviate the computational intractability of multi-target Bayes filtering, various approaches to approximate it are proposed. A brief overview of such approaches follows:

#### 3.2.3.1. Probability Hypothesis Density Filter

The Probability Hypothesis Density filter (PHD) [16] is a computationally inexpensive approximation of the multi-target Bayes filter. Instead of propagating the full multi-target posterior density in time, the PHD filter relies on propagating the posterior intensity function i.e the posterior PHD of the multi-target state \( X \) during each recursion. The key idea behind this approximation is based on the fact that the local maxima of the PHD \( v(.) \) are points in \( X \) with the highest local concentration of expected number of elements contained in \( X \) and hence can be used to generate estimates for the number of targets in the surveillance environment. In practice, this is generally accomplished by rounding \( \hat{N} = \int v(x)dx \). The individual target state estimates are chosen to be the \( x \in X \) points generating highest \( \hat{N} \) peaks of this intensity function.

[16] provides the derivations for the PHD filter recursion under the following assumptions regarding the MTT context:

- Each target evolves and generates observations independently of one another.
- Clutter is Poisson and independent of target-originated measurements.
- The predicted multi-target RFS governed by \( p(X_k|Z^{k-1}) \) is Poisson.

These assumptions are normally taken to be standard assumptions for tracking problems and generally hold for most tracking scenarios of interest. There exists several advanced forms of PHD filter though, which deals with scenarios where these assumptions are found hard to satisfy thus providing superior performance. For the sake of brevity, we present the PHD recursion for computing the posterior PHD, under those assumptions, in the following:

\[
v_k|k-1(x) = \int p_{S,k}(\zeta)p(x|\zeta)v_{k-1}(\zeta)d\zeta + \gamma_k(x) \tag{3.14}
\]

\[
v_k(x) = [1 - p_{D,k}(x)]v_{k|k-1}(x) + \sum_{z \in Z_k} \frac{p_{D,k}(x)p(z|x)v_{k|k-1}(x)}{k_k(z) + \int p_{D,k}(\xi)p(z|\xi)v_{k|k-1}(\xi)d\xi} \tag{3.15}
\]

As shown from this recursion, the PHD filter recursion only involves integrals on \( X \) thus avoiding the computationally expensive integrals on \( \mathcal{F}(X) \) as in the case of Bayes multi-target filter.
3. RFS based Multi-Target Filtering

Gaussian Mixture - Probability Hypothesis Density filter

The PHD filter recursion shown in (3.14), (3.15) does not admit closed-form solutions in general because of the presence of non-exact integrals to enable the use of computational procedure based algorithm. One may argue for using numerical integration but it suffers from the curse of dimensionality and hence does not scale well with the number of targets or state components.

However, [21] have proposed a closed-form PHD recursion by further imposing linear/Gaussian state-space constraints on the MTT context. In other words, they have proved that under linear target dynamics, linear sensor measurement model, additive Gaussian process and measurement noise, the PHD recursion (3.14), (3.15) can compactly be represented as a closed-form solution just as Kalman filter approaches the optimal recursive Bayes filter for STT problem. [21] called this closed-form solution the Gaussian Mixture - Probability Hypothesis Density filter (GM-PHD filter).

To derive the GM-PHD recursion, in addition to the PHD filter assumptions, the MTT context must also satisfy:

- Each target follows a linear Gaussian dynamical model and the sensor has a linear Gaussian measurement model

\[ p(x|\zeta) = N(x; F_k \zeta, Q_k) \]  
\[ p(z|x) = N(z; H_k x, R_k) \]

- The survival and detection probabilities are state independent

\[ p_{S,k}(x) = p_{S,k} \]  
\[ p_{D,k}(x) = p_{D,k} \]

- The PHD or the intensity function of birth RFS \( \gamma_k \) is a Gaussian Mixture (GM) of the form

\[ \gamma_k(x) = \sum_{i=1}^{J_{\Gamma,k}} w_{\Gamma,k}^i N(x; m_{\Gamma,k}^i, P_{\Gamma,k}^i) \]

Under these assumptions, [21] have shown that the predicted posterior PHD as well the posterior state PHD at any \( t_k \) is also a Gaussian Mixture. Specifically at \( t_k \), if the prior PHD is expressed as a GM of the form

\[ v_{k-1}(x) = \sum_{i=1}^{J_{k-1}} w_{k-1}^i N(x; m_{k-1}^i, P_{k-1}^i) \]

then the GM-PHD recursion can be given by:
3.2. Random Finite Set formulation

- Prediction:

\[
v_{k|k-1}(x) = v_{S,k|k-1}(x) + \gamma_k(x) \tag{3.22}
\]

\[
v_{S,k|k-1}(x) = p_{S,k} \sum_{j=1}^{J_{k-1}} w_{k|k-1}^j N(x; m_{S,k|k-1}^j, P_{S,k|k-1}^j) \tag{3.23}
\]

\[
m_{k|k-1}^j = F_{k-1} m_{k-1}^j \tag{3.24}
\]

\[
P_{S,k|k-1}^j = Q_{k-1} + F_{k-1} P_{k-1} F_{k-1}^T \tag{3.25}
\]

- Update:

\[
v_k(x) = (1 - p_{D,k}(x)) v_{k|k-1}(x) + \sum_{z \in Z_k} v_{D,k}(x; z) \tag{3.26}
\]

\[
v_{k|k-1}(x) = \sum_{j=1}^{J_{k|k-1}} w_{k|k-1}^j N(x; m_{k|k-1}^j, P_{k|k-1}^j) \tag{3.27}
\]

\[
v_{D,k}(x; z) = \sum_{j=1}^{J_{k|k-1}} w_{k|k}^j(z) N(x; m_{k|k}^j, P_{k|k}^j) \tag{3.28}
\]

\[
w_{k}^j(z) = \frac{p_{D,k} w_{k|k-1}^j q_{k}^j(z)}{k(z) + p_{D,k} \sum_{l=1}^{J_{k|k-1}} w_{k|k-1}^l q_{k}^l(z)} \tag{3.29}
\]

\[
q_{k}^j(z) = N(z; H_k m_{k|k}^j, R_k + H_k P_{k|k-1} H_k^T) \tag{3.30}
\]

\[
m_{k|k}^j(z) = m_{k|k-1}^j + K_{k}^j(z - H_k m_{k|k-1}^j) \tag{3.31}
\]

\[
P_{k|k}^j = [I - K_{k}^j H_k] P_{k|k-1}^j \tag{3.32}
\]

\[
K_{k}^j = P_{k|k-1}^j H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \tag{3.33}
\]

As shown by these equations, the GM-PHD provides a computationally efficient mechanism to propagate the posterior PHDs of multi-target state \(X_k\) as a GM involving computations of only means and covariances of each of its components. Though with the passage of time, the GM-PHD filter suffers from computation problems associated with the increasing number of such Gaussian components. In practice this problem is dealt by carrying out special pruning procedures as part of each recursion to remove insignificant or negligible Gaussian components based on some pre-designed criterion.

3.2.3.2. Cardinalized Probability Hypothesis Filter

The Cardinalized Probability Hypothesis Density (CPHD) filter [14] is a generalization of the PHD filter that offers superior tracking performance by jointly propagating the PHD \(v_k(.)\) and the cardinality distribution \(\rho_k(.)\) of multi-target state \(X_k\) to provide better performance albeit at a higher computational complexity.
3. RFS based Multi-Target Filtering

The CPHD prediction is the same as the GM-PHD prediction (3.14) except for the additional calculation of the predicted cardinality distribution $\rho_{k|k-1}$ which is given by the convolution of cardinality distributions of birth RFS $\rho_{\tau, k}$ and surviving target RFS $\rho_{s, k|k-1}$:

$$\rho_{k|k-1}(n) = \sum_{j=0}^{n} \rho_{\tau, k}(n - j)\rho_{s, k|k-1}(j)$$  (3.34)

where

$$\rho_{s, k|k-1}(j) = \sum_{l=j}^{\infty} C_l^j \rho_{s, k|k-1}(1 - \bar{P}_{s, k|k-1})^{l-j} p_{k-1}(l)$$  (3.35)

$$\bar{P}_{s, k|k-1} = \left\langle P_{s, k|k-1}, v_{k-1} \right\rangle / \left\langle 1, v_{k-1} \right\rangle$$  (3.36)

$$\left\langle a, b \right\rangle = \int a(x)b(x)dx$$  (3.37)

The CPHD update is given by:

$$\rho_k(n) = \left\langle \gamma_k^{(0)}[v_{k|k-1}, Z_k](n)\rho_{k|k-1}(n) \right\rangle$$  
$$\left\langle \gamma_k^{(0)}[v_{k|k-1}, Z_k], \rho_{k|k-1} \right\rangle$$  (3.38)

$$v_k(x) = \left\langle \gamma_k^{(1)}[v_{k|k-1}, Z_k], \rho_{k|k-1} \right\rangle (1 - P_{D,k}(x))v_{k|k-1}(x)$$  
$$\left\langle \gamma_k^{(0)}[v_{k|k-1}, Z_k], \rho_{k|k-1} \right\rangle$$  (3.39)

$$+ \sum_{z \in Z_k} \left\langle \gamma_k^{(1)}[v_{k|k-1}, Z_k - \{z\}], \rho_{k|k-1} \right\rangle \psi_k(z; x)v_{k|k-1}(x)$$  
$$\left\langle \gamma_k^{(0)}[v_{k|k-1}, Z_k], \rho_{k|k-1} \right\rangle$$  (3.40)

where

$$\gamma_k^{(n)}[v; Z](n) = \sum_{j=0}^{\min(|Z|, n)} (|Z| - j)\rho_{K,k}(|Z| - j)\rho_{k|k-1}^{p_n}$$  
$$\times \left\langle 1 - p_{D,k}, v \right\rangle^{n-(j+u)} e_j(\Xi_k(v, Z))$$  (3.41)

$$e_j(Z) = \sum_{S \subset Z, |S| = j} \prod_{\zeta \in S} \zeta$$  (3.42)

$$\psi_k(z; x) = \left\langle 1, K_k \right\rangle \frac{g_k(z|x)p_{D,k}(x)}{K_k(z)}$$  (3.43)

$$\Xi_k(v, Z) = \left\{ v, \psi_k(z; x) : z \in X \right\}$$  (3.44)

Multi-target state estimation after the recursion for the CPHD filter is similar to that for the PHD filter where the number of targets are estimated using $\arg\max \rho_k(\cdot)$. The CPHD recursions like
the PHD ones, are also intractable since they also encounter the curse of dimensionality. However, under the assumptions used to come up with closed-form PHD recursive filters aka GM-PHD filters, a closed-form of CPHD filters can also be achieved. Interested readers are encouraged to refer to [27] for recursion steps of such closed-form solutions.

### 3.2.3.3. Multi-Bernoulli filters

In addition to the PHD/CPHD filters, other RFS based approximations of the multi-target Bayes filters include Multi-Bernoulli filters and their various extensions. The idea of Multi-Bernoulli filter was first proposed by Mahler [18] proposing a novel multi-target multi-Bernoulli (MeMBer) recursion as a tractable approximation to the recursive Bayes multi-target filter under low-clutter density scenarios, whereby propagating a multi-Bernoulli RFS distribution directly as an approximation to the posterior multi-target state \( X_k \) recursively instead of posterior PHDs.

However, [30] show analytically that Mahler’s MeMBer filter overestimates the cardinality, and propose a new variant called the **Cardinality Balanced Multi-target Multi-Bernoulli** (CBMeMBer) filter. The CBMeMBer filter extracts the cardinality bias of MeMBer filter in the update step and uses this to develop an unbiased update at the end of its recursion. Like the PHD/CPHD filters, [30] have provided closed-form GM based solutions in case of linear/Gaussian state-space models. For general non-linear/non-Gaussian considerations, SMC based implementations have also been provided. Interested readers are encouraged to refer to [30] for a further conceptual understanding of Multi-Bernoulli filters along with the detailed analysis of their prediction and filtering steps.

### 3.2.3.4. Labeled Multi-Bernoulli Filters

While Multi-Bernoulli filters are not formulated to output target tracks, their generalization, referred to as the **Generalized Labeled Multi-Bernoulli** (GLMB) filters [28] have been proposed to overcome this limitation. These filters rely on the notion of labeled RFSs for their working principles. A brief overview of labeled RFS follows for better understanding this family of filters:

**Labeled RFS**

In order to enable the estimation of an objects trajectory in MTT scenarios, a label \( l \in L \) is appended to a given state \( x \in X \). Generally, these labels \( l \) are elements of a discrete label-space \( L = \{\alpha_i : i \in N\} \) where \( N \) represent the set of positive integers; and all elements \( \alpha_i \) are distinct. So, a labeled RFS is defined to be a finite-set valued random variable on the space \( X \times L \) where all realizations of the label-space contain unique labels.

As introduced in [28], a label \( l = (k, i) \) is usually assigned to each target where \( k \) is the time of birth and \( i \in N \) represents a unique index to distinguish between targets born at the same time. This leads to the label-space for new-born targets at any \( t_k \) to be \( L_k = k \times N \). Since the label-sets for new-born and already existing targets are disjoint, the label-space for all targets at \( t_k \) can conveniently be represented as a concatenation of these label-spaces as \( L_{0:k} = L_{0:k-1} \cup L_k \). We represent this label-space conveniently as \( L \) in what follows. Since the labels of a realization \( X_k \) of a labeled RFS \( X \) are required to be distinct, the cardinalities of the set of labels and the set of state vectors have to be equivalent, i.e if a labeled RFS realization has a label-set given by
3. RFS based Multi-Target Filtering

\[ \mathcal{L}(X_k) = \{ \mathcal{L}(x) : x \in X_k \} \text{ where } \mathcal{L}(x) = (l \in \mathcal{L}) = (k, i) \text{ then } |\mathcal{L}(X_k)| = |X_k|. \]

Mathematically, this is ensured using the distinct label indicator as:

\[ \Delta(X) = \delta_{|X'|(|\mathcal{L}(X)|)} \]  

(3.46)

To understand the notion of GLMB RFS based filtering, the concept of association map has to be introduced in the context of labeled RFS. An association map at any \( t_k \) is a function \( \theta : \mathcal{L} \rightarrow \{0, 1, \ldots, |Z|\} \) such that \( \theta(l) = \theta(l') \) implies \( l = l' \). Such a function can be regarded as a unique assignment of labeled targets to measurements, with undetected targets assigned a 0. The set of all such association maps is denoted as \( \Theta_k \); while the subset of association maps with domain \( \mathcal{L} \) is denoted by \( \Theta_k(\mathcal{L}) \).

**Generalized Labeled Multi-Bernoulli RFS**

A Generalized Labeled Multi-Bernoulli (GLMB) RFS \[29\] is a labeled RFS with state-space \( X \times \mathcal{L} \) distributed according to:

\[ p(X) = \Delta(X) \sum_{c \in C} w^c(\mathcal{L}(X))[p^c]^X \]  

(3.47)

where \( C \) is a discrete index set. The weights \( w^c(L) \) and the spatial distributions \( p^c \) satisfy the normalization conditions:

\[ \sum_{L \in \mathcal{L}} \sum_{c \in C} w^c(L) = 1 \]  

(3.48)

\[ \int p^c(x, l)dx = 1 \]  

(3.49)

\( \delta \)-**Generalized Labeled Multi-Bernoulli RFS**

A \( \delta \)-Generalized Labeled Multi-Bernoulli (\( \delta \)-GLMB) RFS \[29\] with state-space \( X \times \mathcal{L} \) is a specialization of GLMB RFS with \( C = \mathcal{T}(\mathcal{L}) \times \Xi \); \( \mathcal{w}^c(L) = \mathcal{w}^{(l, \xi)} \delta_l(L) \); and \( p^c = \mathcal{p}^{(l, \xi)} = \mathcal{p}^{\xi} \). This specialization therefore enables a particular structure on the index space which arises naturally in MTT applications. To summarize, the \( \delta \)-GLMB RFS is defined via its distribution:

\[ p(X) = \Delta(X) \sum_{(l, \xi) \in \mathcal{T}(\mathcal{L}) \times \Xi} \mathcal{w}^{(l, \xi)} \delta_l(\mathcal{L}(X))[\mathcal{p}^\xi]^X \]  

(3.50)

**Labeled Multi-Bernoulli RFS**

Similar to already mentioned Multi-Bernoulli RFS, a Labeled Multi-Bernoulli (LMB) RFS is completely described by the parameter set \( \{ (r^i, p^i) \}_{i=1}^M \). To include the notion of labels, each individual component of this set is appended with a label which for convenience are mapped to
3.2. Random Finite Set formulation

the component indices. Using such labeled notation, the density of an LMB RFS is given by:

\[ p(X) = \triangle(X)w(L(X))p^X \]  
(3.51)

where

\[ w(L) = \prod_{i \in L} (1 - r^i) \prod_{l \in L} \frac{1_{l}(l)^{p^l}}{1 - r^l} \]  
(3.52)

\[ p^X = \prod_{x \in X} p(x, I) \]  
(3.53)

The cardinality distribution of such an LMB RFS is given by:

\[ \rho(n) = \prod_{i \in \mathcal{L}} (1 - r^i) \sum_{L \in \mathcal{F}(n, L)} \prod_{l \in L} \frac{r^l}{1 - r^l} \]  
(3.54)

The structure of (3.51) implies LMB RFS to be a further specialization of GLMB RFSs being a special case of \( \delta \)-GLMB RFS with \( \Xi = \phi \) thus giving only a single feasible set \( I = L(X) \).

Intuitively, the fundamental difference between a GLMB and LMB process is that of a mixture versus single component representation. The sum over \( c \in C \) in the definition of GLMB RFS facilitates the propagation of multiple hypotheses, involving different sets of track labels, arising due to data association uncertainty seen in the Bayes multi-target filtering step. With only a single component as in the definition of LMB process, it is only possible to propagate the uncertainty for a single set of track labels \( L \), although this can be exploited for computational savings.

A note on Labeled Multi-Bernoulli RFS Filtering

[29] have shown that both the GLMB and \( \delta \)-GLMB densities are closed under the multi-target Bayes prediction and update operations. In other words, these filters offer the first exact closed-form solution to the recursive Bayes multi-target filter. While the GLMB filter is closed under Bayesian recursion, it is not clear how its numerical implementation can be accomplished. Meanwhile, the \( \delta \)-GLMB RFS provide a suitable representation to carry out such an implementation. The resulting filter, called the \( \delta \)-GLMB filter, is found to outperform the PHD/CPHD filters while also providing target tracks as outputs. Furthermore, it offers the advantage over PHD/CPHD filters that no special post-processing is required specifically for state-estimation. The only major disadvantage of these filters is that the filtering recursion exhibits an exponential growth in the number of posterior components. This is normally dealt in practice via some sort of pruning mechanisms.

An alternative approach, proposed to overcome this complexity, relies on replacing the \( \delta \)-GLMB RFS formulation with the LMB RFS form. While the number of multi-Bernoulli components grows exponentially for \( \delta \)-GLMB filter, the growth is linear when using LMB filters in application to the same problem. LMB filters do however incur some expected loss in tracking performance because of its non-exact Bayesian recursion; but for MTT contexts comprising large number of targets, its computational efficiency benefits might outweigh these limitations. As explained in chapter 4, this thesis work specifically explores the use of LMB filters to carry out pedestrian tracking in the context of ADAS systems. Consequently, only the LMB filter, from the general class of multi-Bernoulli RFS based filters, is explained further in this chapter.
3. RFS based Multi-Target Filtering

3.3. The LMB Filter

Using LMB RFSs, we can exploit the intuitive mathematical structure of the multi-Bernoulli RFS without their obvious limitations like extraction of target tracks, resulting in the innovation of *Labeled Multi-Bernoulli* (LMB) filters, as proposed by [24]. The premise of LMB filters is the approximation of multi-target predicted and posterior densities by LMB RFSs which represent each track by an existence probability \( r \) and the corresponding spatial distribution \( p(.) \). Compared to the more general \( \delta \)-GLMB filter, the LMB filters requires a significantly smaller number of components. Compared to the multi-Bernoulli filters, the LMB approximation of Bayes recursion is more accurate, since it only requires the approximation of the posterior \( \delta \)-GLMB RFS by an LMB RFS with matching PHD function. In contrast, the multi-Bernoulli filters involve two approximations of the probability generating functional of the multi-object posterior density.

3.3.1. LMB Filter Prediction

As given in Proposition 2 in [29], the multi-target prediction LMB RFS density is closed under the Bayesian prediction step. Assume that the prior density is an LMB RFS with state-space \( X \times \mathcal{L} \) and parameter set \( \{r^l, p^l\}_{l \in \mathcal{L}} \) and the birth density is an LMB RFS with state space \( X \times \mathcal{B} \) with parameter set \( \{r^l_B, p^l_B\}_{l \in \mathcal{B}} \), i.e,

\[
p(X) = \Delta(X) \omega(\mathcal{L}(X)) p^X
\]

\[
p_B(X) = \Delta(X) \omega_B(\mathcal{L}(X)) p^X_B
\]

where

\[
w(\mathcal{L}) = \prod_{l \in \mathcal{L}} (1 - r^l) \prod_{l \in \mathcal{L}} \frac{1 - p^l}{1 - r^l}
\]

\[
w_B(\mathcal{L}) = \prod_{l \in \mathcal{B}} (1 - r^l_B) \prod_{l \in \mathcal{L}} \frac{1 - p^l_B}{1 - r^l_B}
\]

such that the label-spaces of existing targets and new-born targets are distinct \( \mathcal{L} \cup \mathcal{B} = \phi \). Then [29] show that the predicted LMB density with state-space \( X \times \mathcal{L} \) with \( \mathcal{L} = \mathcal{B} \cup \mathcal{L} \) is given by the parameter set:

\[
p_+(X) = \{(r^l_{+,x}, p^l_{+,x})\} \cup \{(r^l_{+,b}, p^l_{+,b})\}
\]

where

\[
r^l_{+,x} = \eta_S(l) r^l
\]

\[
p^l_{+,x} = \frac{\langle p_S(.,l)p(.,.,l), p(.,l) \rangle}{\eta_S(l)}
\]

\[
\eta_S(l) = \langle p_S(.,l), p(.,l) \rangle
\]

where \( p_S(.,l) \) represents the target survival probability on state-space \( X \) while \( p(x|.,l) \) represents the target’s Markovian transition probability on \( X \). It turns out, that the prediction of an LMB RFS is identical to the prediction of the unlabeled multi-Bernoulli distribution.

\[\text{For notational convenience, we use } \text{“+”} \text{ for predicted components}\]
3.3. The LMB Filter

3.3.2. LMB Filter Update

In contrast to the prediction step, an LMB RFS is not closed under the multi-target Bayes update [29]. Generally the Bayesian measurement-update of a predicted LMB density yields a δ-GLMB density. This density is then approximated by a corresponding LMB RFS which exactly matches the first moment or the PHD of the δ-GLMB RFS. Assume that the predicted LMB RFS with state-space \( \mathcal{X} \times \mathcal{L}_+ \) is given by the LMB parameter set \( p(X) = \{r^l, p^l(x)\}_{l \in \mathcal{L}_+} \), then [29] have shown that the LMB RFS with state-space \( \mathcal{X} \times \mathcal{L}_+ \) and parameter-set:

\[
\begin{align*}
  p(X|Z) &= \{(r^l, p^l(\cdot))\}_{l \in \mathcal{L}_+} \\
  \text{where} \\
  r^l &= \sum_{(I_*, \theta) \in \mathcal{F}(\mathcal{L}_+) \times \Theta} w^{(I_*, \theta)}(Z) 1_{I_+}(l) \\
  p^l(x) &= \frac{1}{r^l} \sum_{(I_*, \theta) \in \mathcal{F}(\mathcal{L}_+) \times \Theta} w^{(I_*, \theta)}(Z) 1_{I_+}(l)p^\theta(x, l)
\end{align*}
\]

(3.63)

exactly matches the PHD or the intensity function of the unlabeled multi-target posterior RFS. The existence probabilities \( r^l \) and spatial distributions \( p^l(\cdot) \) are calculated using:

\[
\begin{align*}
  w^{(I_*, \theta)}(Z) &\propto w_*(I_*)[\eta^\theta_Z(l)]^{I_+} \\
  p^\theta(x, l|Z) &= \frac{p_*(x, l)\psi_Z(x, l; \theta)}{\eta^\theta_Z(l)} \\
  \eta^\theta_Z(l) &= \langle p_*(\cdot, l), \psi_Z(\cdot, l; \theta) \rangle \\
  \psi_Z(x, l; \theta) &= \begin{cases} 
    p_D(x, l)p(z|l) \kappa(z|l), & \text{if } \theta(l) > 0 \\
    q_D(x, l), & \text{if } \theta(l) = 0
  \end{cases}
\end{align*}
\]

(3.64)

(3.65)

(3.66)

(3.67)

(3.68)

(3.69)

where \( p_D(x, l) \) denotes the state-dependent target detection probability; \( q_D(x, l) = (1 - p_D(x, l)) \) is the missed detection probability; \( p(z|l) \) is the single-target likelihood function and \( \kappa(.) \) is the PHD of the clutter RFS modeling clutter.

Due to this LMB approximation, the association uncertainty within the measurement update is represented by the spatial distribution of the individual tracks only while the δ-GLMB filter represents all track labels to measurement associations using hypotheses \( (I, \xi) \). Moreover, the PHD approximation within this derivation of LMB update step results in the loss of information about the cardinality distribution due to the avoidance of these hypotheses. However, the matching PHD do ensure an identical mean cardinality of LMB RFS and the actual mutli-target posterior. Additionally, the spatial distributions of the individual tracks \( l \) are identical in both representations.
4. System Design & Implementation

This chapter describes the main essence or the methodology in coming up with the overall design of the pedestrian tracking system. This is followed by a detailed overview of the techniques and strategies utilized in carrying out the implementation of the overall system.

4.1. System Design

Like any good engineering design, the main focus has been to come up with a modular design approach to overcome the system complexity efficiently while aiding in quick development of the system with each module being designed in an isolated fashion having a clear notion of its input/output interfaces. Figure 4.1 presents a higher level abstracted view of the overall pedestrian tracking system resulting from this approach. We explain these modules briefly in the rest of this section.

![Block diagram of Pedestrian Tracking System](image)

Figure 4.1.: Block diagram of Pedestrian Tracking System

4.1.1. Modules

Sensor

The Sensor module represents an information-capturing device that extracts some useful target motion attributes (of interest) within the surveillance scene. For example, this could be a stereo camera or a lidar sensor etc. giving target motion information at regular intervals of time. Specifically for our project, we have made use of following concepts as sensors:

- **Simulated Sensor Model:** For an extensive evaluation of the implemented pedestrian tracking system (chapter 5), we design a simulation scenario simulating point targets whose motion follows linear/Gaussian characteristics. The specific simulation scenario can be considered as a form of Sensor.

- **Video Frames:** Likewise we make use of MOT Benchmark for further evaluation of the tracking system (chapter 5). However in this case we are provided with camera video
footages or frames comprising different kinds of pedestrian motions. These video frames
then act as Sensor observations.

Detector

The Detector module is responsible for extracting the target-specific information from the
Sensor outputs. Generally this involves coming up with target approximate kinematic quantities
from the sensor scans to feed into the tracker module. For our project, the detector detects the
individual target’s 2D position coordinates within the surveillance region. Specifically we make
use of following detectors:

- **Simulated Detector Model**: For carrying out the simulation scenario, as explained earlier,
  the simulated positions of the target from the Sensor are corrupted with Gaussian noise
to yield simulated detections. Furthermore, these detections are generated as part of
probabilistic process governed by a certain probability of detection $p_d$ which allows for
target miss-detections to help coming up with robust tracker algorithms.

- **Fast Feature Pyramids Detector**: The MOT benchmark provides detection annotations
  (2D position coordinates) on each of their training-video (Sensor) frames. These detections
  are extracted by running the Fast feature pyramids object detector algorithm as proposed by
  P.Dollar et al. in [9].

- **Histogram of Oriented Gradients Detector**: The widely popular OpenCV library [7] for
developing computer-vision applications provides a working implementation of the HOG
detector [8] targeting various platforms like C++, Python, CUDA, OpenCL etc. We run this
library function over MOT frames to come up with the target positions that are then fed into
the tracker module further in the chain.

Tracker

The Tracker module is the most crucial/significant processing element of the overall system
as it is responsible for outputting the target/pedestrian tracks utilizing the detections within every
sensor-scan. It accomplishes this by implementing a suitable tracking algorithm based on the
techniques like the ones presented in chapter 2. As will be explained later, we have employed
mainly two trackers in our work:

- **GM-PHD Tracker**: This tracker implements the GM-PHD filtering recursions to estimate
  the 4D (2D position, 2D velocity) target state. The original GM-PHD filtering algorithm
  [21] is enhanced using the techniques presented in [22] in order to be able to extract not just
  individual target states but rather their trajectories or tracks. The algorithm implementation
  is carried out in C++.

- **SMC-LMB Tracker**: This tracker carries out the implementation of the SMC/Particle-filter
  based LMB filter (chapter 3). The LMB filtering is based on Labeled RFSs which helps to
  extract target tracks from their states automatically. The implementation is carried out in
  C++ as well as using OpenCL acceleration.
4.1. System Design

Analyzer

The Analyzer module is the optional module responsible for analyzing the target tracks being produced from the tracker and compute various evaluation metrics enabling extensive evaluation of the implemented system (chapter 5). Enabling this analysis, the Analyzer module helps in coming up with stable efficient tracking system. Though, using this module could make sense in the development phase, it should be removed from the final system as it provides no core functionality regarding pedestrian tracking.

4.1.2. Interfaces

The main intra-module interfaces carried out in the project as shown in the Figure 4.1 are briefly described below:

- **Sensor Input:** Generally, sensor input consists of the whole surveillance view/region containing targets of interest. For our project, this represents either a simulated scenario or a MOT tracking scenario.

- **Sensor-Detector i/f:** The interface between the sensor and the detector mainly represents the sensor outputs. Further processing tasks could be carried out as part of this interface for helping detector in its algorithm. However, as part of this project we simply forward the sensor output-frames into the detector. The frames are structured as 2D pixel data embedding the target motion information.

- **Detector-Tracker i/f:** This interface mainly represents the target detections which act as input stimuli to the target algorithm. For our work, these detections are in the form of 2D position coordinates (i.e in the form of a 2D floating-point vector).

- **Tracker Output:** This primarily represents the overall output of the whole system. For this project, output involves individual target 4D states (a 4D floating-point vector) along with a specific label (a 2-integer structure) being output from the tracker module within every sensor scan.

4.1.3. System Upgrades

As stated earlier, the proposed modular design methodology serves well to employ a plug & play based design approach whereby one can easily replace an existing module for a better alternative without having to redesign the whole system from scratch thus helping in efficient upgrading of the overall system. Some of the key upgrades in the present pedestrian tracking system could be:

- **Sensor:** Camera/Lidar sensors mounted on an embedded platform being developed at TUM Robotics institute. This would generate real-world sensor data for using the pedestrian tracking system to actual automotive scenarios to evaluate its effectiveness in carrying out its functionality.

- **Detector:** Coming up with a detector algorithm of our own. This implemented detector would then help to do detections on the real-scenario video footages.

- **Tracker:** Further optimization of the implemented tracking algorithm improving both the tracker accuracy as well as its execution performance.
4. System Design & Implementation

4.2. System Implementation

This section describes the major implementation aspects of this thesis work. As mentioned above, the Sensor and Detector are either simulated or are used as is (like a black-box) from the MOT benchmark. The optional Analyzer module would be discussed at length in chapter 5. So this section mainly focuses on the implementation of the Tracker module.

As part of this thesis work, we initially carried out the implementation of the GM-PHD algorithm which initially lacked the ability to output target trajectories. This was overcome by using a tree-based approach to group the GM-terms of a single target together to provide a notion of its trajectory. After extensive simulation analysis (chapter 5), we found the GM-PHD filter accuracy to be inadequate in dealing with general pedestrian tracking scenarios where the pedestrians deviated from linear/Gaussian motion characteristics. This led to the exploration of SMC based approaches to offer better accuracy. In this light, a particle-filter based implementation for LMB filter was successfully carried out. Later via extensive profiling, the LMB filter implementation in C++ was accelerated via OpenCL kernels by spotting the performance bottlenecks and re-implementing them using parallel programming constructs.

4.2.1. GM-PHD Tracker

As mentioned in chapter 3, the GM-PHD filter works by propagating the posterior PHD of the multi-target state in time during each of its recursion. The GM-PHD filter recursion is carried out as shown in Figure 4.2. Each of the block represents a C++ class method performing its specific functionality. The arrows represent the data-flow whereby the GM terms representing the PHDs of multi-target state travel back and forth between the Prediction and the Update modules. Each of the Gaussian term used throughout the filtering operation is compactly represented as a C++ struct with weight, mean, covariance as its attributes. To carry out the linear-algebra matrix operations in C++, we make use of efficient open-source library Armadillo. So while the weight is represented as a C++ float variable, the mean and covariance are better represented via armadillo classes.

At the start of each filter iteration/recursion, the new scan detections are used by the Birth model which compares them to the stored previous-scan detections. Based on likely association or similarities between a specific pair within these consecutive scans, the birth model forms new targets by assigning a new set of Gaussian terms i.e a Gaussian Mixture as part of the predicted GM. These components are then mixed/added with the predicted GM of the surviving targets\(^1\) obtained via (3.24), (3.25).

Similarly in the Update block, the current-scan detections are used alongwith the current computed prediction GM to extract the update GM using (3.31), (3.32). These Update GM terms are then passed through the Prune/Merge block where we use a three-fold strategy to reduce the computational complexity arising from the increasing GM terms. These are summarized as follows:

\(^1\)surviving targets are represented by Update GM in past iteration
• First all the close-by Gaussian terms (via their means) are merged together to form composite Gaussian term as they are thought to represent a single target.

• Then all the Gaussian terms whose weights are less than the filter specified threshold are discarded as insignificant terms and are not processed further. This also allows to gracefully terminate target tracks.

• Finally we keep a cap on the maximum number of Gaussian terms corresponding to the maximum number of expected pedestrians within the surveillance zone.

Figure 4.2.: Block diagram of GM-PHD filter recursion

After pruning, the Update GM represents the posterior PHD of the multi-target state with each Gaussian term representing a possible target state. To avoid tracking clutter terms, a second threshold is used here to discard the Gaussian terms that are not too significant as of current iteration but could lead to greater weights in coming iterations. Such terms are not output as current target states but also not discarded as they are being kept in the surviving target GM to be considered for next iterations. For the significant terms, their means represent the individual
target states and are output as such.

As should be obvious from these outputs, this preliminary filter is only capable of extracting individual target states but does not output tracks or trajectories i.e there is no association between currently obtained states and the past ones. To overcome this limitation, the implementation algorithm is enhanced via tree-based techniques. For the sake of brevity, we ignore here the implementation details of those techniques though. Interested readers are referred to [22] for further details in this regard.

4.3. SMC-LMB Tracker

The GM-PHD tracker provides the optimal PHD recursive solution in case of targets and sensors following linear/Gaussian state-space characterizations. So expectedly its accuracy performance should be deteriorated to a certain degree when the targets exhibit non-linear and/or non-Gaussian motion tendencies. We carry out simulation analysis (chapter 5) to investigate this and find that the tracker performance is severely affected up to a point that the error is too much to tolerate. This led to the exploration of non-linear techniques to overcome this limitation.

SMC based Particle Filters have been a popular approach in this context after being introduced in 1990s. We looked into the possibility of using the particle filter based implementation of the PHD filter in order to make it suitable for tackling generic pedestrian motions. However, we found that, recently a new class of RFS based filters called the LMB filters have been proposed which are deemed to be more accurate than PHD filtering techniques (chapter 3). Furthermore, the SMC based implementations of these filters match with those of PHD filters in terms of computational complexity [25]. So further work in this thesis focuses on implementing the SMC-LMB tracker as the main tracker module in Figure 4.1.

Figure 4.3 presents the overview of the implementation of the SMC-LMB tracker. Structurally it is similar to Figure 4.2 like any Bayesian estimator but computationally there are major differences. First, instead of using GM to represent the posterior states, the SMC-LMB tracker relies on propagating a set of multi-Bernoulli terms, each given by the pair: \( \{ r^i, p^i(\cdot) \} \), where every state pdf \( p^i(\cdot) \) is represented via weighted particles \( \{ w^n, x^n \} \). Similar to the GM-PHD filter implementation, these terms are compactly represented as a C++ struct using standard floats for \( r, w \) while the 4D state is conveniently represented via Armadillo vectors. Moreover, being based on labeled RFSs, each instance of this struct has a unique 2D int label vector that act as a tag for each tracked target. Each of the sub-blocks shown in Figure 4.3 is implemented via C++ functions as member functions of the LMB filter class.

4.3.1. Filter Initialization

This function is executed once for each instance of the tracker class at the time of its construction. Here all the tracker parameters are set like number of particles to represent state-pdfs, maximum number of LMB components allowed to represent posterior multi-target state etc. along with the state-space modeling parameters.
4.3. SMC-LMB Tracker

Figure 4.3.: Block diagram of SMC-LMB Tracker Recursion
4.3.2. Birth Model

Similar to GM-PHD filter implementation, the birth model in SMC-LMB tracker implementation relies on the associations between the measurements obtained in consecutive scans. However, in case of birth of new targets, instead of representing it via a GM, a new multi-Bernoulli term is generated. In the current implementation, we find speeds in the Cartesian space between every detection pair using current and the immediate past detection scan. If the speeds for a specific pair lies within the tracker-parametrized $V_{max}$ value, the pair is deemed to correspond to a single target and hence a new target track is created. This track is initialized via parametrized existence probability $r_B$ while its state-pdf is supposed to be a Gaussian and a certain number of particles are drawn from it stochastically. These number of particles are also parametrized and we recommend them to be within powers of 2 for ease in GPU based particle-level processing.

4.3.3. Predict LMB

This module carries out the LMB prediction. Specifically it generates predict-LMB terms for new-born targets from the birth model as well as survive-LMB terms for existing targets via current Update-LMB terms using (3.59), (3.60). We use two different sets of LMB terms instead of single one as it is much easier to do further conversion into $\delta$-GLMB components separately and then merge them together.

4.3.4. Predict-LMB to Predict-$\delta$GLMB

Both of computed birth predict-LMB as well as survive predict-LMB terms are then converted to their $\delta$-GLMB terms. This step is necessary to allow the $\delta$-GLMB update later in the data-flow. Now, even for moderate number of targets/pedestrians these $\delta$-GLMB could become large and processing them quickly becomes computationally expensive. Like for the case of GM-PHD tracker, we introduce pruning schemes to cap maximum number of components. But in contrast to the former approach, the components have not yet been computed. So to avoid computation of all such components followed by propagation of significant components; we rather formulate this problem as a K-shortest path problem and use computationally efficient Eppstein solution [10] (using Bellman-Ford algorithm [4] internally) to directly compute only the significant components without the need of further pruning. Interested readers are encouraged to read [28] in order to come up with such formulation.

After computing the separate $\delta$-GLMB components for the new-born and existing targets, they are convolved together to give the overall $\delta$-GLMB terms that are to be used for the update phase in the next scan/iteration while the LMB terms are simply concatenated together.

4.3.5. Update LMB intermediate

This is the first step within the Update step of the SMC-LMB recursion. It computes all possible Update LMB terms based on every possible association of the current measurement-set with the previous scan Predict LMB terms using (3.63). These terms would be required later on in conversion of Update $\delta$-GLMB terms to their equivalent LMB terms. Hence these terms are considered intermediate within the recursion.
4.3.6. $\delta$-GLMB Update

This step performs the closed form $\delta$-GLMB update on the Predict $\delta$-GLMB [28] terms as obtained in the previous iteration. Here again we are confronted with the similar problems of rapidly growing terms and have to employ some sort of a cap on maximum number of components to deal with computational complexity. However, because of the measurement involvement this problem is formulated as the $K$-best assignment problem as opposed to K-shortest path problem. To solve this problem, we rely on using Murty algorithm [19] (using Hungarian method internally) as explained in greater detail in [28].

4.3.7. Update LMB

As shown in Figure 4.3, coming up with the Update LMB terms within each tracker iteration involves a two-fold process. First a conversion from Update $\delta$-GLMB terms to corresponding LMB terms is carried out such that the LMB set matches the PHD terms of the $\delta$-GLMB set as was explained in chapter 3. Next the particles needed to represent each of LMB term’s state pdf $p(\cdot)$ are replaced with new set in a commonly used procedure referred to as particle resampling to deal with particle impoverishment problem. The computation of these LMB components complete the SMC-LMB recursion.

4.3.8. Track Management

In contrast to GM-PHD tracker, no special procedures are required to output target tracks as the SMC-LMB filter outputs Update LMB terms containing unique tags i.e outputting target tracks or trajectories directly. Here again the techniques of merging (to combine tracks formed from single target) and pruning (for tracks which we are not yet confident of being either a new target or clutter) are used just like for the case of GM-PHD tracker.

4.3.9. State Estimation

The final step within each tracker iteration is to estimate individual target states and to associate them to already existing tracks/trajectories. For this we use Mahler’s ESF function [28] to first estimate stochastically the cardinality of current multi-target state based on the pruned Update LMB terms. Then a certain number of most weighted/significant components corresponding to this cardinality estimate are chosen for state-estimation. Using the particle representation of these components, an empirical measure is easily derived for each chosen such component.

4.4. OpenCL Acceleration

Being satisfied with the tracking accuracy of the SMC-LMB tracker (chapter 5), an extensive profiling of the above mentioned algorithm in C++ was carried out for the purpose of spotting potential performance bottlenecks. As clear from detailed analysis presented in chapter 5, the primary source of execution performance bottleneck within the recursion is the computation of Update LMB terms, roughly amounting to 75% of the computations. So, in our strategy to improve the execution performance of the algorithm, instead of redesigning the whole algorithm from scratch via programming constructs, we relied rather on a hybrid of C++ and OpenCL.
4. System Design & Implementation

computation code. We transformed the sequential execution of Update LMB function into OpenCL kernels to significantly improve the timing performance of the whole algorithm. The main implementation aspects of this strategy are outlined below:

- Generation of uniform random numbers on GPU itself using the AMD CLRNG compute library.
- Breaking down the for-loops within the Update block down to the level of particle computations.
- Efficient parallel scan (prefix-sum) on the cumulative weights array to carry out the particle resampling procedure.
- Optimized memory organization for the LMB terms throughout the Update part of the recursion as high amount of memory transfers between the host CPU and GPU accelerator severely affects the performance and could possibly outdo the benefits achieved via GPU computations.
- The number of particles allocated for each of the LMB term are chosen to be in powers of 2 which makes it easier to use shared-memory optimizations within GPU computations for further acceleration of the application.
5. Evaluation and Analysis

This chapter presents a detailed evaluation and analysis on the pedestrian tracking system as explained in previous chapters. Primarily, the analysis carried out is two-fold. First we discuss the tracking accuracy performance of the proposed design followed by its execution performance analysis.

5.1. Evaluation Metrics

It is paramount to have a clear notion of evaluation metrics before carrying out the actual evaluation of the system itself. Given below is a brief overview of these evaluation metrics which have been used in this thesis work for providing a detailed evaluation and analysis of the proposed system.

5.1.1. Tracking Accuracy

As mentioned earlier, the multi-target tracking problem attempts to jointly find the number of targets as well as their individual states from the received measurements with the passage of time. Therefore, to quantify such a tracker’s accuracy within this thesis work, we make use of following metrics:

- **Cardinality Estimate** which extracts the number of targets at the end of each tracker recursion. This can then be compared with truth/actual cardinality of the multi-target state to figure out the tracking errors in this respect.

- **Optimal Sub-Pattern Assignment (OSPA) metric** to define a notion of miss-distance and corresponding error between actual and estimated individual target states as proposed firstly by [26]

5.1.2. Execution Performance

For profiling the tracker algorithm’s execution performance, we use a simple mechanism involving the computation of the number of CPU cycles across the algorithm. Specifically, if an algorithm or its specific portion requires \( N \) processor cycles then we get a rough idea of computation or execution time using:

\[
  t_{\text{exec}} = \frac{N}{f_{\text{CPU}}} \tag{5.1}
\]

where \( f_{\text{CPU}} \) represents the CPU frequency or the clock-rate. Using this, in the context of video processing, we can obtain an average measure of frame per second as a computational throughput measure for the tracker via:

\[
  FPS = \frac{\text{Number of frames in video}}{\text{Total execution time}} \tag{5.2}
\]
5. Evaluation and Analysis

5.2. Tracking Accuracy Analysis

This section details this thesis work’s findings regarding the tracking performance of the proposed LMB tracker. In order to evaluate the tracker for providing this analysis, we make use of a simulation analysis as well as using available data-set videos.

5.2.1. Simulation Analysis

For carrying out a extensive analysis for evaluating the tracking accuracy of the implemented LMB tracker, we create a 2D surveillance point-targets based simulation scenario. The target motion dynamics as well as the sensor model are made to follow Linear/Gaussian characteristics. Apart from that, the simulation is designed to be highly parametrized in the number of targets; their birth locations; their birth-times and death-times; their detection and survival probabilities; amount of clutter; process/measurement noise etc. This helps to generated diverse range of simulation scenarios to adequately understand the tracking accuracy performance under the influence of different constraints.

The surveillance region is designed to be a rectangular frame modeling a \([-1000, 1000] \times [-1000, 1000]\) pixel image. The state of each target is given by a 4D vector \(x_k = [x, \dot{x}, y, \dot{y}]\) comprising of position coordinates and the velocity within the surveillance region. Similarly the sensor is assumed to provide observed position of the targets generating 2D measurements \(z_k = [x', y']\). These detected measurements are immersed in clutter that is modeled as a Poisson RFS \(K_k\) with intensity function:

\[
K_k(z) = \lambda_c V u(z)
\]

where \(u(z)\) represents the uniform density over the surveillance region of volume \(V = 4 \times 10^6 m^2\). The parameter \(\lambda_c\) denotes the clutter density which is defined to be the average number of clutter returns within \(z_k\) per scan. For all the simulations, we use the following state-space model:

\[
x_k = F_k x_{k-1} + v_k \quad (5.4)
\]

\[
z_k = H_k x_k + w_k \quad (5.5)
\]

where we choose the state-transition matrix \(F_k\), process noise covariance \(Q_k\), observation matrix \(H_k\) and measurement noise covariance \(R_k\) to respectively be:

\[
F_k = \begin{bmatrix} I_2 & \Delta I_2 \\ 0_2 & I_2 \end{bmatrix}
\]

\[
Q_k = \sigma_v^2 \begin{bmatrix} \frac{\Delta^4}{3} I_2 & \frac{\Delta^3}{2} I_2 \\ \frac{\Delta^3}{2} I_2 & \frac{\Delta^2}{3} I_2 \end{bmatrix}
\]

\[
H_k = \begin{bmatrix} I_2 & 0_2 \end{bmatrix}
\]

\[
R_k = \sigma_w^2 I_2
\]

where \(I_2, 0_2\) are the \(2 \times 2\) identity and null matrices respectively. The \(\Delta\) is the sensor sampling period or the inter scan-time. Finally \(\sigma_v, \sigma_w\) represent the variances within the Gaussian process and measurement noises respectively. Figure 5.1 graphically shows such a simulation setup with target tracks starting at \(\circ\) and terminating at \(\Delta\).
5.2. Tracking Accuracy Analysis

5.2.1.1. Comparison with GM-PHD Tracker

The GM-PHD filter, as mentioned earlier, provides an optimal solution to the multi-target tracking problem under linear/Gaussian state-space models considering the propagation of posterior intensity functions. So, this tracker is expected to show sufficiently high tracking accuracy for the designed simulation scenario. Therefore, we use it as a performance benchmark for evaluating the performance of the LMB tracker. Better analysis, in this regard though, could be carried out using the GM implementation of the LMB tracker as it represents the optimal solution to Linear/Gaussian MTT considering the propagation of actual posterior state.

For the comparison, we design two artificial simulation scenarios, whereby simulating 12 linearly moving targets for a duration of 100 time-steps as shown in Figure 5.1. The first scenario is designed to show accuracy performance of the algorithm under highly ideal tracking conditions while the other scenario poses slightly more challenging conditions. These are parametrized as:

- **Easy Tracking Scenario:**

  \[ p_{S,k} = 0.98 \]
  \[ p_{D,k} = 0.98 \]
  \[ \sigma_v = 1 \text{ m/sec}^2 \]
  \[ \sigma_w = 1 \text{ m} \]
  \[ \Delta = 1 \text{ s} \]
  \[ \lambda_c = 5 \]

- **Hard Tracking Scenario:**
5. Evaluation and Analysis

\[ p_{S,k} = 0.90 \]
\[ p_{D,k} = 0.90 \]
\[ \sigma_v = 5m/sec^2 \]
\[ \sigma_w = 10m \]
\[ \triangle = 1s \]
\[ \lambda_c = 60 \]

Figure 5.2 shows the cardinality-estimate and the OSPA distance measures for the GM-PHD tracker for both of these scenarios. For the GM-PHD tracker, we use a track prune-threshold of \( T = 10^{-5} \); a track merge-threshold of \( U = 4 \); and maximum number of tracks within the multi-target GM to be \( J_{max} = 100 \).

As can be seen from Figure 5.2, the GM-PHD tracker performs apparently perfectly in easy-scenario for estimating the individual target states as illustrated by \( OSPA-Loc \) measure. The

\[ \text{(a) Cardinality Estimate (easy-scenario)} \]
\[ \text{(b) OSPA Distance (easy-scenario)} \]
\[ \text{(c) Cardinality Estimate (hard-scenario)} \]
\[ \text{(d) OSPA Distance (hard-scenario)} \]

Figure 5.2.: Tracking accuracy of GM-PHD Tracker
5.2. Tracking Accuracy Analysis

cardinality estimate, however shows that even in ideal conditions, the tracker does make occasional mistakes. This is attributed to the adaptive birth distribution model, being embedded inside the tracker algorithm, which requires some initial scans to confirm successive measurements of new target in order to confirm it as a new track. This leads to cardinality errors at target births. Avoiding this can lead to tracker performance being severely affected causing it to consider every new detection as a new target which could very well be a clutter detection or a false-alarm. Similarly, when the target dies, the tracker expectedly make mistakes because the algorithm can not be sure about the target disappearance as an actual death or a miss-detection because of sensor imperfection. If one designs the tracker algorithm to abruptly terminate tracks just because of one miss-detection, then tracker performance could suffer drastically. Consider for example the occurrence of a target missed-detection. When such a target is re-detected, then the tracker would consider it to be a new target/track instead of continuing the previous known track. This behavior in most tracking scenarios is undesirable. Furthermore, these plots clearly show the worsening performance of the tracker in dealing with more severe tracking environments.

![Cardinality Estimate (easy-scenario)](image)

![OSPA Distance (easy-scenario)](image)

![Cardinality Estimate (hard-scenario)](image)

![OSPA Distance (hard-scenario)](image)

**Figure 5.3:** Tracking accuracy of SMC-LMB Tracker
5. Evaluation and Analysis

To evaluate the SMC-LMB tracker, we use a configuration of 512 particles, and cap the update $\delta$-GLMB hypotheses and the $\delta$-GLMB components to a maximum of 100. Figure 5.3 presents the corresponding results under the designed simulation scenarios.

It is clear from these plots, that using sufficiently high number of particles/samples (like 512 in our case) to approximate the true posterior density, the LMB tracker shows comparable, if not better performance, than the more suitable GM-PHD tracker for linear/Gaussian systems. Especially under severe tracking scenarios where the targets show considerable deviation from the linear/Gaussian as governed by higher $\sigma_v$ value, these results justify the use of SMC based trackers for MTT tracking, by showing their robustness in dealing with process modeling imperfections.

5.2.1.2. Detailed Analysis

In order to study the tracking behavior of the LMB filter more deeply, we design various simulation scenarios to evaluate the tracker while isolating factors like clutter density, number of targets, noise strength etc. We present the findings in the following:

**Clutter Density**

While keeping other settings fixed\(^1\), we carry out simulation scenarios having a clutter density $\lambda_c$ ranging in 5,50,500,2000. Figure 5.4 shows the corresponding plots. As can be seen from the plots, the tracker accuracy is fairly agnostic to the amount of clutter in the surveillance region. The tracker seems to be affected by the clutter for large $\lambda_c > 1500$ which is obviously too much for most of MTT scenarios.

**Sensor Noise, Miss-detections**

Here we repeat the same procedure but this time considering the probability of target detection $p_D$ and the measurement noise variance $\sigma_w$. Both of these quantities reflect the quality of the sensor in that a high quality sensor is characterized by a small $\sigma_w$, high $p_D$ while a low quality sensor would tend to have a higher $\sigma_w$, lower $p_D$. We carry out the simulation scenarios using $(p_D = 0.98, \sigma_w = 1), (p_D = 0.90, \sigma_w = 10), (p_D = 0.8, \sigma_w = 20)$ configurations. Figure 5.5 show the corresponding results. As with clutter density, the LMB tracker is able to perform accurately even under low sensor quality conditions.

**State-space model deviation from Linear/Gaussian**

As mentioned earlier, the proposed LMB uses the linear/Gaussian state-space model for modeling target dynamics and sensor returns. Hence, the tracker accuracy is expected to be sufficient in dealing with scenarios where targets/sensor actually follow these considerations as shown in the analysis carried out so-far. Here we introduce a parameter $\sigma_v$ for both of these models to cause deviations from this ideal linear/Gaussian context. We generate simulation scenarios using $\text{sigma}_v=1,10,20$ to study how the LMB tracker behaves when its underlying models deviate from the actual scenario.

---

\(^1\)we use easy-scenario configurations for fixed parameters so as not to influence tracker performance from other factors
5.2. Tracking Accuracy Analysis

Figure 5.4: LMB Tracker accuracy: clutter density
5. Evaluation and Analysis

Figure 5.5: LMB Tracker accuracy: sensor imperfections

(a) Cardinality Estimate $p_D = 0.98, \sigma_w = 1$

(b) OSPA Distance $p_D = 0.98, \sigma_w = 1$

(c) Cardinality Estimate $p_D = 0.9, \sigma_w = 10$

(d) OSPA Distance $p_D = 0.9, \sigma_w = 10$

(e) Cardinality Estimate $p_D = 0.8, \sigma_w = 20$

(f) OSPA Distance $p_D = 0.8, \sigma_w = 20$
5.2. Tracking Accuracy Analysis

(a) Cardinality Estimate $\sigma_V = 1$

(b) OSPA Distance $\sigma_V = 1$

(c) Cardinality Estimate $\sigma_V = 10$

(d) OSPA Distance $\sigma_V = 10$

(e) Cardinality Estimate $\sigma_V = 20$

(f) OSPA Distance $\sigma_V = 20$

Figure 5.6.: LMB Tracker accuracy: state-space model deviation
5. Evaluation and Analysis

### Table 5.1: LMB Tracker configurations

<table>
<thead>
<tr>
<th># Particles</th>
<th># Birth-Comps</th>
<th># Survive-Comps</th>
<th># Update-Comps</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>5</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>256</td>
<td>10</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>512</td>
<td>20</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1024</td>
<td>50</td>
<td>200</td>
<td>200</td>
</tr>
</tbody>
</table>

Figure 5.6 shows the corresponding plots. These plots clearly demonstrate that the LMB tracker accuracy is primarily dependent on the quality of the underlying models used. When there is a considerable deviation between the actual target motion and the target motion model, the tracker accuracy performance suffers drastically because of continuous track losses as seen by frequent cardinality under-estimation in these plots with increasing deviation $\sigma_y$.

**LMB Tracker parameters**

The LMB tracker is designed in parametric fashion whereby providing various algorithmic configurations to deal with a variety of tracking scenarios. Here we study the impact of changing its number of particles, birth-components, survive-components, update-components. For this purpose, we use four different configurations (Table 5.1) to tackle with a variant of the hard-scenario using $\lambda_c = 5$.

Figure 5.7 provides the corresponding plots. As expected, the tracking performance steadily improves with increasing number of components albeit at higher computational costs and execution times. Moreover, the plots also show that the accuracy performance improvements in Config4 are pretty minimal compared to Config3 thus exhibiting *diminishing returns* in tracking accuracy.

### 5.2.2. MOT Dataset Analysis

This section presents the tracking accuracy performance of the LMB tracker system on publicly available dataset videos. The goal of the analysis is to further build on the understanding of the tracking accuracy analysis developed via comprehensive simulations and to see whether the simulation results corresponds to pedestrian tracking scenarios in actual video footages.

For this purpose, we use an effective benchmark named *Multiple Object Tracking Challenge* [13]. This benchmark is designed to provide video footages covering diverse MTT contexts where the goal is to track pedestrians accurately. The benchmark is designed as a competition where the current state-of-the-art approaches are ranked as per their tracking accuracy. The pedestrian detections are already provided by the benchmark so the accuracy of tracking directly depicts the efficiency of the tracking algorithm in dealing with various tracking challenges.

For the sake of brevity, in this section we present an evaluation analysis of the LMB tracker on two of the MOT videos as summarized in Table 5.2:
5.2. Tracking Accuracy Analysis

Figure 5.7: LMB Tracker accuracy: tracker parameters
Table 5.2.: MOT Dataset videos

<table>
<thead>
<tr>
<th>Video</th>
<th>Resolution</th>
<th>Number of frames</th>
<th>Unique Targets</th>
<th>Maximum Targets per Frame</th>
<th>Target Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>KITTI-17</td>
<td>1224x370</td>
<td>145</td>
<td>9</td>
<td>4</td>
<td>4.7</td>
</tr>
<tr>
<td>PETS09-S2L1</td>
<td>768x576</td>
<td>795</td>
<td>19</td>
<td>8</td>
<td>5.6</td>
</tr>
</tbody>
</table>

- KITTI-17: Static Camera; Mostly linear target motion
- PETS09-S2L1 Static Camera; Targets move in irregular patterns

Figure 5.8 shows the corresponding accuracy plots when these videos are fed to the LMB tracker for pedestrian tracking. For the KITTI-17 video, the tracker takes some initial frames to confirm target tracks thereby making cardinality errors in the initial frames, but once the tracks are confirmed the tracker performs highly accurately in tracking each of the pedestrian motion. The reason for such small OSPA distances in the later phase of the video can be attributed to the motion characteristics of the pedestrians. In KITTI-17 video, mostly targets move in a straight line i.e in a linear fashion hence the tracker performs expectedly well as per the simulation analysis.

On the other hand, the PETS09-S2L1 video presents a much tougher challenge in that the pedestrians move in irregular patterns like moving abruptly or moving in circles etc. As mentioned earlier, due to the use of linear state-space models internally in current SMC-LMB tracker implementation, the accuracy performance deteriorates. Specifically, due to model deviation from actual pedestrian motion, the tracker keeps on making erroneous predictions and when the corresponding target’s measurement does not tally with this prediction, the algorithm terminates the track as evident by regular cardinality errors in the plots. Also the OSPA distances are relatively high as compared to KITTI-17 plots.

5.2.3. Influence of Detector Accuracy

To analyze the sensitivity of the tracker performance to that of the detector, we use two different detectors to provide detections to the tracker for PETS09-S2L1 video:

- Fast Feature Pyramids for Object Detection as proposed by Dollar et al [9]. These detections are provided by MOT dataset on all of their videos.
- Histogram of Oriented Gradients as proposed by Dalal and Triggs [8]. We use an open-source implementation of the detector provided as part of the OpenCV library, though with untuned parameters.

Figure 5.9 presents the corresponding plots. Comparing with MOT detector plots in Figure 5.8, it can clearly be established that the accuracy of the LMB tracker is highly dependent on the input detections being provided by the Detector module. Using sub-optimal/untuned HOG detector, the LMB tracker continuously loses target tracks resulting in large Cardinality errors as shown in these plots.
5.2. Tracking Accuracy Analysis

Figure 5.8.: LMB Tracker accuracy: MOT dataset videos
5. Evaluation and Analysis

![Graphs showing cardinality and OSPA distance over time]

**Figure 5.9:** LMB Tracker accuracy: HOG Detector

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{B,\text{max}}$</td>
<td>Maximum value for a birth LMB component’s existence probability</td>
</tr>
<tr>
<td>$V_{B,\text{max}}$</td>
<td>Maximum speed expected of pedestrians (pixels/frame)</td>
</tr>
<tr>
<td>$r_{\text{track, max}}$</td>
<td>Threshold for a tentative target track to be confirmed</td>
</tr>
<tr>
<td>$r_{\text{track, min}}$</td>
<td>Threshold for discarding an insignificant track</td>
</tr>
<tr>
<td>$U_{\text{merge}}$</td>
<td>Merge threshold for use in pruning</td>
</tr>
<tr>
<td>$N_{\text{particles}}$</td>
<td>Number of particles to represent each target’s state-pdf</td>
</tr>
<tr>
<td>$H_{\text{birth, max}}$</td>
<td>Maximum number of Birth LMB components</td>
</tr>
<tr>
<td>$H_{\text{survive, max}}$</td>
<td>Maximum number of Survive $\delta$-GLMB hypotheses</td>
</tr>
<tr>
<td>$H_{\text{update, max}}$</td>
<td>Maximum number of Update $\delta$-GLMB hypotheses</td>
</tr>
</tbody>
</table>

**Table 5.3:** LMB Tracker parameters

5.2.4. Guide to set LMB Tracker parameters

As explained in chapter 4, the implemented SMC-LMB tracker offers parametrizability or configurability via its parameters that can be set at the time of instance construction. These parameters are summarized in Table 5.3. In addition to these, there do exists some other parameters as well, but they mostly concern with the state-space model and hence are not generic parameters. After extensive evaluation analysis, we present the following insights as a useful guide to set up these parameters for better tracker performance:

- $r_{B,\text{max}}$ is used by the birth model in generating new target tracks. A larger value of $r_{B,\text{max}}$ results in faster track confirmation but yields higher false tracks as well. While, a lower value results in slower track confirmation but correspondingly less clutter being recognized as actual targets. For most of our analysis, we find $r_{B,\text{max}} = 0.25$ to work well.

- $V_{B,\text{max}}$ is also used by the birth model. Basically it is used to set up a distinguishing factor between clutter and actual target while finding associations between consecutive sensor-scan...
5.3. Execution Performance Analysis

This section carries out the extensive profiling of the implemented SMC-LMB tracker algorithm. First we extensively evaluate the execution performance of the C++ code to spot the potential performance bottlenecks. These are then subjected to OpenCL acceleration to come up with an implementation meeting the real-time constraints. We make use of the MOT videos (Table 5.2) to carry out this analysis.

5.3.1. Sequential (C++) Implementation

For helping in spotting the potential bottlenecks, we split the main C++ algorithm into six parts as listed below:

1. Predict LMB
2. Predict $\delta$-GLMB
3. Update LMB intermediate
4. Update $\delta$-GLMB
5. Update LMB
6. Track-management & State-estimation

detections. A large value of $V_{B, \text{max}}$ permits very fast moving targets to be considered for track initiation but could also lead to false tracks because of clutter. While the opposite happens with low values of $V_{B, \text{max}}$, just like $r_{B, \text{max}}$ trade-off.

- $r_{\text{track, max}}$ is used by the track-management module. Only the Update LMB terms that have their existence probabilities $r > r_{\text{track, max}}$ are considered further for state-estimation i.e they are considered to be confirmed targets/tracks. A large value of $r_{\text{track, max}}$ helps to reduce false tracks but makes tracker response to new tracks slow. A lower value on the other hand improves this tracker response but also results in more false tracks resulting from clutter.

- $r_{\text{track, min}}$ is also used by the track-management module for track termination. Here, a large value of this parameter means a relatively strict policy in case of discarding target tracks. For severe tracking conditions, it is better to have a much lower value so as not to continually lose a target from its track.

- $U_{\text{merge}}$ is also used by the track-management module for merging close-by Update LMB components. This parameter controls the resolution of the tracker in forming two close-by tracks. A larger value would reduce this resolution but also reduces the possibility of two tracks emerging from the same target while a smaller value would be help to do the opposite.

- The rest of the parameters in Table 5.3 corresponds to Table 5.1 and their impact on tracker accuracy performance has already been extensively studied before.
5. Evaluation and Analysis

Using the CPU clock-cycle metric as mentioned at the start of this chapter, we present our findings in Figure 5.10. These plots clearly show that the Update LMB computation is the major bottleneck within the sequential SMC-LMB implementation. For KITTI-17, we get an average FPS of 20fps while for the worst-case (WC) frame we get up to 5fps. For a more tough video PETS09-S2L1, we rather get an average FPS of 15fps over all frames while the WC frame corresponds to about 2fps.

5.3.2. OpenCL Acceleration

As explained in chapter 4, we target the Update LMB computation using OpenCL compute kernels. Using this acceleration, we manage to gain substantial improvements in terms of execution performance as shown in Figure 5.11 where for PETS09-S2L1 video, we now get an average FPS of 100fps while WC frame amounts to 36fps. Figure 5.12 further compares the OpenCL timings directly with the C++ ones to further show the substantial gains in performance for the two algorithmic functions that have been accelerated up till now.

Finally, we also carry out the profiling of the LMB tracker algorithm under its different configurations as given by Table 5.1 with the purpose of studying how the execution performance timings scale with the greater number of particles or LMB hypotheses so as to increase tracking performance. These results are presented by in Figure 5.13. As shown in these plots, OpenCL implementation not only offers advantages in terms of sheer execution timings but also provides a much more scalable implementation as compared to its pure C++ counterpart. The execution times rises nearly exponentially for both MOT videos in case of C++ version while the rise is much less steeper or rather linear for the case of OpenCL based implementation.
5.3. Execution Performance Analysis

(a) KITTI17 - Average over all frames

(b) PETS09 - Average over all frames

(c) KITTI17 - WC frame

(d) PETS09 - WC frame

Figure 5.10.: C++ LMB Tracker execution performance
5. Evaluation and Analysis

**Figure 5.11.:** OpenCL accelerated LMB Tracker execution performance
5.3. Execution Performance Analysis

Figure 5.12.: Comparison between C++ and OpenCL LMB implementations

**(a)** Update LMB computation

**(b)** Update LMB intermediate computation
5. Evaluation and Analysis

![Bar chart showing scalability of C++ and OpenCL LMB implementations.](image)

**Figure 5.13:** Scalability of C++ and OpenCL LMB implementations
6. Conclusion

This chapter acts as the overall conclusion of this thesis document. Here, we first mention the main outlook or future work directions that can be taken to further extend this thesis work. Then we present a brief summary of the overall work being carried out as part of this project.

6.1. Summary

As part of this thesis work, we managed to come up with a fully functional pedestrian tracking system in the context of automotive driving and related safety concerns. This work further enhances the research work being carried out the TUM Robotics and Embedded Systems chair, as part of the TU9 research project, in coming up with modern heterogeneous ECUs as per the needs of advanced ADAS systems. These enhancements add on the already implemented lane-tracking algorithms with pedestrian detection and tracking algorithms, thus resulting in a considerably more compute-expensive functionality to run on the developed embedded platform to further study its effectiveness.

Starting from a vast literature review, we first chose the RFS based Bayesian filtering approach to be most suitable for our pedestrian tracking system. Specifically, we explored the use of GM-PHD and SMC-LMB class of filters from this family, to implement the tracking algorithm. Initially, pure SW (relying solely on CPU code in C++) implementations were carried out following a highly modular overall system-design approach. Later, via extensive evaluation analysis concerning tracking accuracy, the GM-PHD trackers were found to be inadequate and hence the implemented focus was then shifted solely to SMC-LMB filters. To run the whole tracking application meeting real-time constraints, we carried out an extensive execution profiling of the algorithm. The spotted performance bottlenecks were then ported to GPU accelerator using OpenCL programming constructs exploiting the parallelization potential of the algorithm. With this acceleration, we managed to come up with a working pedestrian tracking system that provides pedestrian tracks in real-time with decent accuracy. Specifically, we managed to improve from 15fps to 100fps in processing MOT video frames on average, while for the most computationally expensive frame we achieved 18x speedup improvements from 2fps to 36fps.

6.2. Future Work

Though we have successfully managed to come up with a working pedestrian tracking system satisfying real-time constraints covering most of automotive scenarios. But there is still much room for improvement in both the tracking accuracy as well as the execution performance domains. Here we present briefly some of the ideas that can be used to further enhance the capabilities of the designed system:
6. Conclusion

• As part of this thesis work, we have only managed to parallelize the main bottlenecks that were observed in the evaluation analysis of the tracker. This is sufficient enough for the tracker algorithm to meet the real-time constraints with the current HW setup. For less powerful machines, it makes sense to go for further parallelization of other functional blocks within the algorithm. Particularly, if one manages to parallelize the whole Update block within Figure 4.3 then this could help to do parallelized δ-GLMB updates via clustering techniques as presented in [25].

• Employ non-linear/non-Gaussian state-space models within the tracker algorithm. This would enable better tracking accuracy in application to wider range of pedestrian tracking scenarios.

• Currently the SMC-LMB tracker algorithm is accelerated using OpenCL 2.0 programming standard. This, on one hand, provides ease in parallel programming application development along with optimized code but at the same time loses out on the portability advantage. Most of GPU/FPGA hardware vendors currently support upto OpenCL 1.2 standard. Hence it makes sense to transform OpenCL 2.0 constructs to OpenCL 1.2 to support a wider range of hardware like NVidia GPUs or Altera FPGAs.

• With OpenCL 1.2 application developed, an interesting research direction could be to carry out performance comparison analysis between GPU and FPGA technologies in running the same tracker algorithm. This could lead to thoroughly evaluate the portability advantage of OpenCL over other major GP-GPU framework named CUDA.

• So far throughout thesis, we have mainly relied on either simulation analysis or MOT videos for evaluating the efficiency of the pedestrian tracking system. In both cases, the detector implementation was bypassed and all efforts concentrated around the tracker implementation. But in future, for working with real-world video footages which might come from cameras actually mounted on an automobile, an efficient detector design and implementation needs to be carried out along with its integration within the overall system.

• Generally the pedestrian detection process is considerably more expensive in terms of computational complexity than the pedestrian tracking process; as detection has to work with all the pixel data or sensor information while the tracker has to deal with mere detections only irrespective of the image data. Employing a detector of our own in the overall system opens up new design possibilities then to meet the real-time constraints. One possible idea that can be explored is to overlap detections with trackings instead of the tracking-by-detection approach we are currently using. In the overlap approach, the tracker uses its output tracks in the previous scan as if they were the new detections during the time the detector is busy processing initial frame. Once the detector finishes its task, we can feed in the new found detections into the tracker. The tracker can then use this information to rectify if its tracks are deviating from actual pedestrian tracks.

• An interesting idea could be to use both the GM-PHD tracker and the SMC-LMB tracker, developed as part of this thesis work, to work in collaboration to provide tracking for both pedestrians as well as other vehicles within the vicinity of the automobile unit housing this compute platform. Generally, the vehicle motion on the roads, more or less follow
6.2. Future Work

linear/Gaussian motion characteristics hence can be tracked with sufficient accuracy via GM-PHD tracker. The GM-PHD tracker because of its efficient computational complexity can easily be run on conventional embedded CPUs under real-time constraints. So a mix of GM-PHD tracker (possibly on host CPU) and SMC-LMB tracker (mostly on GPU) based application can be developed to further enhance the capabilities of the implemented system to make full use of the heterogeneity offered by the underlying embedded platform.
A. Fundamentals

A.1. Bayes Recursive Filter for Object Tracking

In object tracking, the complete probabilistic knowledge of the object/target states can be described by the joint pdf \( p(X^k) = p(X_k, X_{k-1}, X_{k-2}, \ldots, X_0) \), where \( X_k \) stands for a generic object state representing possibly single-target states, multi-target states, number of objects or the identity of objects etc. at time \( t_k \). If we denote mathematically \( z^k = (z_1, z_2, \ldots, z_k) \) to be all the sensor outputs up till this time instant where \( z_t \) represents the sensor measurement specifically at \( t_t \), then we can apply Bayes’ rule to update the current target state \( X \) using the latest information from the sensor as given by:

\[
p(X^k|z^k) = \frac{p(z^k|X^k)p(X^k)}{p(z^k)} \quad (A.1)
\]

So in essence, the application of Bayes’ theorem in the context of object tracking leads to a process where the updated knowledge of the states of the objects (aka the posterior density \( p(X^k|z^k) \)) can be obtained by multiplying out the prior joint density with the likelihood function \( p(z^k|X^k) \) and re-normalizing it with the normalization factor \( p(z^k) \). In theory, this provides the solution for the tracking problem but for most of the practical scenarios, the measurements are received in a sequential manner (over so called sensor-scans). The posterior density can be updated at each such scan to ensure that all the sensor information is used as soon as it is received to come up with the most accurate predictions and estimations. This leads to transformation of above solution into a recursive solution where the posterior \( p(X^k|z^k) \) is computed at each \( t_k \) after receiving the latest measurement \( z_k \).

To come up with the recursive solution, we breakdown \( z^k \) as \( z^k = (z_k, z^{k-1}) \). This allows:

\[
p(z^k|X^k) = p(z_k, z^{k-1}|X^k) = p(z_k|z^{k-1}, X^k)p(z^{k-1}|X^k) \quad (A.2)
\]

\[
p(X^k) = p(X_k, X^{k-1}) = p(X_k|X^{k-1})p(X^{k-1}) \quad (A.3)
\]

\[
p(z^k) = p(z_k, z^{k-1}) = p(z_k|z^{k-1})p(z^{k-1}) \quad (A.4)
\]

From the causality principle, it is clear that \( p(z^{k-1}|X^k) = p(z^{k-1}|X^{k-1}) \) as \( z^{k-1} \) is independent of future \( X_k \). Substituting these modified terms into the above Bayes rule:

\[
p(X^k|z^k) = \frac{p(z_k|z^{k-1}, X^k)p(z^{k-1}|X^k)p(X_k|X^{k-1})p(X^{k-1})}{p(z_k|z^{k-1})p(z^{k-1})} \quad (A.5)
\]

\[
= \frac{p(z_k|z^{k-1}, X^k)p(X_k|X^{k-1})}{p(z_k|z^{k-1})} \left( \frac{p(z^{k-1}|X^k)p(X^{k-1})}{p(z^{k-1})} \right) \quad (A.6)
\]

\[
= \frac{p(z_k|z^{k-1}, X^k)p(X_k|X^{k-1})}{p(z_k|z^{k-1})}p(X^{k-1}|z^{k-1}) \quad (A.7)
\]

For most of the object tracking problems of interest, it is reasonable to assume that the measurements at a given time \( t_k \) depend only on the object states at the corresponding time and
are conditionally independent of measurements taken at other times (*Markovian Systems*). This allows to formulate above as:

\[
p(X^k|z^k) = \frac{p(z_k|X_k)}{p(z_k|z^{k-1})} p(X_k|X_{k-1}) p(X^{k-1}|z^{k-1})
\]  
(A.8)

This provides the posterior distribution information taking into consideration all the target states uptill \( t_k \). Generally, in object tracking, one is interested in knowing the number of objects and their individual states exactly at \( t_k \). For this the posterior state conditional density \( p(X_k|z^k) \) holds significance which can be derived from the overall \( p(X^k|z^k) \) via its marginalization:

\[
p(X_k|z^k) = \int_{X_{k-1}} \ldots \int_{X_0} p(X^k|z^k) dX_0 \ldots dX_{k-1}
\]  
(A.9)

\[
= \frac{p(z_k|X_k)}{p(z_k|z^{k-1})} \int_{X_{k-1}} \ldots \int_{X_0} p(X_k|X_{k-1}) p(X^{k-1}|z^{k-1}) dX_0 \ldots dX_{k-1}
\]  
(A.10)

\[
= \frac{p(z_k|X_k)}{p(z_k|z^{k-1})} \int_{X_{k-1}} \int_{X_{k-2}} \ldots \int_{X_0} p(X_k|X_{k-1}) p(X^{k-1}|z^{k-1}) dX_0 \ldots dX_{k-1}
\]  
(A.11)

Since \( p(X^{k-1}|z^{k-1}) = p(X_{k-1}, X^{k-2}|z^{k-1}) \), the inner integrals in above simplify to \( p(X_{k-1}|z^{k-1}) \) hence leading to the recursive state-conditional density expression:

\[
p(X_k|z^k) = \frac{p(z_k|X_k) \int_{X_{k-1}} p(X_k|X_{k-1}) p(X_{k-1}|z^{k-1}) dX_{k-1}}{p(z_k|z^{k-1})}
\]  
(A.12)
Bibliography


