sciendo

Transport and Telecommunication, 2023, volume 24, no. 1, 65–72
Transport and Telecommunication Institute, Lomonosova 1, Riga, LV-1019, Latvia
DOI 10.2478/ttj-2023-0007

FAST TIMELINE BASED MULTI OBJECT ONLINE TRACKING

Martin Hünermund[1], Maik Groneberg[2], Nils Brauckmann[3]

[1] Fraunhofer Institute for Factory Operation and Automation IFF
Sandtorstr. 22, 39106, Magdeburg, Germany
martin.huenermund@iff.fraunhofer.de
maik.groneberg@iff.fraunhofer.de
nils.brauckmann@iff.fraunhofer.de

Fast state-of-the-art multi-object-tracking (MOT) schemes, such as reported in challenges MOT16 and Mot20, perform tracking on a single sensor, often couple tracking and detection, support only one kind of object representation or don't take varying latencies and update rates into account.

We propose a fast generic MOT system for use in real world applications which is capable of tracking objects from different sensor / detector types with their respective latencies and update rates. An SORT inspired online tracking scheme is extended by time awareness using timelines as unifying concept. The system supports different object, sensor and filter and tracking types by modularizing and generalizing the online tracking scheme, while ensuring high performance using an efficient data-oriented C++-template-based implementation.

Using the proposed system we achieve, with comparable evaluation metrics, framerates up to ten times higher than the fastest MOT schemes publicly listed for the axis-aligned bounding-box tracking challenges MOT17 and MOT20.

Keywords: multi object tracking, sensor fusion, timeline

1. Introduction

We propose a fast generic MOT system for use in real world applications which is capable of tracking objects from different sensor and detector types with their respective communication latencies and update rates.

For the obstacle tracking of an autonomous e-bike, developed in the context of the research project RavE-Bike, we employ stationary geo-referenced cameras and mobile on-board lidar sensor and color cameras. The tracking of obstacles allows an autonomous control system to plan around the tracked stationary and moving obstacles. In the context of collision avoidance for mobile systems we want the tracking system to not miss any potential obstacles, if possible provide past and future obstacle trajectory information, handle detections from multiple sensors and be reasonable fast on mobile, possibly embedded, computers. An ideal system should be flexible, i.e. independent from actual problem setting, e.g. AABB tracking, it should be multi sensor capable and usable in a (soft) real time context with low computational overhead, in other words it should be fast.

Often new MOT schemes focus on detection, re-identification and association methods. Speed is mostly only of second priority resulting in suboptimal implementations. We generalize and modularize the already fast and mostly generic multi object tracking framework SORT (Bewley, 2016) and support multi sensor capability by correctly handling detection time points using timelines (Reinholtz, 2012) as unifying data structures. High runtime performance is gained due to a data oriented design and heavy use of C++ template metaprogramming. Additionally, we use an computational efficient exponentially weighted moving average filter based approximation for the filtering problem in our use-case. We evaluate our system on public multi object tracking benchmarks MOT17, MOT20 (Milan, 2016) (Dendorfer, 2020). The most important metrics for our application use case of collision avoidance are speed and low false-negative rates, i.e. don't miss objects. Noteworthy is that speed is no official metric in the MOT benchmarks, so we have to rely on the self-reported values from various authors.

2. Related Work

Most recent multi object tracking advances focus on improving overall metrics with various detection, re-identification or association methods, while others strive for real time runtime performance

and generally low computational costs. We evaluate our method in comparison to the currently fastest algorithms of the MOT17 and MOT20 benchmarks: IOU17 (Bochinski, 2017), MAA (Stadler, 2022), SORT17, SORT20 (Bewley, 2016), OCSort (Cao, 2022), CTv0 (Lohn-Jaramillo, 2022), IOU_KMM (Urbann, 2021), and PaddleDetection.

IOU17 is "High-speed tracking-by-detection without using image information" which focuses on simplification of the tracking with a greedy approach for association and matching, resulting in reduced computational costs while still being able to compete with more sophisticated approaches. Bochinski *et al.* (2017) report high frame rates up to 100 khz for certain data sets and has the best framerate of 1523 hz from the public MOT17 benchmark entries that have a listed paper. Stadler and Beyerer (2022) propose with MAA a new assignment algorithm for the association and matching problem that explicitely models ambiguous situations. Ambiguous matches are separately handled and the clear cases are matched using a variant of the Hungarian method. The public entry for the MOT17 benchmark is listed with 189 hz. SORT17 and SORT20 are implementations of "Simple Online Realtime Tracking". Bewley *et al.* (2016) propose a simple architecture for detection based tracking with IOU based association metric and rectangular linear sum assignment using the Jonker-Volgenant algorithm (Jonker, 1987), a variant of the Hungarian method. The public entries of this algorithm list 143 Hz for MOT17 and 57 Hz for MOT20. The architecture is easy to generalize and forms the foundation for various SORT inspired methods like DeepSORT (Wojke, 2017) or OCSort (Cao, 2022). OCSort is "Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking", where a observation centric filtering strategy is used. The public entry for the MOT20 benchmark reports 28 Hz for OCSort. IOU_KMM is "Online and Real-Time Tracking in a Surveillance Scenario" (Urbann, 2021), a SORT based method with an improved deep learning re-identification metric. The corresponding public benchmark entry lists 30 Hz for MOT20. Lohn-Jaramillo *et al.* (2022) propose CTv0 "ClusterTracker: An Efficiency-Focused Multiple Object Tracking Method, with 318 Hz for the MOT20 benchmark on of the fastest entries. They drastically reduce the usually cubic time complexity of matching by spatial clustering to query matching candidates of nearby tracks and detections. One high listed entry, named "Paddle Detection" has no associated paper, so we don't include it in the evaluation section. The implementation uses fast performing detectors and either FairMOT (Zhang, 2021), DeepSORT (Wojke, 2017), JDE (Wang, 2020) or Bytetrack (Zhang, 2022) for the tracking itself. Fair-MOT has a framerate of 25.9 Hz and 13.2 Hz for MOT17 and MOT20 benchmark. Wang (2020) report up to 30.3 hz for JDE on MOT16. They focus on detection and re-identification. Zhang (2022) reports association times of 4 ms for Bytetrack on MOT17 which corresponds to 250 hz. Wojke (2017) report 40 Hz for their method DeepSORT on MOT16. DeepSORT is a SORT based algorithm with deep learning based association metric.

Often we have tracking algorithms that focus on detection and re-identification methods. SORT is often used. Its architecture is mostly generic with easily swappable slots for association metric, matching and filter algorithm.

3. Timeline Based Multi Object Tracking

SORT (Bewley, 2016) is used as basis for a flexible multi object tracking update scheme. We explicitly modularize it into swappable components (boxes in fig. 1). Association will associate tracks with observations, usually by assigning a cost to each pair in a cost matrix. During matching the associations are used to find optimal matches and unmatched observations or tracks. We currently use, like SORT and many others, a variant of the Hungarian algorithm, the Jonker-Volgenant algorithm (Jonker, 1987) for matching. These matching algorithms have a cubic runtime dependent on the number of observations and tracks. Lohn-Jaramillo (2022) recently suggested spatial clustering of observations and tracks to reduce the number of matching candidates and drastically reduce the resulting runtime. A similar scheme is possible with the general architecture of fig. 1 by providing components for association and matching with a different data type than cost matrices as interface in-between. Matched observations are used to update track estimates with a filter. Often Kalman filtering is used for this, we provide a fast approximation using exponential weighted moving average (EMWA) filtering. A track lifecycle strategy determines when to discard tracks. Bookkeeping for this happens during track update, on unmatched tracks, and on track creation of unmatched observations. It is often based on the number of time steps since last matching observation or the track estimation quality which gets worse over time with no matching observations to support it.

Timelines are used as unifying concept (Reinholtz, 2012) to store and process data. They enable us to correctly handle timestamps of observations from various possibly distributed detection sources. The observations are integrated into a single timeline of observations Z_t. The update scheme (Figure 1) is then regularly used to update track estimates. In contrast to pure online tracking schemes, the history of

observations makes it possible to recompute estimates on arrival of older observations. If for example in Figure 2 the missing observations Z_2 arrives later due to communication latencies it can still be integrated into the track estimates by recomputing X_2 to X_n.

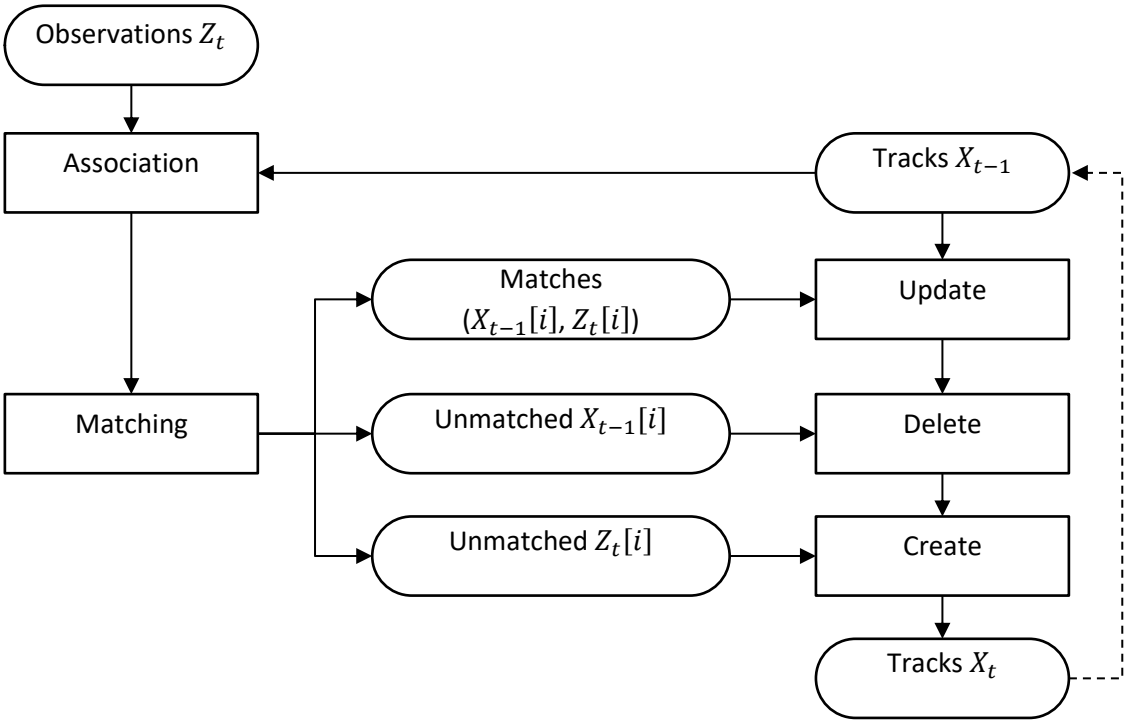


Figure 1. Online-Update Scheme based on SORT (Bewley, 2016)

Multi sensor capability is achieved due to the generalized architecture which allows different kinds of observations, detections, filters, etc. and the ability to correctly handle timestamped data from different sources. Due to the importance of speed a data oriented design and other performance considerations are used for the implementation.

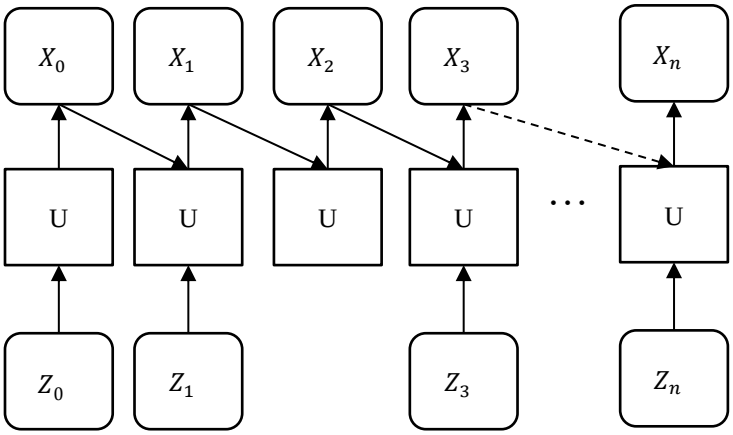


Figure 2. Timeline based updating U of tracks X_k with observations Z_k

3.1. Timeline based Tracking

Detection systems can consist of multiple sensors at different locations with multiple detectors. They produce time stamped detection events which are communicated to the multi object tracking system. Events can sometimes arrive out of order. Correct handling of event time points is required. To achieve this, we use ordered timelines (Reinholtz, 2012). These generally provide a unifying interface for storing and accessing timestamped data. Timeline based tracking needs timelines for detections, tracks corrections and

intermediate tracks predictions. Intermediate prediction events are generated based on configurable maximum prediction step and just advance tracks estimates based on the filter strategy.

New detection events are continuously inserted into the detection timeline. For each detection event a tracks correction event is inserted. In regular intervals of configurable duration, a tracking update is performed. Events are (re-)processed by iterating the timelines starting from the earliest new correction event. Association and matching is performed for each correction event, between each last tracks estimate and detections. New tracks are formed or estimates are corrected according to the chosen filter algorithm. Old tracks are removed depending on the track lifetime strategy.

As we are storing incoming events, ordered insert should be fast. The tracking update repeatedly requires ordered iteration over events, so this should also be fast.

3.2. Data Oriented Design

Data oriented storage ensures good performance. Cache locality shall be leveraged while processing timelines. Timeline events should not be scattered in memory. Instead they are stored in an ordered memory pool. The size of timeline is bounded by a sliding timespan window. The upper bound of items in a timeline is generally proportional to its timespan. Due to possibly varying event rates it is not exactly known a priori. Mostly the number of currently stored time events would not exceed a few ten thousand. For example, one minute of events with an average rate of one kilohertz corresponds to sixty thousand events. The number of items is thus practically bounded to a reasonable size. It would even be feasible to sort such small number of items each time. But we can do better.

We use data oriented linked list index sets for first-in first-out slot recycling memory pooling and for book-keeping of timeline event ordering. The linked list index sets itself are small and can remain in or close to the cache. The underlying dynamic storages for memory pool and linked lists are initialized with an expected maximum size to avoid unnecessary reallocations during operation. In the beginning all slots are unused. During this filling-phase new items are allocated from sequential slots. When all initially unused slots got allocated the filling-phase transitions into the recycle-phase. Due to the sliding window items are freed from their timeline and their slots then made available for new items in a first-in first-out manner. More storage is allocated in case of insufficient capacity, which results in retransitioning to the filling-phase. Because events may arrive from different sources, with different communication latencies, they may arrive out-of-order. Yet the overall order of event arrival is determined by the time point they occurred. Building upon all these points the order of event slots is kept roughly like the order in the timeline with only few and short jumps. During iteration of the ordered timeline the resulting memory access pattern contains only few and small jumps. The memory requirements for the linked lists are small, so they can remain in or close to the cache.

3.3. Linked List Index Set

Data oriented linked lists are used to store an ordered set of indices. Linked list items consists of for- and backward indices and a flag whether the item at its index is contained in the set or not. Items are stored in dynamic sequential storage with its size determined by the expected maximum index, e.g. the capacity of the memory item pool. Items in the linked list are stored at the same index as their corresponding event item in its memory pool.

Item insertion and removal is then just and changing a few indices and a flag. Since we don't need a sorted timeline after each insertion, but rather at each update, we defer the ordering of the insertion until later. All collected unordered insertions will be sorted and then inserted in order. Sorting the unordered insertions makes it easy to find the correct insert positions by merging with the already sorted timeline. Immediate ordered insertion would require access to the corresponding linked lists and finding the insert position each time an event arrives. By batching the ordered insertion, the total number of memory accesses is reduced which improves performance. The removal of items due to sliding time window is also naturally batched.

Usually linked lists are considered bad for performance as their implementation is often using dynamic allocation for each individual item, which results in a completely scattered memory access pattern. By keeping the memory close together, the performance is vastly improved.

3.4. C++ Template Based Implementation

Our approach for modularizing the multi object tracking system to allow user defined types makes heavy use of C++ template metaprogramming, inlining and avoidance of virtual function dispatch. This is

an important optimization technique to gain high performance. Templated functions can be resolved at compile time and consequently some function calls result in no overhead with good opportunities for compiler optimization (Museth, 2013). Concrete system configurations are implemented by specifying template instantiations and optionally implementing new component types.

3.5. Exponentially Weights Moving Average Filter

Often Kalman filters are used to filter track estimates of position, orientation and/or size. Kalman filters use matrix inversion and a lot of matrix multiplications which results in a relatively high computational burden. We explain under which assumptions and how we approximate them with computational very efficient exponential weighted moving average filters.

Kalman filters with constant process and measurement noise often converge into a steady-state where the estimate covariance approaches a limit which is called the steady-state covariance. This happens when the average increase of covariance during estimate prediction approaches the average decrease of covariance during estimate correction. It is independent of actual measurement values, but only depends on the process and measurement noise covariance, and the used time steps. Often, like in SORT (Bewley, 2016), these parameters are chosen a priori and held fix during operation. These Kalman filters then have a warm-up phase and a steady-state phase.

The warm-up phase usually begins with a high covariance, representing initially high uncertainty, which then approaches the state-state covariance. Measurements initially have very high influence on the estimate, which reduces down to a stable state. In the steady-state phase the covariance is nearly at its convergence limit and the influence of each measurement is nearly constant. Summarizing we have a phase where measurements have high influence and a phase where they have a certain constant influence.

The Kalman filter equation for correcting the state estimate depends on the Kalman gain which is constant in steady-state phase. The equation can be formulated as a linear mix between old estimate and linearly transformed observation. In the case of uncorrelated states and direct observation of states, e.g. with identity observation matrix, the equation reduces down to independent linear mixes for each state with its observation, which is exactly exponentially weighted moving average (EWMA) filtering (Haugen, 2008). EWMA filter gain parameters are scalars going from zero to one. We chose the formulation as seen in equation (1) where $K_t = 1$ means maximum observation influence.

The warm-up phase is different as it features a rapidly changing covariance. Expressed in EWMA formulation the filter gain rapidly changes from one to its steady-state value. The result during operation is that the estimated mean value rapidly converges from its initial value to the observed mean. We approximate this behaviour with an EWMA filter by setting the initial value to the first observation, i.e. use the maximum of one as gain, and then continue with steady-state phase.

The exponentially weighted moving average equation (1) is a discrete time low pass filter. The output x_k at step k is computed by a linear weighting between the last filter output x_{k-1} and the current observation value y_k.

$$x_k = y_k \cdot K_t + (1 - K_t) \cdot x_{k-1}. \tag{1}$$

Usually this filter is used with constant time step between updates. In this case the filter behaviour is equal to that of a discretized first order - or time-constant – filter (Haugen, 2008). When the time step is varying, the corresponding filter gain K_t needs to be updated for equal filter behavior, i.e. filter time constant (Hünermund, 2020): Using equations from Haugen *et al.* (2008) it can be shown that the filter gain depending on the actual time step is computed according to equation (2). To support time varying conditions, which occur in real world applications, the exponentially weighted moving average filter is specified by a (K_T, T) tuple. The corresponding filter time constants, corresponding to $T_f s$ in (Haugen, 2008), of (K_T, T) and (K_t, t) are identical.

$$K_t = \frac{T}{\frac{t \cdot (1 - K_T)}{K_T} + T}, \tag{2}$$

K_t is filter gain for actual time step t at each update;
K_T is filter gain for time step T.

The reference time step T can be equal for each filter to reduce the number of parameters. Alternatively, the filter time constants can be used. We prefer tuning the gain parameter because it is bounded between zero and one. The core assumption for approximating Kalman filter with EWMA filter

of constant process and measurement noise does not hold under wildly jittered detection rates or on occasional detector out-takes.

4. Evaluation

Using the proposed system, we achieve, with comparable evaluation metrics, framerates up to ten times higher than the fastest MOT schemes publicly listed for the MOT challenges MOT17 (Table 1) and MOT20 (Table 2). Those challenges are axis-aligned bounding-box multi object tracking challenges (Milan, 2016) (Dendorfer, 2020). For our application of collision avoidance for an autonomous e-bike we tracked rotated rectangles which can also perform simple AABB tracking. For association and matching we use the Jonker-Volgenant algorithm (Jonker, 1987). The implementation is based on the linear sum assignment function used in the library scipy (Virtanen, 2020), but improved for better memory footprint by avoiding unnecessary memory reallocations. The association metric is a weighted sum of IoU, centre distance and area ratio.

We compare to the fasted reported schemes. Our implementation is faster than all other entries. The other object tracking metrics are comparable. Comparing to the fastest other schemes, we win in the metrics speed (Hz) and "Mostly Lost" (ML*). The ML metric counts the number of ground-truth tracks that where (mostly) lost during tracking.

Table 1. Relevant Evaluation Metrics in MOT-17 Benchmark.Ours: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz – Single Core Usage (higher is better except for metrics with *)

MOT-17 Metric	Ours	IOU17	MAA	SORT17
Hz	**10728**	1523	189	143
HOTA	0.38	0.39	**0.62**	0.40
MOTA	0.07	0.45	**0.79**	0.43
FN*	125113	281643	**77661**	287582
FP*	182954	**19993**	37320	28398
IDSW*	5191	5988	**1452**	7127
MT	522	369	**1356**	295
ML*	**354**	953	573	997

The high level metrics HOTA and MOTA balance the other metrics into single numbers. On MOT20 we out-perform the other entries w.r.t to MOTA, but on MOT17 we get very low results for MOTA. Our scheme is tuned for low FN rates at the cost of high FP and IDSW rates. HOTA and MOTA being balanced metrics don't reward going into either extreme. That could explain the variability of MOTA for our method.

In the context of collision avoidance for mobile systems we want the tracking system to be fast and not miss any potential obstacles. In terms of metrics this means low false negatives rates are preferred to low false positive rates. Our proposed system performs best in the relevant metrics speed and "mostly lost".

Table 2. Relevant Evaluation Metrics in MOT-20 Benchmark Ours: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz – Single Core Usage (higher is better except for metrics with *)

MOT-20 Metric	Ours	CTv0	SORT20	IOU KMM	OCSort
Hz	**447**	318	57	30	28
HOTA	0.44	0.33	0.36	0.40	**0.54**
MOTA	**0.63**	0.45	0.43	0.46	0.60
FN*	352104	207927	264694	214777	**202502**
FP*	59437	69491	27521	57517	**4434**
IDSW*	7209	6492	4470	4509	**554**
MT	**903**	409	208	371	478
ML*	**222**	235	326	244	330

5. Conclusions

We proposed a timeline based modularized multi sensor capable multi object tracking framework with runtime optimized implementation. We evaluated the system in MOT benchmarks and got good results in the metrics relevant for our use case of collision avoidance. We applied the proposed general system to the on-board obstacle tracking of an autonomous bike, developed in the context of the research project RavE-Bike, fusing detections from mobile and stationary geo-referenced sensors.

Data oriented design during implementation is a good foundation for a fast runtime. Speed frees up resources for sophisticated detectors or other system components and reduces overall energy consumption. There are still low-hanging fruits for improving memory footprint with data oriented design. For example, we still use a lot of array-of-structs (AoS) instead of struct-of-arrays (SoA), despite using SoA being a core technique in data oriented design. Memory reusage and avoidance of scattered reallocation proved to be very important. In earlier implementation iterations with dynamically allocated tracks over 90% of runtime was spent on memory related overhead. Similarly, we had first modularized the system using inheritance and virtual dispatch of interface functions. This resulted in much slower runtimes as the compiler could not remove function call overhead because of virtual dispatch. Using function calls for modularizing the system thus resulted in reduced runtimes. This was a major frustration during the design process as it resulted in a trade-off between speed and modularity. After switching to metaprogramming for modularization the compiler could make use of function inlining, erasing all function call overhead. Thus we were able to achieve both a general and modularized but also a very fast performing multi object tracking system.

So far we used an implementation of the Jonker-Volgenant algorithm (Jonker, 1987) with cubic time complexity for matching. We see good opportunities for further runtime improvements by providing association and matcher components using spatial querying for match candidate generation as recently suggested by Lohn-Jaramillo (2022). This approach should also greatly benefit from parallelization.

Acknowledgements

The authors acknowledge the Federal Ministry of Education and Research Germany for financial support within the research project RavE-Bike (FKZ 03VNE2063A-D). The project is funded as a joint pro-ject by the Education and Research (BMBF) based on the guidelines for the promotion of SME-centered, strategic R&D networks in networks and clusters (KMU-NetC). The project is being done within the ZIM network „Assistance in Logistics". Project partners are: Pedalpower GmbH, Thorsis Technologies GmbH, Otto-von-Guericke Universität Magdeburg, Fraunhofer-Institut für Fabrikbetrieb und -automatisierung IFF und Zentrum für Produkt-, Verfahrens- und Prozeßinnovation GmbH.



References

1. Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016, September) Simple online and realtime tracking. In: *2016 IEEE international conference on image processing (ICIP)*, 3464-3468. IEEE.
2. Bochinski, E., Eiselein, V., & Sikora, T. (2017, August) High-speed tracking-by-detection without using image information. In: *2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS)*, 1-6. IEEE.
3. Cao, J., Weng, X., Khirodkar, R., Pang, J., & Kitani, K. (2022) Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking. *arXiv preprint arXiv*:2203.14360.
4. Dendorfer, P., Rezatofighi, H., Milan, A., Shi, J., Cremers, D., Reid, I., ... & Leal-Taixé, L. (2020) Mot20: A benchmark for multi object tracking in crowded scenes. *arXiv preprint arXiv*:2003.09003.
5. Haugen, F. (2008) *Derivation of a discrete-time lowpass filter. Differentiation*.
6. Hünermund, M., Groneberg, M., & Schütz, A. (2021) Fast connected components object segmentation on fused lidar and stereo-camera point clouds with visual-inertial-gimbal for mobile applications utilizing GPU acceleration. In: *International Conference on Reliability and Statistics in Transportation and Communication*, 73-83. Springer, Cham.

7. Jonker, R., & Volgenant, A. (1987) A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4), 325-340.
8. Lohn-Jaramillo, J., Ray, L., Granger, R., & Bowen, E. (2022) *Clustertracker: An Efficiency-Focused Multiple Object Tracking Method*. Available at SSRN 4102945.
9. Milan, A., Leal-Taixé, L., Reid, I., Roth, S., & Schindler, K. (2016) MOT16: A benchmark for multi-object tracking. *arXiv preprint arXiv*:1603.00831.
10. Museth, K. (2013) VDB: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)*, 32(3), 1-22.
11. Reinholtz, K. (2012) Timeline as Unifying Concept for Spacecraft Operations. In: *SpaceOps 2012*, 1274906.
12. Stadler, D., & Beyerer, J. (2022) Modelling ambiguous assignments for multi-person tracking in crowds. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 133-142.
13. Urbann, O., Bredtmann, O., Otten, M., Richter, J. P., Bauer, T., & Zibriczky, D. (2021) Online and real-time tracking in a surveillance scenario. *arXiv preprint arXiv*:2106.01153.
14. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & Van Mulbregt, P. (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*, 17(3), 261-272.
15. Wang, Z., Zheng, L., Liu, Y., Li, Y., & Wang, S. (2020, August) Towards real-time multi-object tracking. In: *European Conference on Computer Vision*, 107-122. Springer, Cham.
16. Wojke, N., Bewley, A., & Paulus, D. (2017, September) Simple online and realtime tracking with a deep association metric. In: *2017 IEEE international conference on image processing (ICIP)*, 3645-3649. IEEE.
17. Zhang, Y., Wang, C., Wang, X., Zeng, W., & Liu, W. (2021) Fairmot: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision*, 129(11), 3069-3087.
18. Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., ... & Wang, X. (2022) Bytetrack: Multi-object tracking by associating every detection box. In: *European Conference on Computer Vision*, 1-21. Springer, Cham.