

# ADUULM-TTB: A Scalable, Generic, and Efficient Multi-Sensor Multi-Object Tracking Toolbox

Alexander Scheible, Charlotte Hermann, Thomas Griebel,  
Michael Buchholz, and Klaus Dietmayer

This paper has been accepted for presentation and publication at the 2025 IEEE INTERNATIONAL CONFERENCE ON MULTISENSOR FUSION AND INTEGRATION FOR INTELLIGENT SYSTEMS (MFI), 02-04 September 2025, College Station, USA.

This is the accepted version of the paper, which has not been fully edited and the layout may differ from the original publication.

## **Citation information of the original publication:**

A. Scheible, C. Hermann, T. Griebel, M. Buchholz and K. Dietmayer, "ADUULM-TTB: A Scalable, Generic, and Efficient Multi-Sensor Multi-Object Tracking Toolbox," 2025 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), College Station, TX, USA, 2025, pp. 1-8, doi: 10.1109/MFI67357.2025.11259402.

# ADUULM-TTB: A Scalable, Generic, and Efficient Multi-Sensor Multi-Object Tracking Toolbox

Alexander Scheible, Charlotte Hermann , Thomas Griebel , Michael Buchholz , and Klaus Dietmayer 

*Institute of Measurement, Control and Microtechnology*

*Ulm University, Germany*

{firstname.lastname}@uni-ulm.de

**Abstract**—Tracking multiple objects in dynamic environments is a crucial task in various applications, including automated driving, military surveillance, and robotics. While significant efforts have been made to develop open-source libraries, such as the Stone-Soup framework, these primarily focus on offline evaluation and benchmarking. However, there remains a gap in the availability of a high-performance, scalable, and generic tracking solution suitable for real-world deployment. In this work, we present a multi-sensor multi-object tracking toolbox, referred to as ADUULM-TTB, designed for real-world applications. Our framework combines scalability and generality with an efficient, high-performance implementation, making it suitable for real-time usage. To enhance usability, we integrate a direct interface with the Stone-Soup framework, allowing for seamless evaluation and integration. The ADUULM-TTB is open-source and available at [https://github.com/uulm-mrm/aduulm\\_ttb](https://github.com/uulm-mrm/aduulm_ttb).

## I. INTRODUCTION

Multi-sensor multi-object tracking is a key component in various applications, including automated driving, military surveillance, and robotics. In autonomous driving, for instance, accurately perceiving and tracking dynamic objects in the environment is essential for safe motion planning and decision-making. However, when multiple sensors and numerous objects are involved, the tracking process becomes highly complex and computationally demanding, especially for real-time applications [1], [2].

The research in tracking algorithms has been actively progressing, with many recent efforts focused on developing novel, theoretically complex methods. However, real-time applicability often does not receive the same attention. Additionally, unlike in the deep-learning field, it remains uncommon for researchers in the tracking community to publish the code for their novel algorithms, which hinders reproducibility and limits the ability to compare different approaches. To address this, the open-source Stone Soup framework [3], [4] has emerged as a valuable resource within the target tracking and state estimation community, providing a platform for easy-to-deploy

Parts of this research have been conducted as part of the PoDIUM project and other parts as part of the EVENTS project, which both are funded by the European Union under grant agreement No. 101069547 and No. 101069614, respectively. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them. Parts of this work have been financially supported by the Federal Ministry of Education and Research (project AUTOftech.agil, FKZ 01IS22088W).

We like to thank all contributors.

new algorithms and the evaluation and comparison of different trackers. However, Stone Soup is designed primarily for offline evaluations and lacks support for real-time online applications.

In this work, we address the research gap for an efficient and real-time tracking toolbox that is both scalable and generic, suitable for real-world applications as well as offline evaluations and comparisons of different tracking approaches. We call our proposed framework ADUULM-TTB. The multi-sensor multi-object tracking toolbox presented here is also the framework used for real-time autonomous driving on public roads in Ulm [5], [6], demonstrating its efficiency and real-world applicability over the past few years. A special focus of the toolbox is on implementing Random Finite Set (RFS)-based filters [7], particularly the Labeled Multi-Bernoulli (LMB) filter [8], which is provided in multiple variants. The toolbox is designed to be scalable, which means it can handle various sensor setups, support different tracker configurations, and easily integrate new tracking algorithms. Additionally, it is generic in that it can automatically manage different object state densities and dynamically process incoming measurements of varying types, i.e., heterogeneous measurement spaces. The implementation is high-performance, based on C++ and parallelization through a thread pool for efficient execution.

The main contributions of this paper are:

- an open-source toolbox for multi-sensor multi-object tracking called ADUULM-TTB,
- efficient implementation in C++ for real-world applications such as automated driving,
- real-time capable RFS-based implementations,
- an interface to Python and the Stone Soup tracking framework,
- a scalable and generic implementation for easy-to-configure your own setup and easy-to-implement new tracking algorithms.

## II. RELATED WORK

In the field of tracking and state estimation, several open-source frameworks exist. However, unlike deep learning, it is not yet widely practiced for researchers to openly publish their implementations.

One of the most well-known open-source frameworks developed in recent years is Stone Soup [3], [4], [9], designed for tracking and state estimation, primarily in Python. Stone Soup features a modular architecture, allowing users to seamlessly

combine components as needed. The framework is particularly suited for algorithm developers, enabling them to integrate and test new methods with ease. To ensure rigorous evaluation, Stone Soup includes well-established performance metrics such as the Optimal Sub-pattern Assignment (OSPA) metric [10] and the Generalized OSPA (GOSPA) metric [11]. Recognized as a valuable resource in the target tracking and state estimation community, Stone Soup provides a standardized platform for developing, evaluating, and comparing different tracking algorithms. Its design facilitates easy deployment of new methods, making it a useful tool for research and development. However, the framework is primarily intended for offline evaluations and does not natively support real-time online applications.

Another open-source framework, specifically designed for and known in the field of RFS theory [7], is the RFS tracking toolbox [12] based on MATLAB. This toolbox implements a variety of RFS-based filters, with an emphasis on algorithm and filter development rather than real-time performance. Consequently, the choice of MATLAB as the programming language is suitable, as it allows for easy prototyping and testing of new methods. The toolbox also includes implementations of several metrics, such as the OSPA metric [10] and the OSPA<sup>(2)</sup> metric [13], as well as an implementation of Yen’s  $k$  shortest path algorithm [14].

In addition, MATLAB itself provides built-in support for multi-object tracking algorithms, such as the Sensor Fusion and Tracking Toolbox, including trackers like the Global Nearest Neighbor (GNN) tracker [15] and many more, and also supports metrics like OSPA and GOSPA. A key limitation of MATLAB is that it requires a commercial license, which can be a barrier for some users.

In the field of automated driving, Autoware [16] is a widely recognized open-source framework, designed with a strong emphasis on real-time usability. Implemented in C++ and built on the Robot Operating System (ROS) [17], Autoware leverages ROS as its middleware, facilitating communication between various components such as tracking and planning. For tracking moving objects, Autoware utilizes Kalman filters [18] and particle filters [19], depending on the specific scenario at hand [16]. The framework prioritizes real-time performance for automated driving, which leads to a simplified tracking approach to minimize computational overhead. However, RFS-based trackers, which have shown success in real-time automated driving applications [5], [6], are not implemented within Autoware.

In summary, while there are several open-source frameworks available for tracking and state estimation, each has its limitations. These include issues such as the lack of online applicability and computational complexity, the absence of specific filter implementations like RFS-based filters, or reliance on commercial software as MATLAB. In response to these challenges, we present a new open-source tracking framework that addresses these gaps. Our ADUULM-TTB is efficiently implemented in C++ for real-time applicability, with an interface to Python and the Stone Soup framework. It

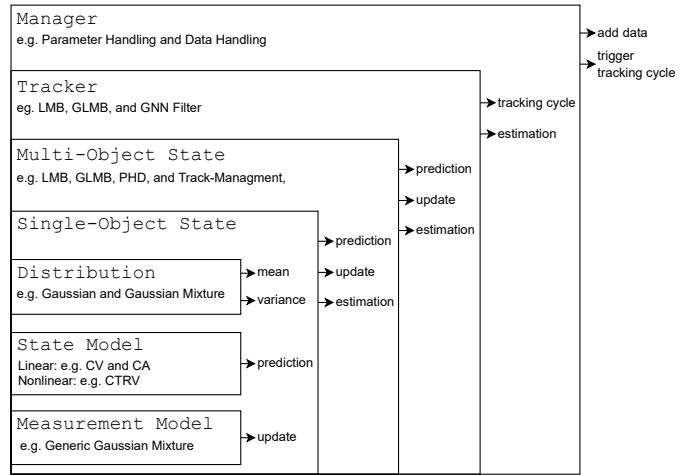


Fig. 1: Conceptual overview of the ADUULM-TTB. The core elements of the framework are typed in monospace, and the arrows show the key methods they provide.

includes a variety of RFS-based filter implementations while maintaining a modular and flexible design that allows for easy development of new methods.

### III. FRAMEWORK OVERVIEW

In this section, we introduce and describe our ADUULM-TTB. Figure 1 provides a conceptual overview, illustrating the connection from the `Single-Object State`, which represents the state of one object, to the `Manager`, which serves as the user interface and oversees all components of the tracking framework.

In the following, we introduce the tracking framework, starting from the foundational level of the `Single-Object State` and progressing upwards to the higher-level `Manager` responsible for coordinating the entire tracking process.

#### A. *Single-Object State*

A `Single-Object State` describes the state of a single object. All information about an object is collected here. These include kinematic information encoded by a stochastic `Distribution`, but also non-kinematic information like classification, label, existence probability, and track score as well as metadata like the time of first appearance, number of updates, and the duration since the last association. Note that in Fig. 1, only the kinematic part is shown. The `Single-Object State` manages the instances of the assumed and used `Distribution`, `State Model`, and `Measurement Model`. These components are closely interconnected, meaning that changes in one often require adjustments in the others. For example, as `Distribution` instance, replacing a Gaussian distribution, which is characterized by its mean and covariance, with a non-parametric particle-based representation necessitates modifications in both the prediction and update of the `State Model` and `Measurement Model`.

1) *Distribution*: The kinematic information of a Single-Object State is encoded by a specific stochastic Distribution. Note that a Distribution on itself has no connection to physical quantities and only provides methods like mean or variance. Currently, Gaussian distributions and Gaussian mixture distributions are fully supported in the framework.

2) *State Model*: To connect the dimensions of a Distribution to physical quantities like the x-position or angular velocity around the z-axis, a State Model is needed. The main functional task of a State Model is to provide the prediction method that allows the prediction of the kinematic information of a state, represented by a Distribution, to a certain time. Linear State Models like nearly constant position (CP), constant velocity (CV), or constant acceleration (CA), and non-linear State Models like constant turn-rate and velocity (CTRV) and constant turn-rate and acceleration (CTRA) are implemented [20], [21]. Independent extent information can extend all models, e.g., by length, width, and height or diameter.

3) *Measurement Model*: To update the kinematic information of a Single-Object State with a Measurement, the Measurement Model is used. Currently, a Generic Gaussian Mixture Measurement Model is implemented. It can handle heterogeneous measurements with different components in a generic way. Based on the components of the State Model and the components of the Measurement, either a linear Kalman update or a nonlinear update based on an unscented transformation [22], [23] as in the unscented Kalman filter is performed. Additionally, the handling of reference points [24] is also supported. Naturally, this only applies when the State Model includes the extent of an object.

4) *Key Abstraction*: As a general principle for a generic and scalable implementation, methods should be implemented at the lowest possible level. This means that the Single-Object State should be the only level that needs to know details about the Distribution, the State Model, and the Measurement Model used. Higher-level components like a Multi-Object State or even a Tracker should never depend on these lower-level details, e.g., whether a particle or Gaussian mixture distribution is used. Instead, they should only rely on the prediction and update provided by this level. Often, information that is computed during the prediction or update, e.g., the measurement likelihood, is also needed at higher levels. These values are saved within the Single-Object State for later usage.

## B. Multi-Object State

A Multi-Object State represents a collection of Single-Object States and includes methods for prediction, update, and estimation. However, unlike in a Single-Object States, these methods operate on the entire collection as a whole.

In this context, the prediction step not only models the evolution of existing objects but also accounts for object birth (appearance of new Single-Object States) and

death (disappearance of old objects). The update step processes a Measurement Set, which consists of multiple Measurements from the same data source at a given time, to update the Multi-Object State. Finally, the estimation method extracts and returns individual Single-Object States.

1) *Multi-Object Distributions*: There are multiple possible ways to represent a Multi-Object State. Historically, the framework was developed to realize RFS-based trackers [7], which are based on set-valued random variables. Within this category, we have implemented the LMBdistribution [25] and the Generalized LMB (GLMB) distribution [7]. Additionally, the Probability Hypothesis Density (PHD) is also implemented.

2) *Track Management*: Another approach is to manage the Multi-Object State as a collection of Single-Object States, using history-based or score-based management logic [26] to handle object birth, object death, as well as the estimation of the Multi-Object State. Our framework includes an implementation of history-based track management [26], which is primarily used by the GNN tracker.

## C. Tracker

A Tracker can perform a tracking cycle and provide an estimation. The tracking cycle uses a Multi-Sensor Measurement Set as input. There are multiple ways to deal with the multi-sensor aspect in the update step. One solution is the Iterator-Corrector (IC) scheme, in which the posterior of one sensor update is used as the prior for the next sensor update. Another solution is based on the Bayes Parallel Combination Rule. Here, the sensor updates use a common prior, and the individual updates are fused afterwards [27], [28]. Our framework includes implementations of both multi-sensor update strategies, providing flexibility in sensor fusion techniques.

## D. Manager

The Manager serves as the top-level entity and the primary interface for users of the ADUULM-TTB. It provides methods for adding sensor data to the framework and for initiating a tracking cycle.

1) *Data Management*: The main functional task of the Manager is the data management of the measurements from all sensors. A major challenge for this task is that data from different sensors can arrive out-of-sequence, meaning not in the order of acquisition. This is a problem for most tracking approaches as they do not solve this problem internally. Possible solutions in this case are data buffering and reprocessing [29].

The solution implemented in the framework is based on an adaptive minimal latency buffer, which delays the data as minimally as possible while still ensuring the in-sequence ordering [30]. Now, if a cycle is triggered, the buffer decides which data can safely be used. The data management can also be bypassed. This can be useful in situations where the data always arrives in sequence, e.g., in simulations, or when out-of-sequence data can be handled explicitly in lower levels.

2) *Parameter Management*: Another task of the Manager is the parameter handling. It stores all parameters of all levels, and the lower-level abstractions are granted only read access. This enables the possibility of the dynamical configuration of the whole framework.

#### E. Implementation

The ADUULM-TTB is implemented with C++. It uses Eigen3 [31] for handling vectors, matrices, and, more generally, linear algebra. Parallelization is supported through the thread pool [32]. Parameterization is supported through YAML and with the help of the great library figcone [33]. Profiling is supported through the frame profiler tracy [34]. The GUI is implemented with imgui. Different parts of the code use boost. Python bindings are implemented for some of the most important data structures and methods. For this, the library nanobind [35] is used.

### IV. HIGHLIGHTED FEATURES

In this section, we present selected features of our proposed tracking framework, highlighting its key strengths and innovations.

#### A. Interface to Stone Soup

Our ADUULM-TTB can be used through Python bindings. To enhance usability, we have developed a bridge from Stone Soup to our tracking framework. This bridge enables users to define and simulate a scenario in Stone Soup, perform the tracking task using the ADUULM-TTB, and then analyze the results back in Stone Soup. By leveraging this integration, users can take advantage of Stone Soup's powerful scripting and data visualization capabilities in Python while benefiting from the high-performance tracking computations in C++. The structure of this bridge and its connection to the Stone Soup framework is illustrated in Fig. 2.

#### B. Time Profiling with Tracy

Another important aspect of developing runtime-efficient code is time profiling during development. To analyze the runtime performance of the framework and its individual components, we use the Tracy frame profiler [34]. Tracy measures the runtime of annotated methods with nanosecond precision and includes a visualization tool for displaying this information. It is easy to use and provides valuable insights, particularly during algorithm development, where identifying performance-critical areas is essential.

#### C. GUI for Visualization and Configuration

The ADUULM-TTB features a GUI for visualizing key aspects of the tracking process. This includes current estimations, tracks, detections, and debug information, such as the delay of different data sources. Additionally, the GUI allows users to view and adjust certain parameters, providing greater flexibility and control over the tracking process.

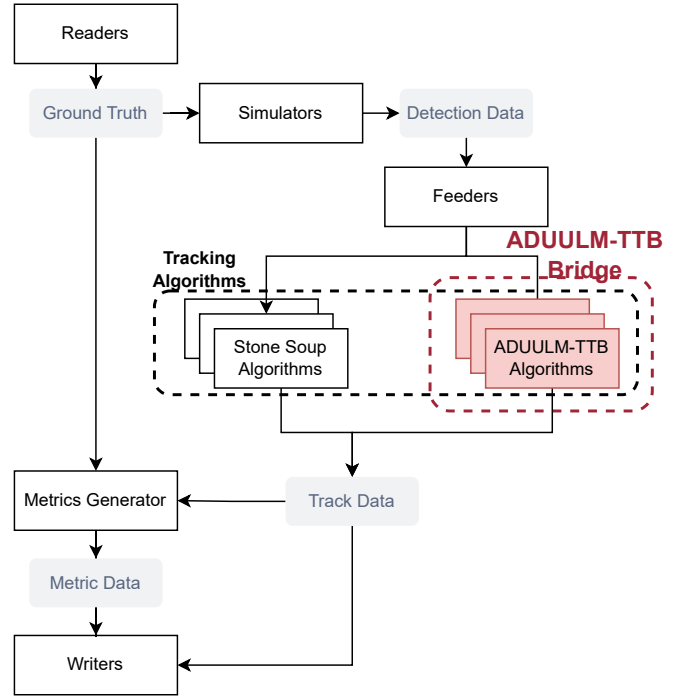


Fig. 2: The conceptual integration of the ADUULM-TTB in the Stone Soup framework. The bridge translates from Stone Soup data types into our data types and back.

#### D. Dynamic Reconfiguration

Since all parameters are stored in a single location, the framework's configuration can be easily modified at runtime. This includes parameters such as the process noise of a State Model, as well as settings related to post-processing or the update characteristics of a Multi-Object State. This flexibility allows the framework to dynamically adapt to external requirements, e.g., maintaining a constant runtime per cycle by adjusting post-processing parameters or even switching the filtering algorithm as needed.

#### E. Generic Measurement Handling

Another key feature of our framework is its generic measurement handling. Since each Measurement is fully self-contained, the framework can seamlessly process heterogeneous Measurements, even from the same sensor at the same time. In this context, 'heterogeneous' refers to Measurements that capture different physical properties. Our generic Gaussian measurement model dynamically chooses the right transformation from the state space to the measurement space. If this transformation is not possible, e.g., when the measurement space contains more information than the state space (e.g., a Measurement includes object height, but the state does not), our generic measurement handling simply ignores the unsupported components for that object.

#### F. $k$ Shortest Path Problem

Many sub-problems in RFS-based tracking can be formulated as  $k$ -shortest path problem. This includes, e.g., the prediction

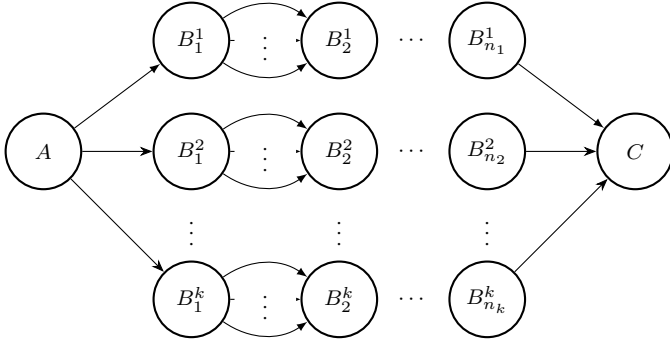


Fig. 3: Our  $k$ -shortest path implementation can solve the  $k$ -shortest problem for the weighted graph with this special structure. The algorithm finds the  $k$ -shortest paths from node  $A$  to node  $C$ .

of a GLMB distribution [25], the generation of new GLMB hypotheses for new objects, and the transformation from an LMB to GLMB distribution [36]. The ADUULM-TTB includes a  $k$ -shortest path solver, the *GeneralizedKBestSelection* algorithm, for graphs of the specific structure introduced in [36] and shown in Fig. 3.

## V. TRACKER IMPLEMENTATIONS AND SELECTED APPLICATIONS

In this section, we provide an overview of the trackers implemented within the ADUULM-TTB, along with selected applications to demonstrate the wide range of work that is possible with this framework. Our framework integrates various concepts from recent research, as proposed in different publications. Instead of implementing each concept in a separate tracker, they are typically integrated at the lowest possible level (cf. Fig. 1). Many of these concepts are embedded within the Multi-Object State level or even below, ensuring efficient and seamless integration into the overall tracking framework.

### A. Implemented Trackers

Various trackers have been implemented within the framework. These include RFS based trackers [7], such as the GLMB filter [25] and the LMB filter [8]. The joint prediction and update implementations [37]–[39] are also available in our framework for the LMB and GLMB filters. In addition, a multi-instances Kalman filter with a Global Nearest Neighbor (GNN) association [15] is implemented, referred to as the GNN tracker in the following. For all trackers, there is an IC implementation [27], [28] for multi-sensor cases. While this method may result in sensor-order dependence, the FPM-LMB filter [27], [36] addresses this issue by calculating single-sensor updates in parallel and then fusing the results. As the framework is designed to be easily extendable due to its generic nature, additional trackers will be incorporated in the future.

In general, the trackers leverage the algorithms implemented at the Multi-Object State level and below. This means the tracker inherits the parameters and various methods of all the underlying levels.

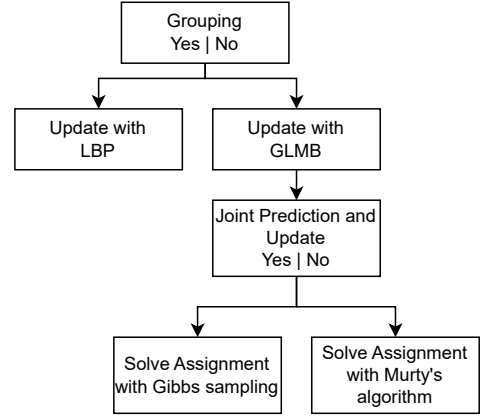


Fig. 4: The different methods to update an LMB distribution with a single Measurement Set. In total, ten different update methods are available.

As an example, the LMB filter, naturally, uses an LMB distribution to represent its Multi-Object State. An LMB distribution represents independent objects where each object has an existence probability and a unique label. See, e.g., [7] for more details. There are different methods to update an LMB distribution, e.g.,

- method A: by loopy belief propagation [39], or
- method B: by transforming it to a GLMB distribution and updating the GLMB [8], [25].

Additionally, grouping can be used or not. If method B is used, naturally, all methods for updating a GLMB distribution are available. This includes, e.g., the choice of whether joint-prediction and update should be used or the method for solving the measurement assignment problem [40]. For the assignment problem, we implemented two solutions: one based on Gibbs-sampling [40], and one based on Murty's algorithm [41]. As illustrated in Fig. 4, this results in ten different update methods for the LMB distribution, not counting different post-processing methods.

### B. Applications of the GeneralizedKBestSelection Algorithm

The *GeneralizedKBestSelection* algorithm [36] is used for different subtasks of the LMB and GLMB filter [25]. This includes the transformation from LMB to GLMB distribution as proposed in [36]. Here, only the  $k$  best GLMB hypotheses are calculated, which simplifies the computationally complex problem in the case of many tracks. Another application area is the truncation of the prediction of a GLMB distribution as proposed in [25]. Finally, the *GeneralizedKBestSelection* algorithm is applied to the fusion of spatial densities in the FPM-LMB filter [36]. Then, only the  $k$  best fused Gaussian mixture components per track are calculated. For more details, see [36].

### C. Self-Assessment with Parameter Estimation

An issue with model-based tracking lies in its heavy dependency on the correctness and accuracy of the used process and measurement models. If the models do not fit the current situation, the performance of the tracking result can rapidly

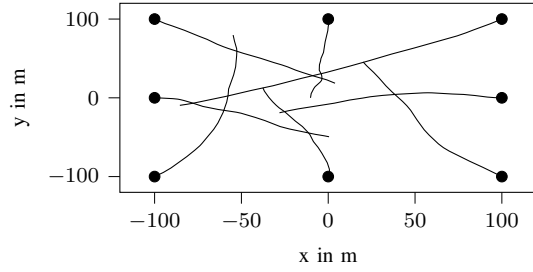


Fig. 5: Exemplary ground-truth paths of the first scenario. Note that only the initial states marked by the circles are fixed. The following states evolve according to a CV state model.

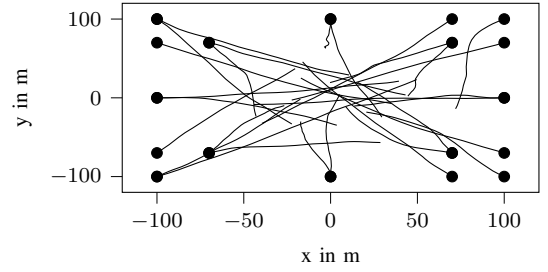


Fig. 6: Exemplary ground-truth paths of the second scenario. Note that only the initial states marked by the circles are fixed. The following states evolve according to a CV state model.

deteriorate. In the ADUULM-TTB, recent work [42], [43] is implemented, which proposes various approaches for estimating distinct parameters of the measurement model. With this estimation, one can compare and check the model parameters against the current situation and detect misconfigurations. Additionally, we plan to integrate and publish more recent research in the field of self-assessment, such as [44], [45], within the framework in the near future.

#### D. Track Classification

One example of non-kinematic information of a *Single-Object State* is the object class. The ADUULM-TTB implements the track classification methods described in [46]. These are based on class measurements, e.g., from a machine-learning detector, and provide additional information about a single object. This non-kinematic information is generally kept independent of the kinematic part but can also be used during prediction and update.

### VI. PERFORMANCE EVALUATION

To provide some insights and as proof of work, we evaluate some of the implemented trackers with Monte Carlo (MC) simulations. We use the introduced Stone Soup interface to generate scenarios and to evaluate the results. Additionally, the isolated runtime of the prediction and update of a single *Single-Object State* are evaluated separately.

#### A. Tracker Evaluation

In this section, we compare different trackers implemented within our ADUULM-TTB with respect to their tracking performance, measured by the generalized optimal sub-pattern assignment (GOSPA) metric [47], and their runtime per tracking cycle. For comparison, the Stone Soup multi-sensor multi-object tracker, as described in tutorial 10 [48], which is conceptually close to our GNN tracker, is also evaluated.

1) *Simulation Setup*: We use two different scenarios for evaluation. Figure 5 shows the ground truth paths of the first scenario. Note that only the initial states marked by the circles are fixed. The subsequent states evolve according to a CV state model with velocity noise diffusion coefficient  $q_x = q_y = 2$  [48]. We use a single sensor that measures the  $x$  and  $y$  position of an object with a measurement variance of  $v_x = v_y = 2 \text{ m}^2$  and a detection probability  $p_d = 0.9$ . In

addition, clutter measurements are uniformly distributed over the simulation area. The number of clutter is Poisson distributed with rate  $\lambda = 1$ .

The second scenario, shown in Fig. 6, is generally more difficult. The density of objects is higher, leading to a more difficult measurement association situation. Two sensors with identical parameters are used. They have a measurement variance of  $v_x = v_y = 2 \text{ m}^2$ , a clutter rate of  $\lambda = 25$ , and a detection probability of  $p_d = 0.7$ .

Both scenarios are evaluated with 50 MC runs. The runtime is calculated per tracking cycle and divided into sequential and parallel execution. For parallel execution, 20 threads are used.

2) *Results*: The results are summarized in Table I. For the simple scenario, all trackers show a similar tracking performance. The GLMB tracker, while theoretically optimal, is the exception and has the worst performance. Note, however, that with optimal pre- and post-processing parameters that are fine-tuned for that specific scenario, the GLMB filter shows the best result. We decided intentionally not to do that to show that the most complex and theoretically best trackers do not necessarily perform best for different scenarios.

In the second scenario, the differences between the trackers are more prominent. Generally, all our trackers show a better performance than the Stone Soup tracker. However, the Stone Soup tracker's poor performance could possibly be due to a bad configuration, although we followed the Stone Soup tutorials closely. With the exception of the GLMB tracker, the runtime of our trackers is mostly determined by the prediction and update of *Single-Object State*. This explains why a conceptually more complex tracker like the LMB filter can have a smaller runtime than the simpler GNN tracker because it has to maintain fewer internal, i.e., non-estimated, tracks.

#### B. Single-Object State Evaluation

For a more fine-grained runtime evaluation, we evaluate the prediction and update of a *Single-Object State* separately.

Table II shows the runtime for the prediction for linear (CV) and non-linear (CTRV and Constant Turn [48]) state models of a single Gaussian distribution. Note that the Constant Turn state model of Stone Soup is comparable to our CTRV state model. For the non-linear prediction, an unscented transformation is used. The results show that our ADUULM-TTB is typically

TABLE I: Runtime per tracking cycle and performance comparison of different trackers. All results are averaged over 50 MC runs.

Tracker	Scenario 1 (easy)			Scenario 2 (complex)		
	GOSPA	Runtime in ms		GOSPA	Runtime in ms	
		sequential	parallel		sequential	parallel
GNN (ADUULM-TTB)	7.20	0.61	0.29	22.79	46.39	10.71
GNN (Stone Soup)	7.97	2.95	-	44.05	58.32	-
LMB IC (ADUULM-TTB)	7.18	0.49	0.22	17.12	24.77	6.27
LMB FPM (ADUULM-TTB)	7.17	0.58	0.35	16.16	32.58	8.79
GLMB IC (ADUULM-TTB)	10.12	7.20	6.48	19.92	190.27	166.42

TABLE II: Runtime comparison of the Single-Object State prediction for different State Models.

State Model	Runtime in $\mu$ s	
	mean	standard derivation
CV (ADUULM-TTB)	6.9	0.3
CV (Stone Soup)	84.5	4.5
CV + Box (ADUULM-TTB)	13.2	0.5
CV + Box (Stone Soup)	116.6	4.7
CTRV (ADUULM-TTB)	15.5	0.7
CTRV + Box (ADUULM-TTB)	34.4	1.0
Constant Turn (Stone Soup)	799.3	841.7
Multi-Modell: CV + CTRV (ADUULM-TTB)	22.6	0.9

TABLE III: Runtime comparison of the Single-Object State update.

Situation	Runtime in $\mu$ s	
	mean	standard derivation
linear update of CV model (ADUULM-TTB)	8.1	0.7
linear update of CV model (Stone Soup)	58.3	2.5
linear update of CTRV + Box with box measurement (ADUULM-TTB)	12.9	2.2
non-linear update of CTRV + Box with camera box measurement (ADUULM-TTB)	26.7	7.9
nonlinear update of Constant Turn with bearing range measurement (Stone Soup)	447.5	22.3

one order faster for the linear state models. In the non-linear case, both implementations are slower because of the more complex unscented transformation. However, the runtime of our ADUULM-TTB is still comparable to the linear case, whereas the runtime of Stone Soup is nearly eight times that of the linear case. In a multi-model setup, the runtime is approximately the sum of that of the individual models.

Table III shows the runtime for the update of a single Gaussian distribution. In general, the runtime for the linear updates is comparable for different state models, whereas the ADUULM-TTB implementation is roughly 5 – 10 times faster than Stone Soup. As before, the non-linear update is done with an unscented transformation and is generally slower than the linear update. In our case, we update a CTRV state model with two bearing measurements from a camera box detection. For Stone Soup, we evaluate the comparable scenario of a Constant Turn state model updated with a bearing measurement. In our ADUULM-TTB, the non-linear update needs roughly two times

the runtime of the linear case, whereas the Stone-Soup is again roughly eight times slower compared to the linear case.

In summary, our ADUULM-TTB's runtime can be expected to be roughly one order of magnitude smaller than Stone Soup's. This supports our claim of an efficient implementation. Additionally, through the different State Models and the generic Gaussian measurement model support, different scenarios are supported. This shows the ADUULM-TTB's flexibility and scalability.

## VII. CONCLUSION AND FUTURE WORK

With this paper, we introduced our ADUULM-TTB for efficient, generic, and scalable multi-sensor multi-object tracking. The toolbox integrates seamlessly into the Stone Soup framework, accelerating the development process. Implementations of trackers with the ADUULM-TTB promise a speedup of roughly one order of magnitude compared to Stone Soup while remaining high-level, thanks to the clear abstractions defined by the framework. As future work, we want to extend the ADUULM-TTB with other algorithms and increase the usability by providing connectivity to other common frameworks like ROS2 or Autoware.

## REFERENCES

- [1] B.-N. Vo, B.-T. Vo, and M. Beard, "Multi-sensor multi-object tracking with the generalized labeled multi-bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 67, no. 23, pp. 5952–5967, 2019.
- [2] K. Da, T. Li, Y. Zhu, H. Fan, and Q. Fu, "Recent advances in multisensor multitarget tracking using random finite set," *Frontiers of Information Technology & Electronic Engineering*, vol. 22, no. 1, pp. 5–24, 2021.
- [3] P. A. Thomas, J. Barr, B. Balaji, and K. White, "An open source framework for tracking and state estimation ('Stone Soup')," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVI*, I. Kadar, Ed., vol. 10200, International Society for Optics and Photonics. SPIE, 2017, p. 1020008.
- [4] S. Hiscocks, J. Barr, N. Perree, J. Wright, H. Pritchett, O. Rosoman, M. Harris, R. Gorman, S. Pike, P. Carniglia, L. Vladimirov, and B. Oakes, "Stone Soup: No Longer Just an Appetiser," in *2023 26th International Conference on Information Fusion (FUSION)*, 2023, pp. 1–8.
- [5] F. Kunz, D. Nuss, J. Wiest, H. Deusch, S. Reuter, F. Gritschneider, A. Scheel, M. Stubler, M. Bach, P. Hatzelmann, C. Wild, and K. Dietmayer, "Autonomous driving at Ulm University: A modular, robust, and sensor-independent fusion approach," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2015-Augus, pp. 666–673, 2015.
- [6] M. Buchholz, J. Müller, M. Herrmann, J. Strohbeck, B. Völz, M. Maier, J. Paczia, O. Stein, H. Rehborn, and R.-W. Henn, "Handling occlusions in automated driving using a multiaccess edge computing server-based environment model from infrastructure sensors," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 3, pp. 106–120, 2022.
- [7] R. P. S. Mahler, *Statistical multisource-multitarget information fusion*. Boston: Artech House, 2007.



- [8] S. Reuter, B.-T. Vo, B.-N. Vo, and K. Dietmayer, "The Labeled Multi-Bernoulli Filter," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3246–3260.
- [9] E. P. Blasch, R. Niu, and S. O'Rourke, "Target Tracking Analysis for Stone Soup," in *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, 2020, pp. 1–8.
- [10] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A Consistent Metric for Performance Evaluation of Multi-Object Filters," *IEEE Transactions on Signal Processing*, vol. 56, no. 8, pp. 3447–3457, Aug. 2008.
- [11] A. S. Rahmathullah, A. F. Garcia-Fernandez, and L. Svensson, "Generalized optimal sub-pattern assignment metric," in *2017 20th International Conference on Information Fusion (Fusion)*, Xi'an, China, Jul. 2017, pp. 1–8.
- [12] B.-T. Vo, B.-N. Vo, D. Y. Kim, S. Reuter, and M. Beard, "Matlab random finite set tracking toolbox," 2013, <https://ba-tuong.vo-au.com/codes.html>, accessed: 2025-02-27.
- [13] M. Beard, B. T. Vo, and B.-N. Vo, "OSPA <sup>(2)</sup> : Using the OSPA metric to evaluate multi-target tracking performance," in *2017 International Conference on Control, Automation and Information Sciences (ICCAIS)*, Chiang Mai, Thailand, Oct. 2017, pp. 86–91.
- [14] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.
- [15] S. S. Blackman and R. Popoli, *Design and analysis of modern tracking systems*. Artech House, 1999.
- [16] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPs)*, 2018, pp. 287–296.
- [17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *Proc. of IEEE International Conference on Robotics and Automation Workshop on Open Source Software*, vol. 3, no. 3.2, 2009, pp. 1–6.
- [18] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering, Transactions of the ASME*, vol. 82, no. 1, pp. 35–45, 1960.
- [19] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.
- [20] X. Rong Li and V. Jilkov, "Survey of maneuvering target tracking. Part I. Dynamic models," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1333–1364, 2003.
- [21] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *2008 11th International Conference on Information Fusion*, 2008, pp. 1–6.
- [22] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, I. Kadar, Ed., vol. 3068, International Society for Optics and Photonics. SPIE, 1997, pp. 182 – 193.
- [23] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000, pp. 153–158.
- [24] K. Schueler, T. Weiherer, E. Bouzouraa, and U. Hofmann, "360 Degree multi sensor fusion for static and dynamic obstacles," in *2012 IEEE Intelligent Vehicles Symposium*, 2012, pp. 692–697.
- [25] B.-T. Vo and B.-N. Vo, "Labeled Random Finite Sets and Multi-Object Conjugate Priors," *IEEE Transactions on Signal Processing*, vol. 61, no. 13, pp. 3460–3475, Jul. 2013.
- [26] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*, ser. Artech House radar library. Artech House, 1999.
- [27] C. Hermann, M. Herrmann, T. Griebel, M. Buchholz, and K. Dietmayer, "The Fast Product Multi-Sensor Labeled Multi-Bernoulli Filter," in *2023 26th International Conference on Information Fusion (FUSION)*, 2023, pp. 1–8.
- [28] M. Herrmann, C. Hermann, and M. Buchholz, "Distributed implementation of the centralized generalized labeled multi-Bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 69, pp. 5159–5174, 2021.
- [29] H. Winner, *Handbuch Assistiertes und Automatisiertes Fahren*, 4th ed., ser. ATZ/MTZ-Fachbuch, H. Winner, K. C. J. Dietmayer, L. Eckstein, M. Jipp, M. Maurer, and C. Stiller, Eds. Wiesbaden, Germany: Springer Vieweg, Aug. 2024.
- [30] T. Wodtke, A. Scheible, D. Authaler, and M. Buchholz, "Adaptive Minimal Latency In-Sequence Ordering for Multi-Channel Data Fusion in Autonomous Driving," in *submitted to 2025 IEEE 36th Intelligent Vehicles Symposium (IV)*, 2025, pp. 1–6.
- [31] Gaël Guennebaud and Benoît Jacob and others, "Eigen v3," <http://eigen.tuxfamily.org>, 2010.
- [32] B. Shoshany, "A C++17 Thread Pool for High-Performance Scientific Computing," *SoftwareX*, vol. 26, p. 101687, 2024.
- [33] kamchatka volcano, "figcone," <https://github.com/kamchatka-volcano/figcone>, 2022.
- [34] B. Taudul, "tracy," <https://github.com/wolfpld/tracy>, 2022.
- [35] W. Jakob, "nanobind: tiny and efficient C++/Python bindings," 2022, <https://github.com/wjakob/nanobind>.
- [36] C. Hermann, A. Scheible, M. Buchholz, and K. Dietmayer, "An Efficient Implementation of the Fast Product Multi-Sensor Labeled Multi-Bernoulli Filter," in *2024 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2024, pp. 1–8.
- [37] B.-N. Vo, B.-T. Vo, and H. G. Hoang, "An Efficient Implementation of the Generalized Labeled Multi-Bernoulli Filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2017.
- [38] S. Reuter, A. Danzer, M. Stübler, A. Scheel, and K. Granström, "A fast implementation of the Labeled Multi-Bernoulli filter using gibbs sampling," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 765–772.
- [39] T. Kropffreiter, F. Meyer, and F. Hlawatsch, "A Fast Labeled Multi-Bernoulli Filter Using Belief Propagation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 3, pp. 2478–2488, 2020.
- [40] B.-N. Vo, B.-T. Vo, and H. G. Hoang, "An Efficient Implementation of the Generalized Labeled Multi-Bernoulli Filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2017.
- [41] M. Miller, H. Stone, and I. Cox, "Optimizing Murty's ranked assignment method," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 3, pp. 851–862, 1997.
- [42] A. Scheible, T. Griebel, and M. Buchholz, "Self-Monitored Clutter Rate Estimation for the Labeled Multi-Bernoulli Filter," in *2024 27th International Conference on Information Fusion (FUSION)*, 2024, pp. 1–7.
- [43] —, "Self-Monitored Detection Probability Estimation for the Labeled Multi-Bernoulli Filter," in *accepted at 27th IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2024, pp. 1–7.
- [44] T. Griebel, J. Müller, P. Geisler, C. Hermann, M. Herrmann, M. Buchholz, and K. Dietmayer, "Self-Assessment for Single-Object Tracking in Clutter Using Subjective Logic," in *2022 25th International Conference on Information Fusion (FUSION)*, 2022, pp. 1–8.
- [45] T. Griebel, N. Dehler, A. Scheible, M. Buchholz, and K. Dietmayer, "Self-assessment for multi-object tracking based on subjective logic," in *2024 IEEE Intelligent Vehicles Symposium (IV)*, 2024, pp. 1750–1757.
- [46] A. Scheible, T. Griebel, M. Herrmann, C. Hermann, and M. Buchholz, "Track Classification for Random Finite Set Based Multi-Sensor Multi-Object Tracking," in *2023 IEEE Symposium Sensor Data Fusion and International Conference on Multisensor Fusion and Integration (SDF-MFI)*, 2023, pp. 1–8.
- [47] A. S. Rahmathullah, A. F. Garcia-Fernandez, and L. Svensson, "Generalized optimal sub-pattern assignment metric," in *2017 20th International Conference on Information Fusion (Fusion)*, 2017, pp. 1–8.
- [48] D. Science, T. L. UK, D. Research, D. C. . R. et développement pour la défense Canada, U. of Liverpool UK, F. FKIE, J. Hiles, R. Ltd, R. M. R. L. UK, and L. U. UK, "Stone Soup," <https://stonesoup.readthedocs.io/en/v1.5/>, 2025.