Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Artificial Intelligence

AI-based dynamic environment detection and movement tracking for XR applications

Master's Thesis

Author
Péter Szögi

Advisor
Attila Török
János Dóka
dr. Balázs Sonkoly

December 15, 2024

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Szögi Péter*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2024. december 15.

Szögi Péter
hallgató

DEPARTMENT HEAD

MSc Thesis Task Description

Péter Szögi
candidate for MSc degree in Computer Engineering

AI-based dynamic environment detection and movement tracking for XR applications

Movement tracking plays an important role in visual surveillance, autonomous driving or sports, and nowadays it is handled with AI-based computer vision techniques. As with the advent of new types of sensors and boosted computing power the XR (Extended Reality) applications emerge, the need for accurate and efficient movement tracking solutions also appears. Unfortunately, the top performing object tracking algorithms are quite resource hungry (running on GPUs with only a few frame/sec), therefore their adaptation in XR systems is yet unfeasible. Fully offloading these tasks to edge cloud infrastructure is also challenging due to the stringent delay requirements of XR applications. Luckily, the special sensors (RGB and depth from ToF/RGB-D cameras) integrated in XR devices and the rise of semantic SLAM (Simultaneous Localization and Mapping) solutions can provide some new, powerful tools, presenting an opportunity to create novel object tracking solutions for XR. Integrating semantic knowledge into object tracking solutions could improve the performance, precision and robustness of the algorithms, allowing the internal components to provide context-aware motion modeling and occlusion handling, reasoning about the proper interaction and exclusion models or providing a scenario-tailored inference model for tracking.

The student's task is to provide object tracking solutions, enhanced by semantic knowledge, targeting the field of XR applications. A hypothetical XR application (e.g. a multi-user game) should be envisioned to customize and integrate the semantic knowledge with the tracking solution.

Tasks to be performed by the student shall include:

- Present the main directions in object tracking and survey the research field by focusing on algorithms applicable in XR environments; respectively study their main components in terms of the target application!
- Select a proper object tracking solution, investigate how the semantic knowledge can be incorporated in the different components of the algorithm!
- Implement and enhance the components which enable the semantic understanding and allow a context-aware handling of the tracking task!
- Evaluate the performance of the solution making use of appropriate datasets and scenarios!
- Present the technical results in detail!

Supervisor at the department:	Dr. Balázs Sonkoly, associate professor
	János Dóka, research assistant
External supervisor:	Attila Török, researcher (TKI)

Budapest, 4 March 2024

Dr. Pál Varga
head of department

Kivonat

A dinamikus környezetérzékelés és mozgáskövetés már kulcsszerepet tölt be az autonóm járművek, a robotika és a kibővített valóság (extended realiy - XR) világában. Segítségükkel ezek a rendszerek valós időben érzékelni és elemezni tudják a környezetükben bekövetkezett változásokat, valamint nyomon tudják követni a körülöttük mozgó objektumok helyzetét, sebességét és irányát. A kihívások között szerepel a valós idejű feldolgozás, a zajos adatok kezelése és bizonyos eszközökön a szűkös számítási kapacitás is.

Ebben a dolgozatban egy dinamikus környezetérzékelést és mozgáskövetést megvalósító rendszer kerül bemutatásra, amely kifejezetten XR alkalmazások támogatására lett kifejlesztve, figyelembe véve ezek késleltetéskövetelményeit és erőforrásait. A rendszer része egy szemantikailag továbbfejlesztett követési keretrendszer, amely az egyidejű lokalizációt és feltérképezést (SLAM), valamint könnyűsúlyú követési algoritmusokat integrálva biztosít robusztus, valós idejű teljesítményt. Az előterjesztett megoldás olyan kihívásokkal foglalkozik, mint a szigorú késleltetési követelmények, az objektumok takarásának kezelése, valamint a dinamikus környezet megértése, amelyek kritikusak az XR élmények szempontjából.

A kutatás átfogó áttekintést nyújtott az objektumkövetési módszertanokról, és egy klasszikus és modern technikákat ötvöző keretrendszer fejlesztésével zárult. A teljesítményértékelések azt mutatták, hogy a keretrendszer megbízható követést biztosít ellenőrzött körülmények között, de kihívásokkal szembesült nagy sebességű mozgások és összetett környezetek esetén. Ezek az eredmények rámutatnak a szemantikai információk objektumkövetésbe történő integrálásának lehetőségeire, miközben hangsúlyozzák a további finomítás szükségességét a robusztusság és az alkalmazkodóképesség javítása érdekében dinamikus körülmények között.

Abstract

Dynamic environment understanding and motion tracking already play a key role in fields of autonomous vehicles, robotics and extended reality (XR). These technologies enable systems to detect and analyze changes in their surroundings in real time, as well as track the position, speed, and direction of moving objects around them. Key challenges include real-time processing, handling noisy data, and addressing limited computation resources in certain devices.

This paper presents a system for dynamic environment understanding and motion tracking, with a particular emphasis on applications in extended reality (XR), taking into account their latency requirements and resource constraints. The system features a semantically enhanced tracking framework that integrates Simultaneous Localization and Mapping (SLAM) and lightweight tracking algorithms to deliver robust, real-time performance. The proposed solution addresses challenges such as stringent latency requirements, occlusion handling, and dynamic scene understanding, critical for immersive XR experiences.

The research involved a comprehensive review of object tracking methodologies, emphasizing their applicability to XR environments, and culminated in the development of a novel framework combining classical and modern techniques. Performance evaluations revealed that the framework demonstrated reliable tracking in controlled scenarios but encountered challenges in high-speed motion and complex environments. These findings highlight the potential of integrating semantic insights into object tracking while emphasizing the need for further refinement to improve robustness and adaptability under dynamic conditions.

Chapter 1

Introduction

Evolving Extended Reality (XR) applications necessitate immersive and reflexive ways to participate and interact with the real and virtual world. For example, consider a manufacturing scenario, where the assembly process is guided step-by-step by the XR application, or a household gaming/learning environment for children, where activity/interaction handling between real and virtual characters is based on scene physics/mechanics understanding. Therefore, movement detection, prediction and tracking directly affects the quality of the XR applications and ultimately the user experience, since the interaction between the physical and virtual artifacts will be driven by the positioning of actors and objects of interest.

To adhere these challenges, besides the classical AI-based solutions (Simultaneous Localization and Mapping (SLAM) and computer vision), XR systems require the introduction of scene and activity understanding. These techniques are needed to build an up-to-date semantic knowledge from the surrounding world based upon a higher level of dynamic context interpretation, where the different object types and their spatio-temporal relationships can be infered in real-time and acted upon accordingly. Thus, the demand for accurate and efficient context-aware Multiple Object Tracking (MOT) solutions also becomes imperative.

Current high-performance object tracking algorithms, while effective, are resource-intensive and operate suboptimally in terms of speed, even on graphics processing units (GPUs) they rarely go beyond 20 frames per second. This can be attributed to the use of convolutional neural network based (CNN) Object Detector (OD) algorithms as their fundamental building block. This computational performance limitation poses a significant challenge for their integration into XR systems, which are characterized by their need for real-time processing to ensure a seamless user experience. Additionally, the prospect of fully offloading these tasks to edge cloud infrastructures is hindered by the strict latency requirements intrinsic to XR environments. Since the OD provides a very robust mechanism to cope with challenged use cases their omission from XR-based MOT design is not advisable, the question is if we can use them in a more optimized, sophisticated way.

Classical computer vision algorithms, using pixel level visual characteristics (e.g. contours, corners, line segments) and keypoint based detection and matching, are also frequently used for object tracking and SLAM related tasks. While these solutions are efficient in terms of processing and resource utilization, they are less robust in dynamic environments (perspective changes, rotation and scale variances, lighting conditions, etc.) and in the presence of multiple, often severely occluded objects. To overcome these issues, one should employ a feature point management mechanism, which acts on a frame-by-frame basis, continuously detecting and matching the visual keypoints related to the tracked objects. Now the question is, how we distinguish the keypoints belonging to the objects of interest and the surroundings?

As we point out the careful introduction and proper management of semantic information would allow to employ these different solutions in a unified system, which combines the advantages of both worlds.

This thesis posits that the utilization of specialized sensors, such as RGB and depth sensors (Time of Flight (ToF) and RGB-D cameras) integrated within XR devices, alongside the combination of object detection and semantic SLAM techniques with feature point matching methods, presents a novel opportunity to develop innovative object tracking solutions tailored for XR applications.

Our goals with context-aware MOT design and development are:

- to design mechanisms, which provide object and scene dynamics handling with a layered semantic knowledge built upon the fusion of feature matching and more advanced, AI-based solutions
- to combine highly efficient classical and modern MOT methods, which can be deployed in XR environments
- to design context-aware MOT components that enable observability and adaptability on dynamic object behavior

By integrating lightweight semantic knowledge into object tracking systems, there is a potential to significantly enhance algorithmic performance, precision, and robustness. This approach would enable more effective context-aware appearance and motion modeling, improved occlusion handling, and the provision of customized inference models suited to specific XR scenarios.

The focal point of this work is the development of an object tracking framework that incorporates semantic knowledge, specifically designed for XR applications. This framework will be demonstrated through a hypothetical XR application, such as a multi-user game, which will serve to contextualize and demonstrate the efficacy of the semantic integration within the tracking solutions.

1.1 Chapter Overview

This thesis is structured to progressively build a comprehensive understanding of the challenges, methodologies, and solutions in AI-based dynamic environment detection and movement tracking for XR applications. Below is an overview of the content covered in each chapter:

- **Chapter 2: Theoretical Overview**

  This chapter provides a foundational understanding of the core concepts necessary for XR systems, including the requirements for low-latency object tracking and the key principles of visual object tracking. It categorizes tracking methodologies and introduces advanced algorithms and metrics, with a focus on their applicability to XR environments.
- **Chapter 3: Related Work**

Here, the most relevant prior research and innovative tracking systems are discussed. The chapter highlights key advancements in visual tracking, semantic tracking, and their integration into XR systems. It also identifies the gaps and limitations of existing approaches, setting the stage for the proposed methodology.

- **Chapter 4: Proposed Method**

  This chapter introduces the conceptual framework of the proposed tracking system, emphasizing semantic keypoint management for dynamic environments. It details the architecture, main algorithmic modes, and the integration of lightweight semantic knowledge to enhance performance and robustness for XR applications.

- **Chapter 5: Implementation**

  The implementation chapter focuses on the practical realization of the proposed framework. It discusses the dynamic-static ORB keypoint separation method, the algorithmic steps of the tracker, and adaptations for handling multiple dynamic objects. Key implementation challenges and strategies for achieving real-time performance are also addressed.

- **Chapter 6: Evaluation**

  This chapter outlines the configuration of the test environment and evaluates the proposed solution using a comprehensive set of experiments conducted in controlled conditions. It examines performance under various scenarios, including dynamic object motion, camera displacement, and trajectory testing, while providing insights into the robustness, efficiency, and limitations of the system.

- **Chapter 7: Summary and Future Directions**

  The final chapter summarizes the research findings and their implications for XR applications. It also discusses the system's limitations and proposes potential avenues for future work, aiming to enhance robustness, scalability, and adaptability in complex environments.

Chapter 2

Theoretical overview

2.1 Requirements of XR

The importance of low-latency tracking in augmented reality (AR) and mixed reality (MR) environments lies in its critical role in ensuring a seamless and immersive user experience. Latency in object tracking disrupts the spatial alignment between virtual and real-world objects, leading to a phenomenon known as "misregistration" [19]. Misregistration causes virtual objects to appear to lag behind or shift inconsistently with the user's viewpoint, significantly degrading the perceived realism and usability of the system. Furthermore, high latency exacerbates user discomfort by introducing visual and proprioceptive conflicts, which have been linked to cybersickness symptoms such as nausea and dizziness [39].

Quantitative and qualitative studies have shown that even small delays in object tracking, typically exceeding 20 milliseconds, can lead to noticeable perceptual discrepancies and reduced task performance. It was demonstrated [19] that low-latency tracking minimizes the errors in virtual object placement, ensuring that virtual content remains anchored to its intended position within the real world. This precise alignment enhances the user's sense of presence and interaction fidelity, key factors in the overall effectiveness of AR/MR systems.

Thus, achieving low-latency tracking is not merely a technical requirement but a fundamental necessity for preserving the integrity of the AR/MR experience. The findings underscore the need for robust tracking algorithms and optimized system architectures that prioritize minimal lag to meet the stringent demands of immersive applications.

2.2 Visual object tracking

Object tracking is a fundamental task in computer vision and image processing, often called Visual Object Tracking (VOT). Its purpose is the precise localization and continuous monitoring of an object's position within a sequence of frames. This process is critical in a variety of applications, ranging from surveillance systems, autonomous vehicles, and robotics to augmented reality (AR) and human-computer interaction. By accurately tracking objects, systems can make informed decisions, predict future states, and interact intelligently with their environment.

The importance of object tracking lies in its ability to enable real-time analysis and interpretation of dynamic scenes. For instance, in autonomous driving, object tracking ensures the safe navigation of vehicles by continuously monitoring surrounding objects such as pedestrians, other vehicles, and obstacles. In AR, object tracking allows for the seamless integration of virtual objects into the real world, providing immersive and interactive experiences. Similarly, in surveillance, object tracking facilitates the monitoring of suspicious activities and enhances security measures.

2.2.1 Types of Visual Object Tracking

Object tracking methodologies can be broadly categorized into several types, each suited for different applications and scenarios. These categories include Single Object Tracking (SOT), Multiple Object Tracking (MOT), and the distinction between online, offline, and batch processing methods. Each category addresses specific challenges and requirements inherent to the application context.

2.2.1.1 Single and Multi Object Tracking

Object tracking methodologies can be divided into two primary categories: SOT and MOT. These categories address different scenarios and present unique challenges and techniques to ensure accurate and efficient tracking. Below is a brief overview of each category and its specific focus.

- **Single Object Tracking (SOT):** Focuses on tracking a single target object within the video frames. Methods in this category must efficiently handle challenges such as occlusions, illumination changes, and background clutter. Techniques like correlation filters and deep learning-based approaches are commonly used.
- **Multiple Object Tracking (MOT):** Involves tracking multiple objects simultaneously. This category addresses the complexities of managing the identities of multiple objects as they move and interact within the scene. Approaches in MOT often employ data association strategies with the use of Kalman filters and the Hungarian algorithm for assignment problems.

1. **Detection-free Tracking**:

   Detection-free tracking, or in its perhaps more common name detection-free tracking, is pivotal for tasks where predefined object models or detectors are unavailable or impractical. Algorithms here are designed to adaptively learn and update the object's appearance model on-the-fly, making them highly flexible. Examples include the MedianFlow, MOSSE, and CSRT trackers, which can be found in OpenCV[1] natively. They are typically initialized with a bounding box around the object to be tracked, either provided by the user or identified through some form of scene change detection [9].

   - **Generative trackers** focus on modeling the appearance of the target object. They work by constructing a model of the object and then searching for the region in each frame that best matches this model. Common techniques include template matching and particle filtering. Template matching involves comparing segments of the frame with the stored template of the target object to find the best match, while particle filtering uses a probabilistic approach to estimate the state of the object across frames [9].

[1]OpenCV is a popular computer vision library.

- **Discriminative Trackers** treat the tracking problem as a binary classification task. They focus on distinguishing the target object from the background. These trackers employ classifiers or correlation filters to separate the target from its surroundings. Discriminative trackers are known for their ability to handle occlusions and dynamic backgrounds effectively. Techniques like the Multiple Instance Learning (MIL), Structured Output Tracking with Kernels (Struck), and correlation filter-based methods are prominent in this category [9].
- **Collaborative trackers** combine the strengths of both generative and discriminative methods. These trackers use a collaborative approach to leverage the complementary benefits of both models. For instance, a tracker may use a generative model to generate potential target appearances and a discriminative model to select the best match. This approach enhances robustness and accuracy in challenging tracking scenarios such as occlusions and rapid appearance changes [9].
- **Deep Learning-Based Trackers** utilize convolutional neural networks (CNNs) and other deep learning architectures to learn robust feature representations directly from the data. These trackers can be divided into two categories: those that combine deep features with traditional tracking algorithms and those that implement end-to-end learning frameworks. The deep learning approach allows trackers to adapt to a wide variety of appearance changes and achieve state-of-the-art performance in terms of accuracy and robustness. Examples include Siamese network-based trackers, recurrent neural network (RNN) based trackers, and attention mechanism-based trackers [9].
- **Feature-based Trackers**

2. **Detection-based Tracking**

   The term "detection-based" is seldom used, as utilizing some form of detection model prior to tracking is a standard practice. However, grouping the following two methods separately from detection-free methods under this term provides greater clarity.

   - **Tracking-by-Detection (TBD)** is the type of object tracking approach, where, as the name suggests, object detection runs every frame. The tracker processes the detection results while incorporating information from previous frames to maintain continuity. In the context of Multiple Object Tracking, this typically involves using a Kalman filter to predict object locations in the frame preceding the current one, followed by associating detected object identities across frames using the Hungarian algorithm.
   - **Joint Detection and Tracking (JDT)** utilizes advanced techniques such as backbone architectures, object detection, attention mechanisms, graph convolutional networks, and deep convolutional networks for end-to-end training [16]. It differs from traditional tracking-by-detection methods by creating more reliable tracklets[2], enforcing temporal consistency, and not requiring a data association mechanism for a given set of frames. It is considered more efficient since it does the detection and tracking in 1 single step, but despite that, JDT is still more tasking computationally for real-time applications.

[2]A tracklet is a short sequence of detections that represents the movement of an object over a brief period.

2.2.1.2 Process Methods

In visual object tracking, the methods used to process video frames can significantly impact the tracking performance and application suitability. These methods can be categorized based on how they handle and process the video data.

- **Online** object trackers process video frames sequentially in real-time. These trackers update their model with each incoming frame, allowing them to adapt dynamically to changes in the object's appearance, lighting conditions, and occlusions.
- **Offline** object trackers operate on pre-recorded video sequences where the entire video is available before tracking begins, as shown at the right side of figure 2.1. These trackers can utilize global information from the entire sequence to improve tracking accuracy.
- **Batch** object trackers bridge the gap between online and offline trackers by processing video frames in fixed-size batches. These trackers aim to balance real-time adaptability with the accuracy of global information.

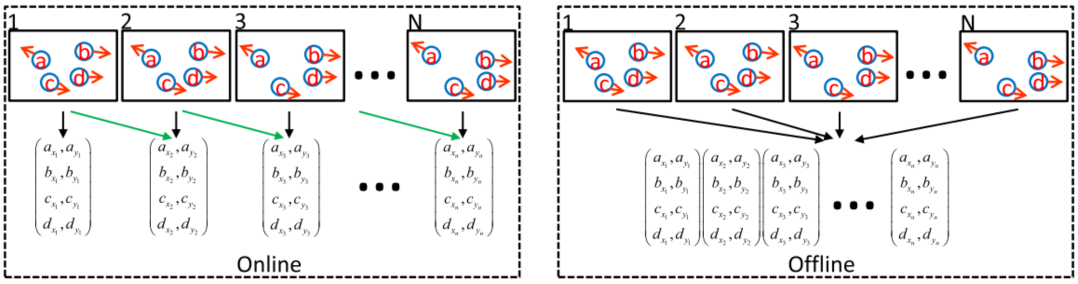

Figure 2.1: Online Tracking on the left, while offline tracking on the right [28].

2.2.1.3 Number of Cameras

Another significant categorization in object tracking is based on the number of cameras involved. Single-Camera Tracking (SCT) and Multi-Camera Tracking (MCT) are two distinct approaches that cater to different surveillance and monitoring requirements [21]. SCT is typically simpler and involves tracking objects within the field of view of a single camera. In contrast, MCT leverages multiple cameras to provide a comprehensive view of the scene, enhancing tracking accuracy and robustness.

- **Single-Camera Tracking (SCT)**

  Object tracking can be accomplished using video streams from a single camera, referred to as single-camera tracking. In such tracking solutions, the video stream may come from either stationary or moving cameras. When dealing with stationary cameras, there is no need for camera motion compensation; however, a motion model is generally required to predict the target object's movements in subsequent frames. Conversely, for moving cameras, both camera motion compensation and an object motion model are necessary. Many deep learning-based approaches in object tracking aim to enhance the accuracy and efficiency of SCT by addressing these challenges.

- **Multi-camera tracking (MCT)**

  Multi-camera tracking involves processing video streams from multiple cameras to track and associate objects across different views. This approach is inherently more complex than SCT. MCT typically employs SCT techniques to track objects within individual video streams and then performs an additional step to associate these local tracks across multiple cameras. MCT scenarios can involve either overlapping or non-overlapping fields-of-view (FOV). In overlapping FOV settings, common in smaller areas like rooms, objects are simultaneously visible to multiple cameras, allowing for the use of additional information such as ground-plane coordinates and movement patterns to match local tracks. In non-overlapping FOV settings, where this extra information is unavailable, a re-identification (ReID) module is required. The ReID module compares images or videos of targets from different cameras, computing similarity scores to match local tracks. If the similarity score is sufficiently high, the local tracks from different cameras are combined into a single global track.

2.2.2 Tracking Metrics

There are quite a few metrics for evaluating tracking algorithms and solutions performance. The purpose of this section is to make this field more clear and understandable.

The first 2 metrics to discuss are MOTA and MOTP [3] they are internationally accepted as CLEAR metrics [42]. In the following lists there is either ↑ or ↓ after the name of the metric. The ↑ indicates that the performance is better when the number is bigger, while ↓ meaning the other way around.

2.2.2.1 Spatial Accuracy

These metrics evaluate how accurately the tracked object's locations are estimated relative to their true positions in the video frames.

- **Intersubsection over Union (IoU)** [34] ↑

  IoU (also known as Jaccard Index) is a primary metric used especially in visual tracking to assess the accuracy of object detectors employed within the tracking systems. IoU measures the overlap between the predicted bounding box and the ground-truth bounding box, expressing this as the ratio of their intersubsection area to their union area. A higher IoU score indicates better tracking accuracy, as it reflects a greater overlap between the predicted and actual object positions.

$$\text{IoU} = \frac{\text{Area of Overlap between Predicted and Ground Truth Boxes}}{\text{Area of Union between Predicted and Ground Truth Boxes}} \tag{2.1}$$

- **Multiple Object Tracking Precision (MOTP)** [3] ↑

  MOTP quantifies the accuracy of object localization in tracking scenarios by measuring the average Euclidean distance between all correctly matched predicted positions and their corresponding ground truth targets. This metric evaluates the tracker's ability to estimate precise object positions across frames, contingent on successful object identification and matching. While MOTP provides valuable insights into localization accuracy, it does not reflect other critical aspects of tracking performance such as the ability to minimize identity switches, false positives, or missed targets, thus offering limited insight into the tracker's overall effectiveness. It is calculated with the formula in equation 2.2.

$$\text{MOTP} = \frac{\sum_{t,i} d_{t,i}}{\sum_t m_t} \tag{2.2}$$

Where $d_{t,i}$ represents the distance between the predicted and the actual position of object i in frame t, and m_t is the number of matched objects in frame t.

2.2.2.2 Detection Quality

These metrics assess the ability of a tracking system to correctly detect objects and avoid false detections.

- **Tracking Precision (Pr)** [14] $\uparrow$

  Precision measures the proportion of correctly tracked objects among all the tracked objects, essentially quantifying the absence of false positives.

$$\text{Precision (Pr)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{2.3}$$

- **Recall (Re)** [14] $\uparrow$

  Recall assesses the proportion of actual tracked objects that were successfully detected by the tracker, highlighting the tracker's ability to minimize missed detections.

$$\text{Recall (Re)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2.4}$$

2.2.2.3 Overall Tracking Performance

These metrics provide a holistic view of the tracker's performance, integrating aspects of detection accuracy, identity maintenance, and sometimes spatial accuracy.

- **Multiple Object Tracking Accuracy (MOTA)** [3] $\uparrow$

  MOTA is a critical metric used to evaluate the overall accuracy of object tracking algorithms. It integrates three main error sources: false positives, missed targets, and identity switches, to provide a comprehensive score. MOTA is calculated by subtracting the sum of all tracking errors (normalized by the total number of ground-truth objects) from 1, as shown at equation 2.5. Specifically, it considers the ratio of missed detections (objects that should have been detected but were not), false positives (objects wrongly added in the scene), and identity switches (times an object's identity is incorrectly changed) relative to the total number of actual objects. This metric is essential in the fields of computer vision and video processing because it reflects how well a tracking system maintains consistent and accurate object identities over time, making it a standard for assessing the performance of tracking algorithms in various applications, from autonomous driving to surveillance and sports analytics.

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FP}_t + \text{FN}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t} \tag{2.5}$$

Here, FP_t is the number of false positives, FN_t is the number of false negatives (missed targets) IDSW_t represents the number of identity switches, and GT_t is the total number of ground truth objects in frame t

- **Average Multi-Object Tracking Accuracy (AMOTA)** [48] ↑

  AMOTA is an extension of the MOTA metric that averages the MOTA scores at different recall levels to provide a more robust evaluation of a tracking system across varying levels of detection difficulty. This metric addresses the limitation of MOTA being overly sensitive to the number of detections by normalizing performance across a range of detection thresholds. AMOTA thus gives a comprehensive view of a tracker's performance under varied conditions.

  $$\mathrm{AMOTA} = \frac{1}{R} \sum_{r \in [0,1]} \mathrm{MOTA}(r) \tag{2.6}$$

  In equation 2.6 r denotes different recall levels and R is the number of recall points considered.

- **scaled Average Multi-Object Tracking Accuracy (sAMOTA)** [48] ↑

  sAMOTA scales the AMOTA score to a range between 0 and 1, making it easier to compare performances across different datasets or tracking challenges. This scaling ensures that the metric remains consistent and comparable across different testing scenarios.

- **Higher Order Tracking Accuracy (HOTA)** [27] ↑

  HOTA balances the importance of detecting all objects (detection accuracy) and keeping correct track identities over time (association accuracy). Unlike MOTA, which predominantly focuses on detection, HOTA equally weights the association of identities across frames. This metric provides a holistic measure of performance, making it particularly suitable for applications where maintaining identity is crucial.

  $$\mathrm{HOTA} = \sqrt{\frac{\mathrm{DetA} \times \mathrm{AssA}}{\mathrm{DetA} + \mathrm{AssA} - \mathrm{DetA} \times \mathrm{AssA}}} \tag{2.7}$$

  Here, in equation 2.7 DetA represents detection accuracy and AssA is association accuracy.

2.2.2.4 Identity Maintenance

These metrics specifically focus on how well the tracking system maintains consistent identities of the objects across frames.

- **Identity Switches (IDs, IDSWs)** [28] ↓

  IDs is a critical evaluation parameter in multi-object tracking. An Identity Switch occurs when a tracking algorithm incorrectly changes the identity of a tracked object from one identity to another. This metric is crucial for assessing the consistency and reliability of tracking algorithms in maintaining object identities over time. IDs is often reported alongside MOTA to provide insights into the quality of trajectory handling by the tracking algorithm. High numbers of identity switches can significantly degrade user trust in a tracking system, especially in applications like surveillance or sports analytics, where consistent object identity is crucial.

- **IDF1** [35] $\uparrow$

  IDF1 is a metric introduced to evaluate the accuracy of identity preservation in multi-object tracking. It measures the ratio of correctly identified detections over the average number of ground-truth and computed detections. It balances the precision and recall of identity assignments, providing a comprehensive view of how effectively a tracker maintains true identities across different frames.

$$\text{IDF1} = \frac{2 \times \text{IDTP}}{\text{IDTP} + \text{IDFP} + \text{IDFN}} \tag{2.8}$$

  In the above 2.8 equation IDTP (Identity True Positives) are correctly associated identities, IDTP (Identity False Positives) are wrong identity associations considered as matches, and IDFN (Identity False Negatives) are missed matches.

2.2.2.5 Continuity and Trajectory Integrity

These metrics evaluate the continuity and integrity of object trajectories throughout a tracking sequence.

- **Success Rate** $\uparrow$

  The Success Rate in tracking is typically used in single object tracking scenarios to measure the percentage of time a tracker successfully keeps the target within the predefined bounds of accuracy, often defined by an overlap threshold. It is commonly used in visual tracking benchmarks to evaluate how often a tracker succeeds in following an object despite challenges like occlusion, motion blur, or lighting changes.

$$\text{Success Rate} = \frac{\text{Number of Successful Frames}}{\text{Total Number of Frames}} \tag{2.9}$$

  In equation 2.9 the intuitive formula of Success Rate is shown. It is a direct measure of a tracker's effectiveness over the course of a video or a series of frames.

- **Fragmentation (FRAG)** $\downarrow$

  Fragmentation is used to assess the integrity of tracking trajectories in a multi-object tracking scenario. Fragmentation occurs when a single object track is interrupted and resumed multiple times, effectively breaking a single trajectory into several fragmented parts. This metric is indicative of the tracking system's ability to handle occlusions and interactions among objects without losing track continuity.

2.3 ORB

The key elements of the ORB [13] (Oriented FAST and Rotated BRIEF) algorithm are keypoints (KPs) and descriptors. Keypoints are specific points in the images that are detected based on features such as edges and corners, identified by their position, scale, and orientation. Descriptors describe the surroundings of these keypoints using a vector-based representation, usually generated by feature detection algorithms. The matchers that are more universal compare descriptors between different images, disregarding the position of the keypoints during the matching process.

The steps of the ORB algorithm are as follows: detection of keypoints, computation of descriptors (time-consuming), and comparison of descriptors (usually using a brute force approach). The algorithm also supports image pyramids, which provide scale invariance, allowing keypoints to be extracted from different pyramid levels at varying levels of detail.

There are two main matcher methods for comparing descriptors:

- **Brute Force Matcher (BFMatcher)**: This method compares all descriptors between images pairwise. It is particularly fast for binary descriptors, especially when using Hamming distances. Symmetric matching, where matching is performed from both sides, provides more robust results. This matcher is suitable for real-time systems with a moderate number of keypoints (e.g., fewer than 2000).
- **FLANN-based Matcher**: The FLANN-based matcher utilizes distances between descriptors and nearest neighbors, making it useful for larger keypoint sets. However, symmetric matching with FLANN is carried out through two-way knnMatch, which is not as precise as brute force but significantly faster when dealing with larger sets.

The quality and quantity of keypoints are critical in real-time algorithms. For small objects (e.g., toy cars), 50-500 keypoints may be sufficient. Dynamic management of the number of keypoints, depending on movement and the number of objects, is essential to maintain speed. To enhance robustness, several techniques can be applied, such as filtering through a homography model or validating keypoints based on orientation and scale.

Reviewing the quality of keypoints and descriptors can contribute to the overall efficiency of the algorithm, ensuring that low-quality or false keypoints do not replace relevant ones. Therefore, multiple filtering steps and selection strategies are employed to ensure that keypoints have the desired quality during tracking and object detection.

2.4 Tessellations

Tessellations are the set of keypoints appearing in the 2D space (for example on the camera image), and are formed from 2D pixel coordinates. Due to the perspective issues, this representation does not correspond to their actual spatial positioning, but it is suitable for gaining insight into the distribution characteristics of the detected KPs within the image (e.g., whether they are distributed along the depth or perpendicular to it) and for comparing them in a straightforward manner with the tessellation of the object model.

2.4.1 Principal Component Analysis (PCA) and Ellipsoid Similarity Metrics

Principal Component Analysis (PCA) is a statistical method that allows for dimensionality reduction of high-dimensional data, transforming it into a set of orthogonal components called principal components. These principal components are arranged in decreasing order of variance within the dataset. PCA enables data to be aligned along the directions of greatest variance, providing a comprehensive view of the complex distribution of the data. In this study, PCA was used to analyze and compare the spatial distribution of keypoint sets extracted from two images. Using PCA, we identified the main directions and spread of the two keypoint sets, allowing comparison of their shape, orientation, and overlap through the construction of ellipsoids.

During PCA-based analysis, we determined the principal components and their corresponding eigenvalues for both keypoint sets, which represent the primary axes and variances of the ellipsoidal distributions that approximate the shapes of these sets. Based on these ellipsoidal representations, several metrics were introduced to quantitatively evaluate the similarity and alignment of the distributions. These metrics include the **Distance Intersection over Union (DIoU)**, **Inlier Ratio**, **Distance Metrics**, **Variance Alignment**, and **Aspect Ratio Comparison**. Each metric aims to capture different aspects of the spatial relationship and structural similarity of the two distributions.

- **Distance Intersection over Union (DIoU)**: DIoU, applied to ellipsoids in this study, measures the overlap and spatial alignment of ellipsoids. This metric is derived from the standard IoU measure, enhanced by a penalty term based on the Euclidean distance between the centers of the ellipsoids. DIoU can serve as a primary metric for assessing the positional and shape alignment of the two distributions' ellipsoidal approximations. High DIoU values indicate significant overlap and proximity between the ellipsoidal distributions, signifying strong similarity.
- **Inlier Ratio**: This ratio indicates the percentage of points from the second keypoint set (Set B) that fall within the ellipsoid boundaries defined by the first set (Set A). The Inlier Ratio provides information about the spatial inclusion and similarity of the distributions; a higher ratio indicates that the keypoints of Set B closely follow the spatial boundaries defined by Set A.
- **Average and Standard Deviation of Distances to Ellipsoid Center**: This metric represents the average distance of points from Set B to the center of the ellipsoid representing Set A in PCA analysis. Along with the corresponding standard deviation, it indicates how concentrated the points of Set B are around the center of Set A's distribution. Lower values suggest that points in Set B are closer to the center of Set A, whereas higher values indicate greater divergence.
- **Variance Alignment Ratio**: The Variance Alignment Ratio compares the spread of points in Set B with respect to the principal directions of Set A. By projecting Set B onto the principal component axes of Set A and calculating the variance along each axis, the ratio of variances along the first and second principal components is determined. This ratio reflects the shape similarity of the distributions, with values close to 1 indicating similar elongations, while significant deviations suggest differences in shape.
- **Aspect Ratio Comparison**: The aspect ratios of ellipsoids were used to evaluate the structural alignment between Sets A and B. Aspect ratio, calculated as the square root of the ratio of eigenvalues, indicates the elongation of the ellipsoids. Comparing these values provides insight into whether the two keypoint sets share a similar shape or if one distribution is more elongated than the other.
- **Variance Coverage**: Variance Coverage evaluates the alignment of variance between Set B and Set A by comparing variance ratios along Set A's principal components with the corresponding values of Set B. A high coverage value indicates that Set B extends to a similar degree as Set A within the PCA-transformed space, signifying consistency in the size and spread of the distributions.

These metrics collectively allow for a comprehensive evaluation of the spatial and structural similarity between the two keypoint sets. Representing the keypoints with ellipsoidal distributions and employing metrics like DIoU, inlier ratio, and variance alignment provides an effective framework for assessing overlap and shape alignment. These metrics facilitate a robust comparison between distributions, especially in applications where the geometric consistency of keypoints extracted from images is critical, thus supporting consistent conclusions regarding the similarity of keypoint sets based on their spatial configuration.

Chapter 3

Related Work

In this chapter the most important techniques and semantic tracker related solutions are presented, which show interesting characteristics, provide basic toolsets for building a real-time semantic tracker for dynamic MR applications. In the first part we provide a general view on proposals which use feature matching at different extent for tracking/pose refinement purposes, then we present a few important trackers where semantic information is also included. The second part of the chapter deals with modern, mostly tracking-by-detection solutions which could be used as a back-end (as association, motion modeling and tracklet manager) in our integrated tracker.

Since our proposal uses a combination of object detectors with feature matching to extract and maintain the scenario related semantic knowledge we explore the related concepts and solutions, with the remark that usually all these systems are also object detector dependent. These modern day trackers assume a detector as a front-end module, a spatial information provider (2D/3D bounding boxes of the tracked object exemplars), which is then associated with the objects managed by the system. However, we use the object detector quite differently than these solutions, thus their exact characteristics are irrelevant at some point, so their detailed presentation can be omitted. Also, due to the constraints of MR applications, we don't investigate those modern CNN-based feature matching solutions (e.g. SuperGlue [37], SuperPoint [11], which albeit provide outstanding matching performance, require GPU-based hardware and run with a very moderate FPS (around 10-15 FPS). The presentation of some practical tracking solutions, like KCF (Kernelized Correlation Filters), CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) or MOSSE (Minimum Output Sum of Squared Error) [18, 47] is also omitted, since these trackers focus on fixed camera settings, single object tracking, and are used in static scenarios where perspective changes, complex occlusions are usually absent. The adaptation of these solutions for our requirements (multiple object tracking, dynamic scenarios, severe occlusions) would be rather painful and cumbersome.

3.1 Keypoint Matching in SLAM, Tracking and Pose Refinement

The most popular use case of ORB is related to SLAM (Simultaneous Localization and Mapping) [31], where camera tracking is performed based on the collection and spatial graph based optimization of the ORB keypoints detected and matched in the environment. It is important to notice, that the goal here is to recognize and track the topological characteristics, in order to determine the camera position and to build a feature point based map representation of the environment, not to detect and track the dynamic objects in it, which requires a completely different toolset. Usually on top of the classical SLAM algorithms (ORB-SLAM [31], RTABMap [22]) are built the so called semantic SLAM solutions (e.g. Kimera [36], PanopticFusion [32], where the map data is further extended with semantic information, the different categories of recognized objects (e.g. indoor: furniture types, building parts, like corridors, stairs; outdoor: buildings, vegetation, etc.) in the mapped environment. The recognition and labeling is usually provided as a subsequent operation, using the recorded video stream/point cloud, since it requires CNN-based object detection algorithms (e.g. YOLO [33] Mask R-CNN [17]), which are resource heavy and can run with 5-30 FPS. Some of these semantic SLAM solutions also employ dynamic object detection (SLAMANTIC [38], Mask-SLAM [20], in order to filter out the keypoints belonging to moving objects, which would result in invalid maps and erroneous localization during subsequent use. However, the detectors are used on a keyframe basis, which also degrades the system performance. Since these solutions are less revelant for our use case (e.g. provide useless semantic data types, cannot track multiple objects in real-time), considering the most demanding requirements of real-time MR tracking applications, we omit their detailed exploration here.

In some recent papers [44, 57, 56], ORB is used for robot or UAV localization/navigation, further improving its characteristics in low light conditions or poorly textured environments. These solutions generally improve upon the classic two-level solutions, where an initial coarse and then an advanced RANSAC-based algorithm is used to optimize keypoint associations between frames and filter out redundant matches and outliers. Their common characteristic is that keypoint similarity checks are performed sequentially in the Hamming space of associations and the Euclidean space of local image structures, thereby creating a multi-level association strategy. Thus, they primarily examine geometric constraints, the ORB descriptors (e.g. angle, octave, response, size) and their characteristics considering the dynamic and static objects in the environment are not considered in these solutions. For these use cases, it is allowed to use more complex algorithms, since the decrease in image processing performance (usually about 10-30 FPS) still makes them useful for robotic applications.

Among the classical multiple object tracking (MOT) solutions for 2D objects the feature point based methods play a prominent role in matching and associating keypoints of the dynamic objects. In the following we present some of these solutions. In [10] the authors present a 2D tracker solution, where ORB is used to match and associate feature points extracted from adjacent RGB frames. The ORB feature matcher is improved with a velocity based keypoint distance calculation method, which allows to filter out mismatches, thus approving data association accuracy. The feature points are extracted using a detection algorithm (E-McGM), which extracts the edge of the object sequences and separates the background using motion information. The contour of the object is recovered from edge segments, these are connected to provide the segments of the object. Thus, the ORB keypoints are extracted from the detected object contours, which makes the solution dependent on a relatively slow performing algorithm. The extracted and matched keypoints are only validated with a 2D point based velocity model, thus, the perspective and dynamic scene changing issues still exist for the 3D use cases. The solution runs at 12-15 FPS, without the contour detector, this also indicates its inapplicability in MR use cases. The solution presented in [52] uses the improved version of AKAZE to match point pairs, where the real-time image is matched with a benchmark image. Descriptors are extracted and matched, the best matching points are selected based on the fusion of Hamming distance and matching line angle. RANSAC is used to calculate the homography matrix between images. The algorithm uses a Jetson Nano B01 board with Nvidia GPU, which implies that the algorithm is resource demanding. The RANSAC algorithm uses homography, which is a 2D-2D based geometric transformation method, it is also not directly applicable for the dynamic 3D scenes. In [51] keypoint correspondences between consecutive frames are found using the nearest neighbor search method between the ORB feature sets. A special RANSAC method (RAMOSAC) is then applied to the keypoint pairs to find the most appropriate motion model among multiple object transformations (translation, similarity, affine, projektive) and using the information of the previous location. The ORB feature set is updated by adding new features and pruning outliers. The validation of new feature points is done iteratively based on their score, instead of using the information from the different ORB parameters extracted from the actual frame. The authors tested the algorithm in typical foreground-background senarios, where the object to be tracked (e.g. human) was moving in the foreground, which represents quite simple use cases. This solution is also resource intensive, due to the application of RANSAC, which makes its application difficult for tracking multiple objects. In [50] the ORB descriptors are extracted and matched between subsequent frames. Location is updated on matched keypoints and on temporal/spatial constraints, depending on the success of keypoint matches. The comaprison of keypoints is done based on their Hamming distances. For the next frame a search window is defined based on simple motion modeling. Based on this the keypoint extractions and associations are applied. The solution was tested in scenarios with classical fixed camera settings (e.g. surveillance cameras), where the tracked objects (humans, cars) are moving along fixed trajectories with relative moderate and constant speed.

In the following a few important solutions are presented, where the ORB-based feature matching is combined with some additional computer vision techniques (e.g. silhouette detection, object detection), in order to provide tracking and 6D pose refinement in different scenarios (e.g. AR, robotic object manipulations). While these solutions are quite powerful, their application mode doesn't allows to directly apply them in our use case. The BYTE association method is extended in [41] to utilize the solution in heavy-occluded detections for the associations which are usually discarded. Camera motion compensation is applied for tracking by using the ORB feature detector and the RANSAC algorithm. The ORB method is used like in SLAM, as a sparse image registration technique, in that foreground objects like moving persons can be neglected. Also a method to terminate inactive tracks under severe camera motion is proposed which further improves the performance. However, this solution uses the ORB algorithm just for the camera tracking, to compensate for small camera movements, thus the keypoints are not used in the dynamic object's detection, association or tracking. Object detection is provided by the usual CNN-based algorithms, which makes it unapplicable for real-time MR applications. In [24] the authors use a graph-based single-view RGB-D tracking approach for assembly guidance based object tracking in AR environments, which is a markerless approach. Tracking is initialized by YOLOv8Pose, which is followed by a constant pose update. For the tracking between AR assisted assembly steps a graph is defined, with the kinematic links between each assembly pair in different steps. GBOT can dynamically switch between different assembly states, while other graph-based approaches require initialization per state. Each assembly step is defined by the relative pose between the objects. The object detection and tracking is done with the YOLOv8Pose algorithm, which aquires both bounding boxes (BB) and keypoints detected in the object's BB, by using non-maximum suppression and Perspective-n-Point (PnP) recovery of the 6D object poses. Due to the usage of CNN-based object detection and the PnP algorithm the solution requires GPU support, which makes it unfeasible for MR applications. The solution of ICG+ in [43] uses region- and keypoint-based techniques, combined with depth data to perform a 6DoF object pose estimate of a rigid object. For each frame a silhouette is constructed by statistically differentiating the foreground and background. Detected ORB keypoints are are matched with stored keyframe features. Keyframe formation is similar to ORB-SLAM. In case the new frame is considered a keyframe a depth rendering process is initiated, and for points falling on silhouette 3D model points are reconstructed. These are stored as a new keyframe. Then the new object pose is calculated. ICG+ basically proposes an improvement for region-based detection techniques, since all the other information sources are merged and filtered (depth, feature point) using the object's silhouette. However, region-based methods are mainly used for single object and close proximity scenarios, since they can only moderately tolerate scale variations, are prone to background interference and proved to be less sensitve for fast motion. Since ICG+ uses keypoint as a region validation technique it does the keypoint management quite easily: all the keypoints falling on the shilouette can be considered as valid ones for the new keyframe. In our case, since keypoints are the primary focus factor of detection, a careful validation and acceptance criteria have to be set, which involves several filtering and validation steps, depending on their actual quality and the context.

3.2 Semantic Tracking

In the following, several notable semantic tracker solutions are discussed, highlighting that such methods generally rely on 2D or 3D object detection on a frame-by-frame basis. This inherent dependency makes their direct application challenging in mixed reality (MR) scenarios, especially if real-time performance on resource-constrained devices is required.

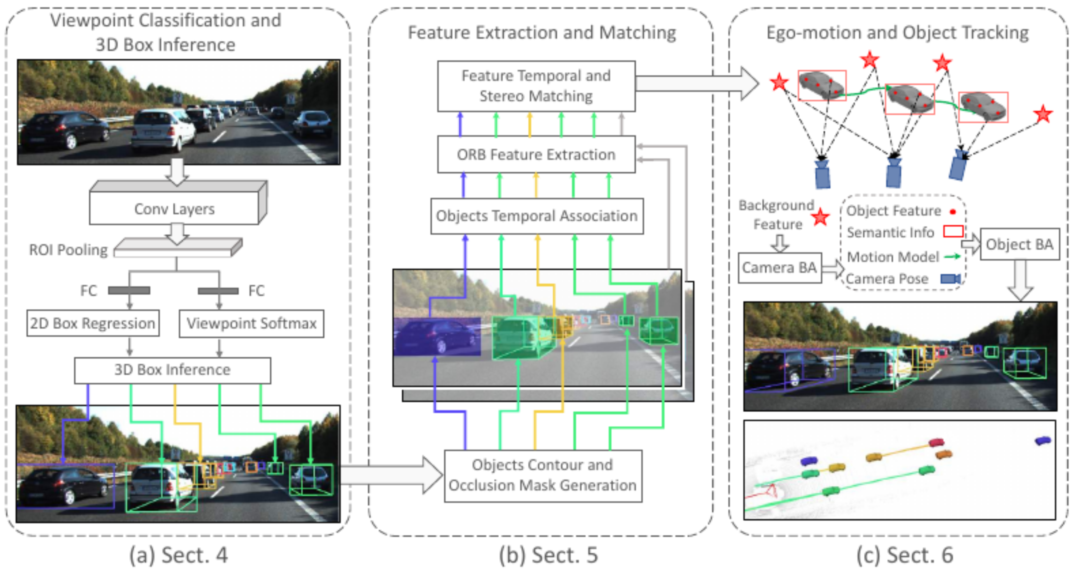

Figure 3.1: Overall architecture of the stereo vision-based semantic tracking system proposed in [23].

A representative solution for autonomous driving scenarios is presented in [23], where both camera ego-motion and 3D semantic object tracking are estimated concurrently. Figure 3.1 shows the pipeline from initial 2D detection through 3D semantic inference and feature matching, culminating in a dynamic object bundle adjustment stage. The system leverages a 2D object detector (as opposed to a 3D detector) to reduce training complexity and computational burden. The semantic inference (3D box with direction) is constructed from a 2D box and a viewpoint based classification. From the object contours and occlusion masks provided by the 3D box inference module ORB feature points are extracted, which are matched using temporal and stereo matching. The temporal matching associates objects for successive frames by a 2D box similarity score voting, while the stereo matching is performed by epipolar line searching. Object association with previously detected objects is achieved by comparing the computed similarity scores, and a RANSAC-based outlier rejection using a local fundamental matrix test ensures robust feature correspondences. Finally, a dynamic object bundle adjustment integrates these feature measurements to continuously track object states alongside the camera pose.

By separating dynamic objects from static scene elements and performing 3D tracking on associated ORB keypoints, the approach provides semantic information regarding object states over time. Its evaluation on the KITTI benchmark dataset shows that the ego-motion estimation is on par with or better than ORB-SLAM2 in certain cases. Object localization and tracking also appear effective, although a comprehensive quantitative analysis of all aspects is not provided.

Despite these advantages, the reliance on a 2D detector (Faster R-CNN) executed on a frame-by-frame basis, combined with the computational cost of the dynamic bundle adjustment, limits the approach's feasibility for real-time MR scenarios. The resource-intensive nature of both the detection and the optimization steps poses challenges for deployment on devices with constrained computational and energy budgets.

A more recent solution, UDOLO [45] stands out from other trackers as its system utilizes dynamic object occupancy maps (OOM) and previous object states (3D bounding boxes, predicted poses) as a spatial-temporal memory to enhance efficiency and accuracy in detecting and tracking objects in various environments. This provides an extended, semantic knowledge of the environment, which improves the new detections focusing the tracker on relevant spaces, by providing 3D occupancy maps of the segmented areas, categorizing them based on different spatial characteristics, such as free, occupied, blocked or potentially traversable zones. By incorporating temporal information directly into the detection pipeline, UDOLO increases stability, reduces runtime, and enhances detection precision over baseline methods. The system is shown in 3.2 The primary data flow is indicated by solid arrows. Notably, the design incorporates a feedback mechanism highlighted by red arrows. At each time increment, the front-end component, referred to as the OOM-Guided RPN, utilizes the point cloud as input. It selectively generates object proposals for the current frame within regions demonstrating high object occupancy scores (depicted by red points) as indicated by the object occupancy map. Additionally, it identifies unobserved regions (illustrated by blue points) where new objects may emerge. These proposals undergo fusion with back-end predictions of object future states from the preceding frame and are subsequently processed through the Fusion R-CNN detector in the second stage. Following tracklet association, the current front-end predictions feed into the Kalman Filter to yield fused object states representing the final bounding box prediction. Subsequently, the motion prediction module updates the object occupancy map based on future object states. The system achieved 32ms inference time minimum on unspecified hardware. This speed, which can be translated into roughly 31 FPS, includes the time for object detection as well. The most contributing factor to achieving such speed is the early integration design in the OOM-Guided Region Proposal Network (RPN). However, its performance metrics are provided for LIDAR input, which typically contains fewer 3D points than stereo- or RGB-D-based depth maps. Furthermore, UDOLO relies heavily on GPU-accelerated CNNs, making it unsuitable for MR devices with limited computational resources. There is no attention paid on point clouds generation methods, how depth images are processed, what processing characteristics they have, and whether there is a need to accelerate the generation of depth information. This leads to a rather heavyweight solution, regarding the energy and computing requirements.

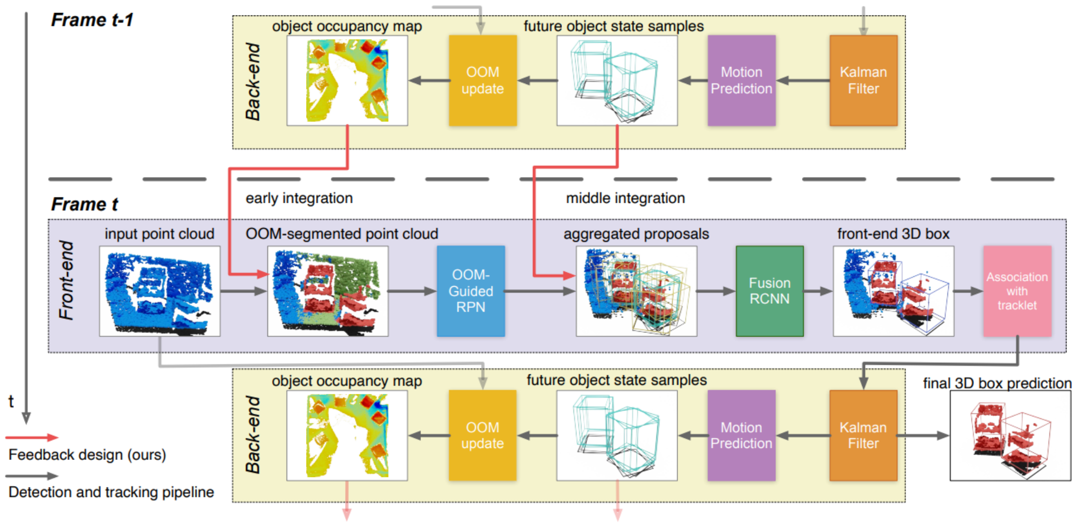

Figure 3.2: UDOLO system.

Similarly, the YOLO object detector family was extended in [40] to offer semantic tracking using point clouds coming from LIDAR and RGB camera streams. By merging 3D CNN-based detections with visually derived semantic segmentations, the system provides object detection and tracking tailored for autonomous driving, tested on the KITTI dataset.

Another approach, proposed in [1], builds on LIDAR-based panoptic segmentation. Here, semantic scene understanding is achieved by categorizing the environment into various classes (e.g., cars, pedestrians, roads, vegetation) and modeling object instances as probability distributions in the 4D spatio-temporal domain. This approach instead of relying on explicit temporal data association, processes point clouds in parallel. The solution was tested on the SemanticKITTI benchmark for LIDAR datasets.

Despite the advances outlined above, these semantic tracking approaches remain largely dependent on heavy computational resources, such as GPUs and CNN-based object detectors, often limiting their applicability in constrained mixed reality (MR) environments. More importantly, many of these solutions lack the flexibility to integrate seamlessly with non-neural detection modules or to adapt to stringent real-time performance requirements on resource-limited hardware. Therefore, it becomes crucial to identify tracking algorithms that are both modular and lightweight, capable of running at high frame rates, and easily integrable with various front-ends and back-end data association schemes. In the following, we present the criteria for an ideal baseline tracker that will serve as the core of our proposed semantically enhanced MR tracking system.

3.3 Ideal Tracker

The ideal tracker to be the cornerstone of the semantically enhanced tracking system should have the following properties:

1. It should have minimal use of neural networks to **enable modification in any part**.
2. It should have a **flexible** enough architecture to join with custom front-ends like object detection or output of another tracker and so on.
3. It should be **modular** as to aid a distributed architecture.
4. It should be able to run with at least **60 Frames Per Second (FPS)** on the target hardware.
5. It should have **clear**, concise and easy to read **implementation**.
6. Would be beneficial if it has an actively managed GitHub[1] page and active community.

3.4 Influential and Innovative Trackers

To identify suitable building blocks for a fast, modular, and easily integrable semantic tracking system, it is helpful to examine the state-of-the-art solutions in both single and multiple object tracking contexts. Although many of these existing trackers incorporate complex neural components or rely on non-real-time hardware, they provide valuable insights into current methodologies, performance benchmarks, and architectural principles. In particular, detection-free solutions and those with open-source communities may offer a promising starting point for developing a flexible tracker tailored to MR constraints. Below, we first consider notable single-object trackers and then proceed to review cutting-edge multi-object trackers, both in 2D and 3D domains.

3.4.1 Single Object Trackers

Although SOT is not the primary focus of this thesis, these trackers are worth mentioning. They represent the current state of the art, and there is potential for their use in the Head Mounted Devide (HMD) for last-minute processing. Notably, all the trackers listed below operate without reliance on detection.

- **OSTrack** [54] stands out from other trackers because it uses a lightweight and flexible framework and does not depend on large training datasets and complex architectures. OSTrack is designed for real-time applications, with the help of its innovative use of attention mechanisms and transformer-based models. Thanks to these features it can focus on important details and maintain high accuracy, even in tough scenarios.

  Its smaller model, OSTrack$_{256}$ is one of the fastest trackers running on GPU in the detection-free category. It can achieve 105 FPS on GOT-10k dataset benchmark with GeForce GTX TITAN X GPU.

- **ARTrackV2** [2] is very similar to OSTrack in some ways, but it further advances the attention mechanisms and enhanced transformer-based architecture. These improvements allow the tracker to capture and focus on relevant features, even in complex environments. Unlike other trackers that need large computational resources and

[1]GitHub is a platform that allows people to share and version control code

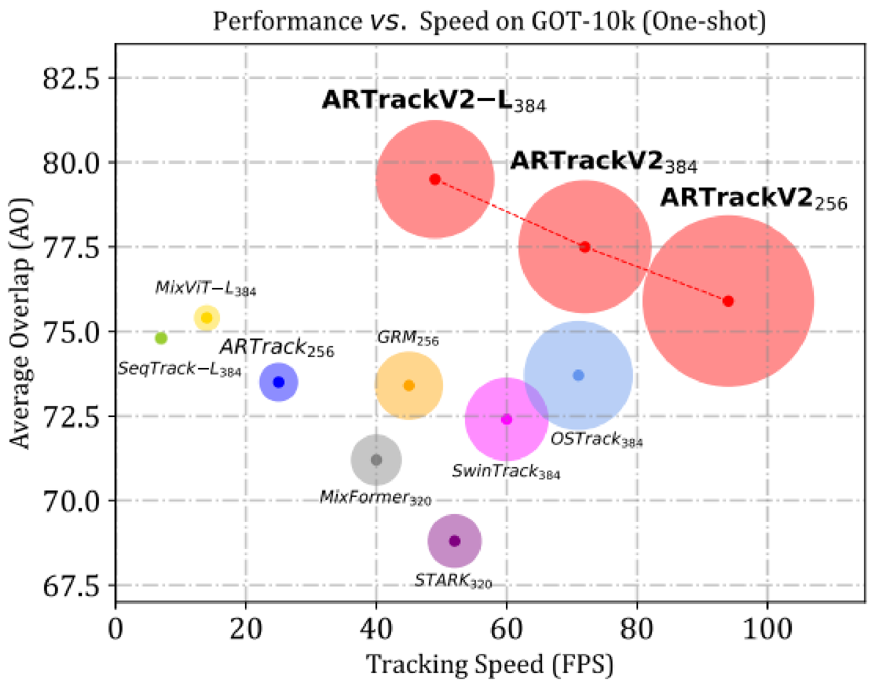

Figure 3.3: Comparison of accuracy and speed of the current top preforming, GPU focused, SOT trackers [2].

struggle with real-time applications, ARTrackV2 is optimized for efficiency without sacrificing accuracy. It tops OSTrack in accuracy even with its fastest and least accurate model ARTrackV2$_{256}$ by 4% (meaning an increase from around 73% to 77%) as shown by Figure 3.3. While being more accurate, ARTrackV2$_{256}$ falls short to the speed of OSTrack$_{256}$ by 11 FPS with achieving 94 FPS on GOT-10k dataset benchmark on GeForce GTX TITAN X GPU.

In Figure 3.4 you can see an example on how well the current State-Of-The-Art (SOTA) detection-free single object tracker responds to object rotation, deformation and size change.

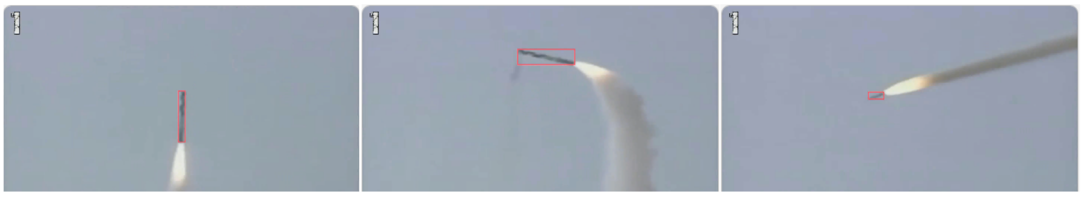

Figure 3.4: ARTrack bounding box resizing [46]

- **STARK** [53] is a currently more that 3 years old tarcker that based on its achievements, had a quite big impact on the field of VOT. STARK won the VOT-21 RGB-D challenge and has 617 stars on its GitHub repository. The novelties STARK sets itself apart with are the spatio-temporal transformer network, end-to-end learning framework and advanced attention mechanisms. Its benchmark performance is no longer the SOTA, but they claim to achieve more than 200 FPS on RTX TITAN GPU with their fast version of STARK called STARK-Lightning.

- **E.T.Track** [6] presents significant advancements in visual tracking by introducing a novel transformer-based architecture that balances high performance with real-time operation with its 2 primary novelties described below:

    1. **Exemplar Attention**: A new attention mechanism designed specifically for visual tracking. Unlike traditional attention mechanisms, Exemplar Attention leverages domain-specific knowledge to capture more explicit information about the target object. It uses a single global query value to identify the object of interest, reducing the computational complexity. This mechanism aggregates information efficiently by utilizing a small set of exemplar values, which act as a shared memory between the samples of the dataset, significantly improving the tracker's performance without incurring a substantial runtime penalty. The Exemplar Attention module replaces conventional convolutional layers in the tracker heads, resulting in a lightweight yet effective transformer layer that enhances the tracker's accuracy and robustness.
    2. **Real-Time Capability**: Despite using a transformer-based approach, E.T.Track operates at 46.8 FPS on a CPU, making it up to 8× faster than other transformer-based trackers. This impressive speed is achieved without compromising much on performance, as E.T.Track consistently performs on par with other methods on several benchmark datasets, including LaSOT, OTB-100, NFS, TrackingNet, and VOT-ST2020.

3.4.2 Multi Object 2D Trackers

While single-object trackers provide essential insights into detection-free methods and the potential of lightweight architectures, multi-object tracking (MOT) introduces additional complexities. MOT systems must handle challenges such as object occlusion, interaction, re-identification, and the maintenance of unique identities across multiple detections. This section focuses on prominent multi-object trackers in the 2D domain, which are highly relevant for applications requiring real-time performance and robust tracking capabilities across dynamic scenes.

These trackers demonstrate diverse approaches, from leveraging traditional motion models and simple data association techniques to employing sophisticated appearance features and advanced optimization strategies. Many of the solutions highlighted also prioritize speed and efficiency, making them suitable for deployment on devices with limited computational resources. Below, we explore several state-of-the-art 2D MOT frameworks, analyzing their methodologies, strengths, and benchmark performances.

- **SORT** [5] is a very influential solution introduced more than 9 years ago. It became the base for many trackers later. The article it was proposed in has been cited in around 4000 other works, and the GitHub repository containing its open source code has 3800 stars.

  In a more technical perspective SORT is a highly efficient and straightforward approach to multiple object tracking by focusing on real-time applications. Its methodology can be summarized in 4 points:

    1. **Detection**: The flexible and simple architecture of SORT can utilize the output of any object detector as long as it is in the the correct data format.
    2. **Estimation Model**: Employs a linear constant velocity model to predict object states (position, scale, aspect ratio) between frames. This prediction is

   updated using the Kalman filter, a classical method for estimating the state of a dynamic system.

3. **Data Association**: Uses the Hungarian algorithm to associate detected objects with existing tracks. The association is based on the IoU metric, which measures the overlap between predicted and detected bounding boxes. A minimum IOU threshold is set to ensure reliable associations. It also handles short-term occlusions implicitly by favoring detections with similar scales, thereby maintaining robust tracking even when objects are temporarily obscured.
4. **Track Management**: As a mandatory finishing step, SORT initializes new tracks for detections that do not match existing tracks, and assigns unique identities to them. Tracks are deleted if they are not detected for a predefined number of frames.

They claim that the tracking component runs at 260 Hz on single core of an Intel i7 2.5GHz machine on the MOT15 dataset benchmark, which is impressive. It easily achieves real-time performance on other datasets from different research papers as well: 60 FPS on MOT16 [49], 143 FPS on MOT17 and 57.3 FPS on MOT20 [12]. However, it is important to note that the time for detection is often excluded from the reported FPS, as is the case here.

- **OC-SORT** [7, 4], or Observation-Centric SORT, is an enhancement of the original SORT algorithm, designed to improve robustness in multi-object tracking, especially under challenging conditions such as occlusion and non-linear motion, where the original is lacking. The main problem with SORT is; when a track does not receive updates for several time steps, errors accumulate during state propagations. These errors can become excessive, causing the track to be incorrectly lost even after an update. This issue arises from the discrepancy between the true track direction and the estimated direction. As the difference between these directions grows, the track may be lost again after re-association.

  OC-SORT addresses these limitations by introducing several key innovations:

  1. **Observation-Centric Re-Update (ORU)**: This method corrects errors that accumulate during occlusions. When an object is re-detected after being occluded, ORU recalibrates the object's state based on the latest observations, rather than relying solely on prior estimates. This significantly reduces the error that builds up during periods when the object is not visible.
  2. **Observation-Centric Momentum (OCM)**: OC-SORT adds a momentum term to the data association process. This term helps maintain consistency in the object's velocity, improving tracking performance, especially during rapid and non-linear movements.
  3. **Enhanced Data Association**: Improves the association of detected objects with existing tracks by incorporating more robust metrics that consider both position and velocity, making the tracking more resilient to occlusions and abrupt changes in motion.

  OC-SORT simply overperforms SORT in every metric as on the MOT17 dataset it almost doubled the achieved MOTA and HOTA scores and reduced the number of identify switches (IDSW) to less than half. Speed wise OC-SORT runs at over 700 FPS on a single CPU (detection excluded), but it is unclear on what dataset exactly. If if was on MOT17, which is likely, then the speed increase is a little less than a 5$\times$ improvement.

- **SFSORT** [30] is designed to be the fastest multi object tracking system. It employs a tracking-by-detection method following the prior SORT literature. It also introduces a novel descriptor, the Bounding Box Similarity Index (BBSI) that enhances object tracking by considering the shape similarity, distance, and overlapping area of bounding boxes. BBSI efficiently manages both overlapping and non-overlapping bounding boxes without the need for motion prediction tools like the Kalman Filter. Among its advantages are:
  - reducing dependency on motion prediction, leading to faster tracking, and
  - effective handling of non-overlapping bounding boxes.

  Another key point of SFSORT is, that it incorporates scene features such as depth and camera motion to improve object-track association and track post-processing, which results in more accurate tracking. They call these components:
  - **Camera Motion Detection**: Utilizes ORB features to detect camera motion, adjusting interpolation timeouts according to whether the camera is moving or stationary.
  - **Scene Depth Estimation**: Introduces a depth score metric that evaluates the variation in object heights within the scene, aiding in the adjustment of tracking parameters for scenes with significant depth.

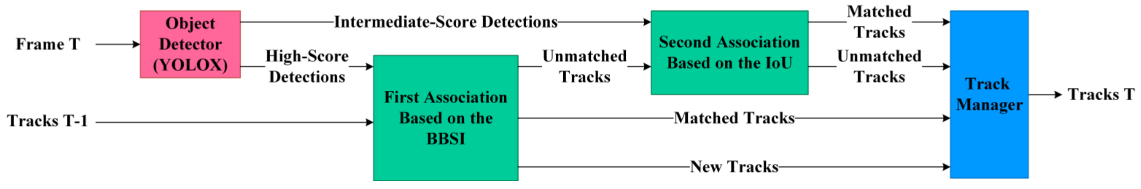

Figure 3.5: SFSORT pipeline [30].

  In Figure 3.5 the SFSORT system is shown. The last component on the right, labeled "Track Manager", has the final innovation of the tracker. As usual, it manages matched, new, and unmatched tracks, updating their status and handling lost tracks, but it does so based on their last observed location (central or marginal). So SFSORT differentiates between tracks lost at the margins and those lost in the central areas of the frame. It applies different timeouts based on this information hence the likelihood of successfully revisiting lost tracks increases.

  SFSORT delivers on the set goal of being the fastest tracker without compromising too much on other metrics. Operating on a 2.2 GHz Intel Xeon CPU it achieves a HOTA of 61.7% with a processing speed of 2242 Hz on the MOT17 dataset and 60.9% with a processing speed of 304 Hz on the MOT20 dataset.

- **UCMCTrack** [55] is a tracking-by-detection based MOT system designed to manage significant camera movements efficiently that uses motion model only. It introduces a uniform Camera Motion Compensation (CMC) approach that uses the same compensation parameters consistently across a video sequence, rather than recalculating them for each frame. UCMCTrack employs a Kalman filter on the ground plane and utilizes the Mapped Mahalanobis Distance (MMD) for data association.

  The mentioned Mapped Mahalanobis Distance replaces the Intersection over Union (IoU) distance measure and uses projected probability distributions on the ground plane. This method effectively handles uncertainties introduced by homography projections and improves tracking accuracy. Utilizing the Kalman filter on the ground plane helps in mitigating the effects of camera movement by treating it as noise within the motion model.

Regarding performance, UCMCTrack was benchmarked on MOT17, MOT20, DanceTrack and KITTI datasets and when provided with detections, it can operate at more than 1000 FPS. The exact hardware is not specified. The UCMCTrack+, which is the enhancement of UCMCTrack with the additional incorporation of CMC, achieves scores very close to the SOTA on every datase with the exception of MOT20.

- **TOPIC** [8], or the Two rOund Parallel matchIng meChanism is a multi object tracking framework, that utilizes both motion and appearance features in parallel, enhancing tracking performance. Additionally, it uses an Attention-based Appearance Reconstruction Module (AARM) to improve appearance feature representation. AARM enhances appearance feature embeddings by reconstructing them using an attention mechanism, improving the distinction between different objects and the similarity for the same object across frames.

  The main reason for mentioning this tracker is its parallel association modul, but it also performs slightly better on average than OC-SORT on both MOT17 (as shown in Table 3.1) and MOT20.

- **EMO** [15] is not really a tracker by itself, more like a set of optimization strategies designed to enhance the efficiency and performance of existing multi-object tracking systems when deployed on resource-constrained edge devices. It can be integrated with various existing trackers to improve their computational efficiency without significantly compromising tracking accuracy.

  EMO achieves this by frame skipping using one of the 2 following strategies:

  - **Motion Aware Periodic Skipping**: A simple approach where detections are skipped at regular intervals (e.g., every second, third, or fourth frame). This method reduces the number of detection operations but may miss new objects or significant movements.
  - **Context Aware Skipping**: A more advanced approach that uses the context of the video, such as motion patterns and frame similarity, to decide when to skip detections. This method aims to balance computational savings with tracking accuracy.

Tracker	**HOTA ↑**	**MOTA ↑**	**IDF1 ↑**	**IDSW ↓**	**FPS ↑**
SORT	34.0	43.1	39.8	4,852	143.3
OC-SORT	63.2	78.0	77.5	1,950	*700*
SFSORT	61.7	78.8	74.4	3,264	**2241.8**
UCMCTrack+	**65.8**	**80.5**	**81.1**	1,689	**157.1**
TOPIC	**63.9**	**78.8**	78.7	**1,515**	-
EMO (+FairMot)	63.0	71.3	**79.4**	**196**	*<100*

Table 3.1: Benchmark results on MOT17 dataset. The best result is noted with the color **red**, the second best result with color **blue** and what is uncertain if correct is *tilted* [49] [7] [30] [55] [8] [15] [29]

These were the most unique or promising 2D multi object trackers that i have overviewed. The discussed trackers benchmark details on the MOT17 dataset are shown in Table 3.1. It can be seen that UCMCTrack+ dominates the MOT17 benchmark, but with the exception of SORT, there is not a very big difference in performance. UCMCTrack is noted as the second fastest tracker not by mistake, but because on the MOT17 benchmark results i could not confirm the claimed 700 FPS of OC-SORT, rather found multiple versions with varying values ranging from 28-67.

3.4.3 Multiple Object 3D Trackers

Building on the principles and challenges of 2D multi-object tracking, 3D tracking extends these systems into spatial domains where depth information is critical. This capability is essential for applications such as autonomous driving, robotics, and augmented reality, where understanding the position and movement of objects in three dimensions allows for more sophisticated and context-aware interactions. Unlike their 2D counterparts, 3D trackers must account for the complexities of depth estimation, rotational variance, and sensor-specific noise, such as that from LiDAR or RGB-D cameras.

This section explores state-of-the-art solutions in 3D multi-object tracking, highlighting methodologies designed for real-time performance and robust operation in dynamic environments. These trackers demonstrate a range of approaches, from filter-based methods that emphasize computational efficiency to learning-free frameworks optimized for rotational alignment and scalability. Below, I examine key 3D trackers, emphasizing their pipelines, innovations, and benchmark results to identify trends and opportunities in this rapidly evolving domain.

- **AB3DMOT** [48] is a simple yet effective 3D multi-object tracking (MOT) system designed for real-time applications such as autonomous driving. It leverages a combination of a 3D Kalman filter and the Hungarian algorithm for state estimation and data association. The system pipeline as shown in figure 3.6 includes: (A) obtaining 3D detections from LiDAR point cloud using off-the-shelf 3D object detector, (B) predicting the state of associated trajectories using a 3D Kalman filter, (C) matching predicted trajectories and current frame detections with Hungarian algorithm, (D) updating matched trajectories with the 3D Kalman filter, and (E) creating/deleting trajectories for new/disappeared objects. AB3DMOT does not require training and can be directly used for inference.

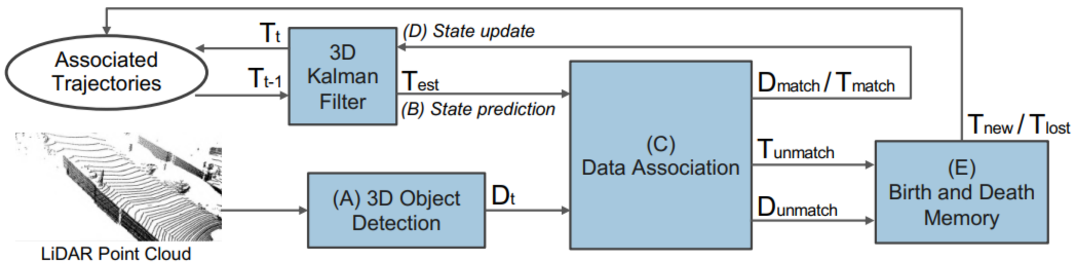

Figure 3.6: AB3DMOT proposed pipeline.

With the output of an object detector provided, AB3DMOT achieves 207 FPS on the KITTI dataset for cars, the highest among 3D trackers, and even higher for pedestrians and cyclists. This speed was achieved on a CPU, though the exact hardware specifications were not disclosed.

- **Fast-Poly** [26] is a learning-free method for 3D multi-object tracking that utilizes filter-based techniques to enhance accuracy and speed. It addresses object rotational anisotropy, enhances local computation densification, and leverages parallelization techniques to improve inference speed and precision. The method employs two independent computing processes to filter 3D detections and predict existing trajectories at each frame, utilizing SF (Score Filter) and NMS (Non-Maximum Suppression) filters for processing detections and predicting motion, score, and time-invariant states of trajectories. Key features contributing to Fast-Poly outperforming other methods include alignment to counter object rotation, densification to increase computational efficiency, and parallelization to alleviate serial bottlenecks. Additionally, Fast-Poly demonstrates robustness in device migration and parallel efficiency as the number of objects increases, showcasing outstanding computational efficiency in autonomous driving scenarios. It achieves high performance on nuScenes and Waymo datasets, outperforming other advanced methods in terms of speed and accuracy. Concretely it can run with 34.2 and 35.5 FPS on nuScenes and Waymo datasets on a personal CPU. However, the inference time for object detection is not included in the reported FPS values.

Its pipeline is shown in figure 3.7. Notable enhancements to the baseline method (which was called Poly-MOT [25]) are visually distinguished. Specifically, alignment adjustments to counter object rotation are represented in orange, densification strategies aimed at augmenting computational efficiency are highlighted in blue, while cyan indicates parallelization methods employed to alleviate serial bottlenecks.

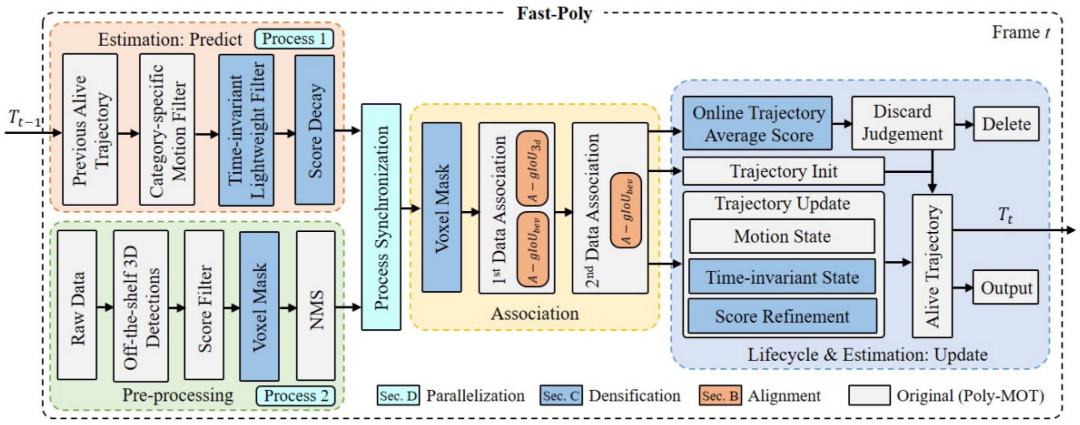

Figure 3.7: Fast-Poly pipeline [26].

All three 3D trackers reviewed perform well on their respective datasets, but they were not benchmarked on the same datasets, making direct comparison difficult. However, Fast-Poly, being the most recent method, demonstrates a significant lead on its benchmarked datasets, establishing itself as the state-of-the-art in 3D multi-object tracking.

3.5 Chosen Approach

Ultimately, none of the mentioned trackers provide the exact end-to-end utility we were searching for, thus I decided to design and go with a custom solution, which uses the inner workings of ORB feature points and maintaines the object related keypoint sets initially identified by an object detector.

To achieve this I came up with a theory how to mitigate the drawbacks of the feature points based tracking, and concluded a research to back it up.

Chapter 4

Proposed Method

This chapter explains the proposed method, how it works, and the reasons behind its design. It also discusses how the solution addresses the issues identified in existing approaches.

4.1 Core Concept of Semantic Keypoint Manangement

As I previously pointed out current MOT solutions are not able to cope with the requirements of XR scenarios, mainly because their frame-by-frame OD dependency. Semantic information in MOT is also introduced through OD solutions, with the main goal to distinguish between different object categories (e.g. cars, pedestrians) or between dynamic and static object types (e.g. tracked cars vs. road elements, buildings, vegetation).

Based on these findings I categorize the use of object detectors in MOT as follows:

- **Tracking-by-detection methods:** The tracked objects (usually focusing on specific object categories, e.g. people) are found frame-by-frame in the video stream by the OD, the outcome of this process are the 2D/3D bounding boxes built (BB) around the respective objects. The object categories, semantic information is not considered in the tracking process beyond this point. After this the tracker associates the BBs found with the managed objects, based on the previous tracker state (object poses, motion model) and the BB parameters (size, coordinates).
- **Semantic tracking methods:** The categories of detected objects are used as semantic information to distinguish object types in the tracker back-end. This information is usually used in the association process, like in [40] [1], while there are also solutions [23], where the semantic BBs are used to extract category related keypoints, which can belong to static or dynamic objects. Then these semantic keypoint categories are used to provide information for the tracking and camera ego-motion modeling tasks.

Albeit all these trackers [40][1][23] are using semantic information in some form, they are also using OD (as the source of semantic information) on a frame-by-frame basis. This means that these trackers would also fail in XR scenarios.

To overcome the above shortcomings I devised a tracking solution where the OD is used only at specific points of the system's life-cycle (initialization step, fallback for severe use cases: e.g. objects completely lost). In our case the outcome of the OD (the object's

2D BB) is also used to extract semantic information in form of keypoints (as in [23]), but after this the keypoint populations are maintained by other, more lightweight mechanisms. Thus, by providing an initial semantic separation of keypoints (groupings extracted from object instances) and the possibility to validate and update the related keypoint sets on each iteration (incoming frame) I avoid to use the OD on a frame-by-frame basis.

To provide the required semantic information maintenance I must employ several mechanisms, like the separation of detected keypoint types (whether they are extracted from static or dynamic objects), the association of the dynamic keypoint sets with the tracked objects, respectively the update of the object related keypoint database, which will be used in the next step as reference dataset. The solution and exact mechanisms are presented in the following.

4.2 Architecture

Figure 4.1 presents the architecture of nearly the entire system. I contributed to its development as part of a collaborative effort, though it is far from being solely my work. Components that are less relevant to the primary focus of this thesis have been excluded; however, the architecture still remains too large to explain every element in detail. This section will focus on elaborating the most relevant components that are closely related to the main topic of this thesis.

- **Feature matcher:** The primary focus of this thesis. It is a feature point based associator, filterer and validator, providing tailored keypoint sets for other components. It processes the incoming camera frames based on the received Region of Interests (RoIs) and the stored object keypoint sets (KPs).
- **Voxar:** The main controller component, it serves as an mediator between components to ensure keypoints and depth related information for other components.
- **RoI-based Depth:** Creates a depth map of the received frame or frame part, which is used by the Voxel Matcher in cases when the detected and matched keypoints aren't enough for object identification.
- **Displacement Model:** Receives the device (headset) pose, and its camera properties. From these and the previous location and trajectory of the tracked objects, it creates a prediction in the form of a RoI and a list of visible faces of the object. These RoIs are then used both by the Feature Matcher and the Voxel Matcher.
- **Voxel Matcher:** Its main role is to provide a 6DoF pose for the tracked objects and in the meantime to update the object's keypoint set with their spatially validated coordinates. It is composed from three sub-components, each playing a distinct role in processing of dynamic objects:
  - The **Sprite detector** identifies voxelized dynamic objects in the environment by distinguishing between dynamic, static, and noise voxels. Object voxel shells are formed along the visible sides of objects and are searched based on their predicted displacement to enhance detection accuracy. Ray casting is utilized to locate object voxels effectively, while the search space is progressively reduced by focusing on the identified sides of the objects, optimizing both performance and computational efficiency.

– **3D Refin** determines the exact orientation of the object using point cloud registration algorithms and similar techniques.

– The **Associator** ensures the continuity of object identities. Its role is akin to that in methods like SORT, employing algorithms such as the Hungarian algorithm to associate data between frames and maintain consistent tracking.

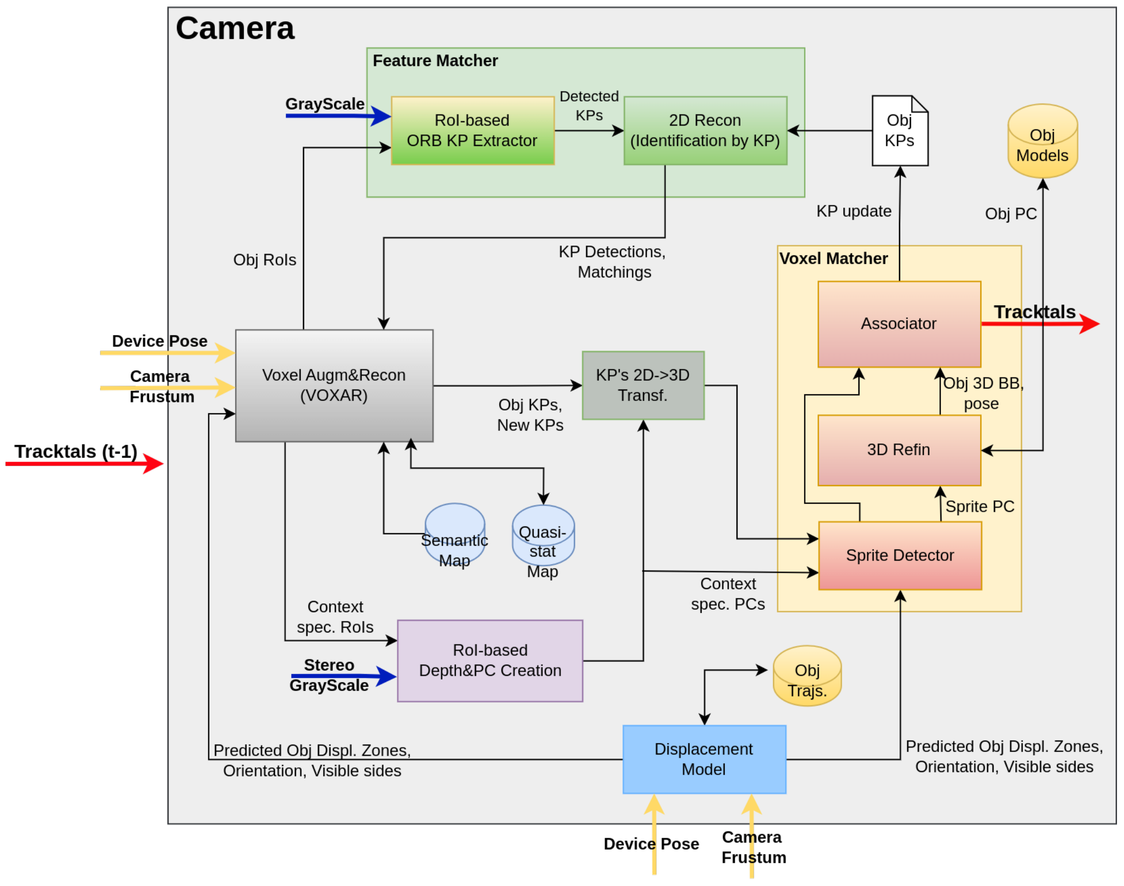

Figure 4.1: Architecture of the system the proposed method is a core component of.

4.2.1 Feature Matcher

The Feature Matcher is used for the semantic information based keypoint management in the tracker, by providing the detection, filtering and primary association steps, while the final phase of the keypoint set forming (spatial filtering and object model based association) is done by the Voxel Matcher. This way the information derived from the 2D and 3D image data is decoupled, which leads to a more robust and efficient system architecture.

For further understanding, Figure 4.2 provides a detailed view of the architecture of the Feature Matcher component. All non-gray elements in the figure represent the main contribution of this work. It should be noted that this architecture is meant to form a detachable module from the Feature Matcher, facilitating the development and testing phases, while the final tracker will look like the one in Figure 4.1.

- The **Manager** subcomponent, represented in blue, initiates the entire process. The gray camera and phone icons indicate real devices that stream data to their respective processing components. In the final system, the phone will not be present; it currently simulates pose data required by the Displacement model, which in production will be obtained from the tracker's output. This data (tracked object's pose) is solely used by the Displacement Model to predict the RoI's position and size for the current frame. Physically the phone is attached to the tracked object (LEGO toycar) to provide accurate position and orientation data, using a SLAM algorithm.

The camera streams frames via a cable to a script running on a computer. This script places the frames and their timestamps into a ring buffer to ensure no frames are lost for consumers with different rates. The Feature Matcher processes each frame sequentially, retrieving the next one only after completing its current processing task.

The Controller subcomponent orchestrates and supervises the overall functionality, ensuring seamless operation of the various components.

- **Pybind11:** The Displacement model, written in C++ by a colleague, was ported to Python using Pybind11. Its functionality remained unchanged: using incoming pose data, it predicts the visible faces of the object and estimates its future pose after a predefined time interval (for the next frame's arrival). These predictions are stored in a ring buffer, from where the Feature Matcher retrieves them as needed.

- **Feature Matcher:** The Feature Matcher starts its cycle with ORB feature detection in every case. It retrieves a frame and applies ORB feature point detection to identify keypoints (KPs). If critical information required for the main loop is missing, this step (Main function) is excluded, and the process continues with object segmentation, which provides the missing semantic information by adding the keypoints of the frame and the detected object to the KPDB. Ideally, this only needs to be executed once, at the start of the algorithm. If no semantic data is missing, the main process proceeds as follows:

    1. **Keypoint matching:** The detected keypoints are matched with those from the previous frame.
    2. **Get RoIs:** The algorithm retrieves a prediction from the Displacement model in the form of a Region of Interest (RoI) corresponding to frame **t**'s timestamp.
    3. **Filter by RoI:** If the RoI is successfully acquired, keypoints outside the RoI are filtered out, as they are not relevant to the objects being tracked.
    4. **Keypoint Classification:** One of the main novelties of this thesis is the classification of keypoints into those belonging to static and dynamic objects. Details on this process are discussed in Section 5.1.
    5. **Identification (IDing):** The remaining keypoints are assigned to specific objects or are discarded if they are not relevant. A successful IDing operation ensures that enough keypoints remain for the voxel matcher to proceed with the final step, the spatial keypoint mapping.
    6. **Voxel Matching:** The voxel matcher uses the retained keypoints, along with techniques like point cloud registration, to estimate the precise position and orientation of the object. This information is used to update a custom database with the ORB feature points of the object (parameters, 3D constellation, associations to object sides, etc.)

Additionally, static keypoints from the current frame are also saved to facilitate filtering in subsequent processing steps. If the RoI cannot be acquired, the algorithm bypasses the intermediate steps and directly saves the keypoints from the frame into the database. Similarly, if the identification process fails, an object detection algorithm is triggered, and its results are stored in the database for future use.

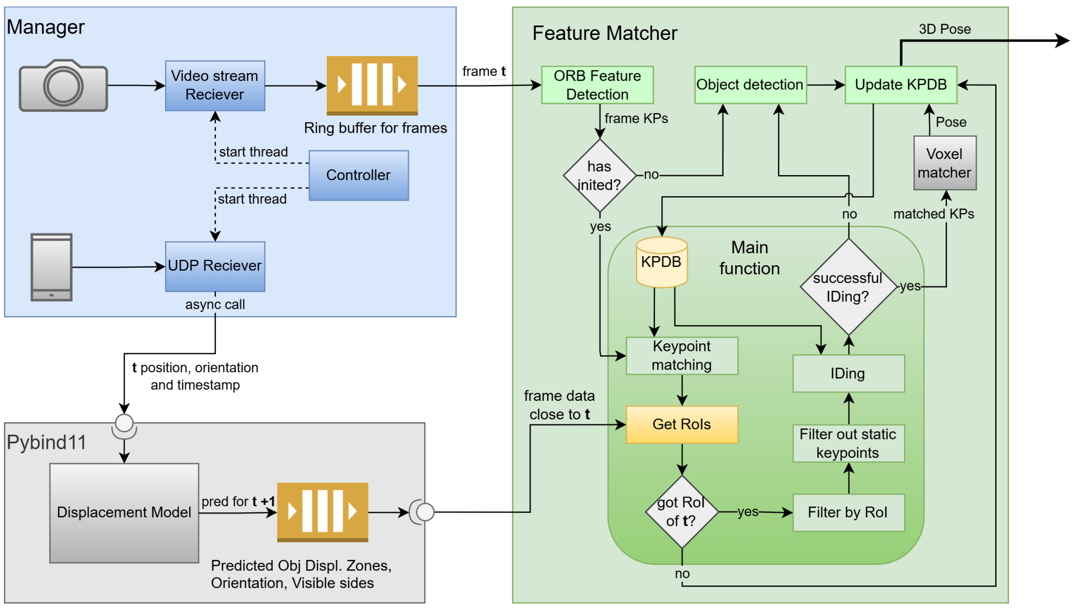

Figure 4.2: Architecture of the system the proposed method is a core component of.

4.3 Main Modes of the Algorithm

1. **Initialization:** The initial state, assuming static objects, where an object detector (OD) is run on the objects within the camera's field of view and we wait for its result (without significant movement or camera displacement). Unlike continuous operation, there is no previous state or keypoint dataset to match the currently detected keypoints against.

   - Extracting keypoints and descriptors from the initial image(s) from the object bounding box (BB) as defined by the OD, which initializes the system. These keypoints will represent the semantic information in the system, since a clear distinction between object exemplars, different categories and other static static elements can be formed as an outcome of this step.
   - Transferring the respective keypoints and descriptors, associated with the objects of interest, to the other system components (for keypoint database update and initial pose refinement).

2. **Continuous (frame-by-frame) operation:** this is the normal operation mode of the system, when the object tracking task is supported by the Feature Matcher output.

   (a) Extracting keypoints and descriptors from the current frame, based on the RoI determined by the prediction component (Displacement Model).

   (b) Matching, filtering and validating the stored (from previous frames and from KPDB) and the currently extracted keypoint descriptors, in order to identify dynamic objects and to associate keypoints with the tracked objects. This step maintains the semantic keypoint set, which enables to find and associate the proper objects with the managed tracklets.

(c) Transmitting the matching results and new potential keypoints to other components, where the keypoints and their descriptor dataset are managed. At the end of this step the keypoint database (KPDB) will be also updated with the spatially filtered and visible side associated keypoints.

(d) Handling special cases (e.g., when no keypoints are found in the given RoI):
- Potentially expanding RoI zones (if no matchable keypoints are found and there is no occlusion).
- Signalize potential occlusion to components.
- Indicate the omission of depth image creation for a given object, based on keypoint matching results.

4.4 2D and 3D Keypoint Characteristics

The following section presents the different operational levels and steps of the Feature Matcher, aimed to address the problems and requirements described earlier. During operation, we progress from basic information processing to more complex interpretations, thus building an expanding a contextual knowledge that accelerates computationally intensive operations.

1. **Keypoint Detection and Pre-processing**:

   During the pre-processing and filtering of keypoints (KPs) detected in the current camera frame, we rank and pre-filter them based on their quality. The goal of pre-processing is:

   - Determining the characteristics of keypoints and assessing their quality:
     – Based on ORB indicators (orientation/angle, response, scale/octave, size).
     – Based on their 2D spatial characteristics.
   - Ranking and pre-filtering keypoints, such as deprioritizing low-quality keypoints to reduce runtime during the matching process, if needed.

2. **Keypoint Matching and Validation**

   The keypoint matching process is based on comparing ORB descriptors, which results in a ranked list based on Hamming distances. The comparison occurs between two sets of keypoints: those detected in the current frame and those saved in the keypoint database of the object.

   Reducing the set of keypoints enhances the efficiency of the matching process with a larger number of KPs and provides an opportunity to interpret the ranking within the matching list. The goal of validation is:

   - Interpretation and validation of the keypoint matching list, for example, considering the ORB ranking in subsequent steps.
   - Filtering the keypoint set for higher-quality analyses and faster execution.
   - Updating the keypoint database and determining the position of the object if no significant changes have occurred.

The objective is to determine at this stage whether further 3D analysis is required or if the keypoint set associated with the object and the 6D position can be updated appropriately.

3. **Analysis of Matching List**

   The result of keypoint matching and validation can lead in two directions:

   (a) If keypoint matching is consistent and of high quality, the object's 3D position can be estimated, and the keypoint database can be updated without requiring further detailed analysis.

   (b) If the matching result is unsatisfactory or not convincing enough, a 3D analysis of the keypoints may be necessary.

   During validation, other indicators and information can be considered alongside the ORB ranking, such as:

   - Tessellation information related to the object model.
   - Analysis of object sides and positions determined by the camera model.

4.5 Keypoint Management

Knowing the object's 6D pose and visible sides helps to intelligently manage the keypoint database, the spatial filtering of the matched and newly identified members of the keypoint set, respectively the association of keypoints with the visible object sides. The outcome of this also accelerates of the next matching process, by focusing the processing only on the subset of keypoints assigned to the actually visible sides of the object. This allows the currently detected keypoints to be evaluated based on context, such as examining the disappearance, orientation, or emergence of new keypoints. Also helps to assess the potential displacement or occlusion of the object.

The set of keypoints placed on the 3D object is categorized and stored per side in previous iterations, enabling efficient searching and analysis. The sides of the object model are divided into segments (e.g., Side A, B further subdivided into segments I, II, III), as illustrated in Figure 4.3. This allows for simple spatial categorization of keypoints without requiring complex geospatial tools.

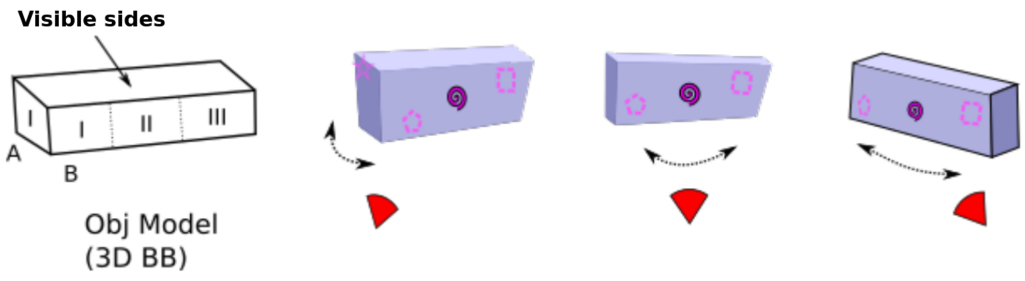

Figure 4.3: Visible sides with regions and an example of a point drifting out of the camera field of view.

The knowledge of side lengths and the associated keypoints aids in analyzing keypoint stability and preparing for the disappearance of keypoints. It is worth combining this information with the filtering and spatial characteristics of keypoints and other relevant data for more efficient analysis.

Chapter 5

Implementation

This chapter delves into the implementation details of the proposed solution, outlining the key components and processes that drive its functionality. The feature matcher, central to the tracking algorithm, is explained in depth, including its handling of dynamic-static keypoint separation, the use of RoIs, and the integration of keypoints into the Key Point Database (KPDB). Additionally, the chapter explores the steps of the tracker, from initial detection to the identification process, and discusses the challenges of tracking multiple dynamic objects simultaneously. The implementation builds upon the theoretical framework presented earlier, with a focus on real-world applicability, efficiency, and adaptability.

5.1 Dynamic-Static ORB Keypoint Separation

Classifying whether a given pixel belongs to the static background or a moving object without relying on an object detector can be achieved by analyzing the intrinsic values of ORB feature points. Specifically, the internal ORB characteristics change differently between matched keypoints depending on whether the points belong to the static background or the dynamic object. When the camera remains relatively stable and the object moves (i.e., it is dynamic), the keypoints on the object exhibit greater changes compared to those on the static background. This behavior is supported by the results presented later in Section 6.3.

Before delving into these figures, consider Figure 5.1, which illustrates two clearly distinct sets of matches. Matches where the keypoints and their connecting lines are red represent dynamic keypoints, assuming the object has moved to some extent (in this example, the lower image is 10 cm further away from the camera than the upper image). In contrast, matches represented by green points and lines indicate keypoints that remained unchanged between the two images and are thus considered static.

One of the primary objectives of this research is to demonstrate that this distinction between static and dynamic keypoints can be achieved using the intrinsic values of ORB feature points.

The measurements are detailed in the following list, but some clarification on their organization is necessary. While the ideal and contaminated scenarios are explained within the list, the concepts of set-based and matched-point-based approaches require further elaboration.

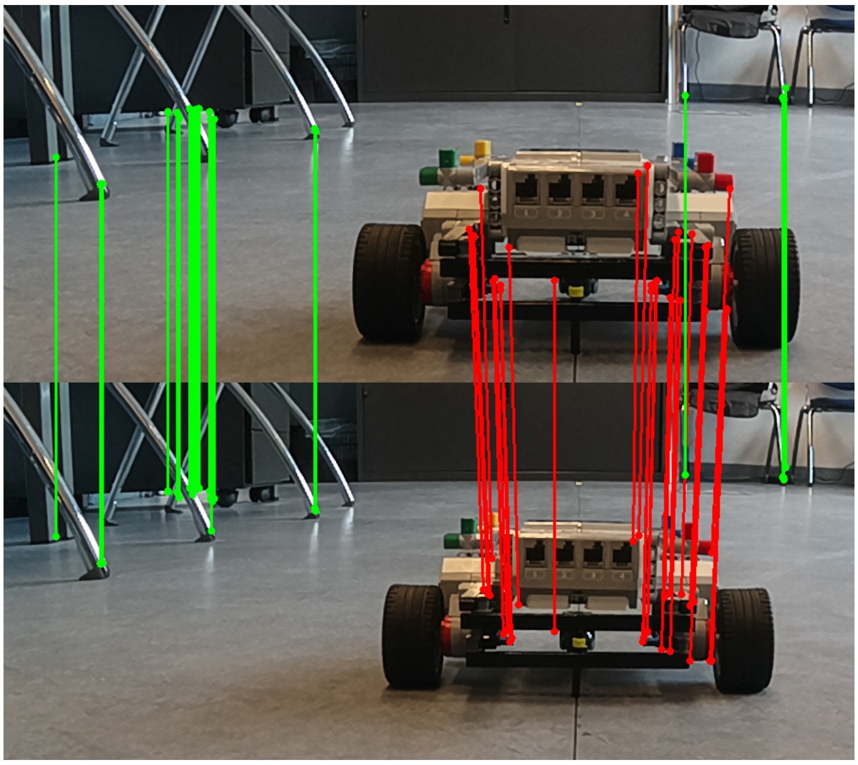

Figure 5.1: Illustration of dynamic (red) and static points (green)

Set-Based Approach: This approach treats keypoints as sets, evaluating a specific property of all points in the set, computing the average value of that property, and then comparing the averages across sets.

Matched-Point-Based Approach: This approach calculates differences in specific properties (e.g., angle) between matched keypoint pairs. The differences for all pairs are then averaged to provide a final result.

List Explaining the Ideal and Contaminated Scenarios

1. **Ideal Scenario:** This scenario represents an idealized condition where noise is entirely absent, and comparisons are made with perfect precision. Here, we utilize masks from both images, ensuring that points are clearly and exclusively categorized into one of two sets: internal (belonging to the object) or external (belonging to the background).

   - **Set-Based:** ORB keypoint detection is applied to both images, and each image's keypoints are split into internal and external sets using segmentation masks. For the internal keypoints in the first frame, average values are calculated for each ORB characteristic (angle, response, size, and octave, as described in 2.3). These averages are then compared with the averages of internal keypoints in the second frame.
   - **Matched-Point-Based:** Descriptors from ORB keypoint detection in both images are matched. The matches are split into two groups: internal matches, where both keypoints lie within the mask in their respective images, and external matches, where both keypoints lie outside the mask. Differences in ORB

characteristics between matched points are calculated for each group, and the averages and standard deviations of these differences are computed.

2. **Contaminated Environment Scenario:** In a realistic scenario, perfect masks are not available, leading to potential contamination of keypoint sets. For example, a keypoint on the toy car may be incorrectly matched with a point in the background. This contamination is expected to impact the results, particularly in terms of increased variance and altered averages. Notably, the internal (object) set remains uncontaminated in this case, as the initial mask ensures no extraneous keypoints are included. However, the external (environment) set becomes contaminated with points from the internal set.

   - **Set-Based:** In the first image, keypoints are separated using the segmentation mask as before. For the second image, internal keypoints are selected based on matches with internal keypoints from the first image. Unmatched keypoints are classified as external, resulting in contamination of the environment set. Then, the average values are calculated as in the previous set-based case.
   - **Matched-Point-Based:** Using the same internal and external sets as in the contaminated set-based approach, the differences in ORB characteristics are calculated for all matched keypoint pairs within each group. Unlike the set-based approach, where differences between averages are computed, here the averages of individual (pair) differences are used.

The graphs illustrating these results have the 5 measurement scenarios on the horizontal axis, each encompassing nine image pairs in very similar situations. These scenarios specifically involve:

- **Small Distance Change:** Changing the object distance from 50 cm to 52 cm with different orientations.
- **Medium Distance Change:** Changing the object distance from 50 cm to 60 cm with different orientations.
- **Object Rotation at Close Range:** Involves a 10-degree rotation at object distances of 50, 52, and 54 cm.
- **Object Rotation at Medium Distance:** Involves a 10-degree rotation at object distances of 152, 156, and 160 cm.
- **Object Rotation at Long Distance:** Involves a 10-degree rotation at object distances of 318 and 324 cm.

5.2 Feature Matcher Process

This section addresses the additional differences in the feature matcher compared to the idealized process shown in Figure 4.2. While the steps may be familiar from the previous architectural descriptions, Figure 5.2 highlights key differences. The steps are outlined as follows:

- **ORB Detection:** ORB feature point detection is performed on the entire frame. This step also includes the matching of points from the Key Point Database (KPDB).

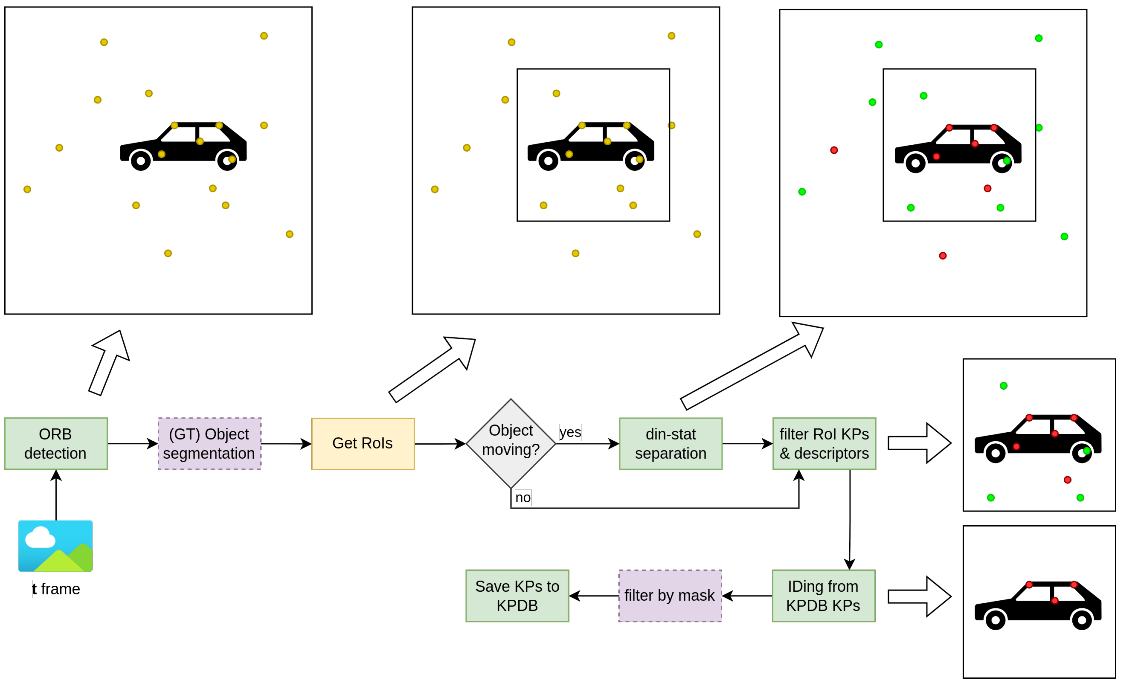

Figure 5.2: Operation of the implemented feature matcher.

- **(GT) Object segmentation:** Object segmentation is executed for every frame, even before the step it is intended to simulate (discussed further below in 'Filter by mask'). This simplifies the implementation by reducing variability and also enables evaluation using ground truth data. In the final solution, object segmentation or detection will not be part of the main loop.
- **Get RoI-s:** Acquire predicted RoI from the Displacement model.
- **Object moving?:** This conditional step determines whether the object is likely moving by comparing the predicted RoI with the RoI obtained from object segmentation. If the distance between the centers of the two RoIs exceeds 2 pixels (a heuristic "magic number" currently used for evaluation), the object is classified as dynamic, and the din-stat point separation is triggered. In the finalized algorithm, this information will be derived from more robust sources. For now, object segmentation is used as a placeholder to simulate the intended functionality.
- **din-stat separation:** The classification of keypoints as static (green) or dynamic (red) based on their intrinsic properties. This step gets skipped if the object is static, since currently that case is not handled properly in the classification algorithm.
- **Filter RoI KPs & descriptors:** Filter the remaining dynamic points further (or all the points if the din-stat was skipped) by removing KPs that are outside of the predicted RoI.
- **IDing:** The KPs of the KPDB that belong to the object get matched to the remaining KPs further reducing the number of KPs while increasing their quality.
- **Filter by mask:** This step is currently substitutes the functionality of the voxel matcher component 4.2. Using the mask gained from running the object segmentation we select new KPs from the current frame to put into the KPDB

- **Save KPs to KPDB:** Save the current frames all KPs and the objects new KPs into the KPDB.

5.3 Steps of tracker

Figure 5.3 illustrates the steps the tracker goes through under ideal conditions. The images are numbered sequentially to reflect their order of execution.

1. **Initial detection:** The first image displays all the keypoints detected in the frame, visualized as red points. Additionally, the ground truth RoI is shown, but this is included solely for evaluation purposes.
2. **Din-stat separation:** The second image shows the state of the keypoints after the dynamic-static (din-stat) separation. Most static keypoints have been filtered out, although some remain, and a few object keypoints have also been lost.
3. **RoI-based filtering:** In the third image, keypoints are filtered based on the predicted RoI. This step removes distant keypoints that were not eliminated during the din-stat separation.
4. **Identification process:** The final step involves matching the remaining keypoints to the object's keypoints stored in the KPDB. Once this step is complete, the presumed center of the object is calculated from the surviving keypoints. Subsequently, new keypoints are added to the KPDB while the previous ones are removed.

In future implementations, the addition and removal of keypoints in the KPDB will be managed by a more advanced algorithm to improve performance and accuracy.

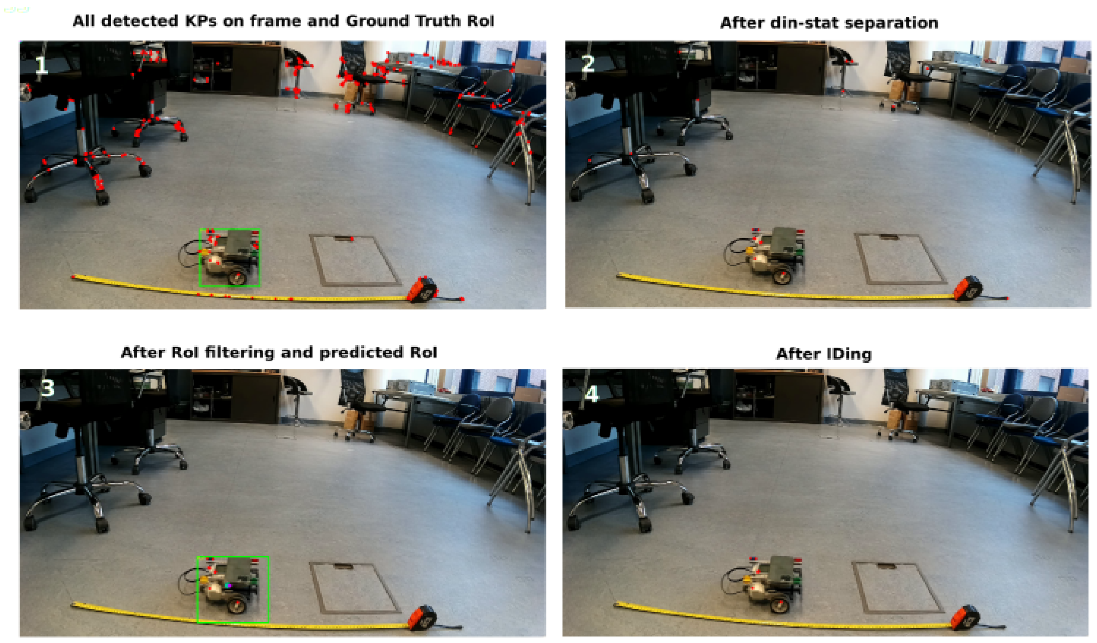
All detected KPs on frame and Ground Truth RoI
1
After din-stat separation
2
After RoI filtering and predicted RoI
3
After IDing
4

Figure 5.3: Steps of the tracker

5.3.1 Multiple Dynamic Objects

Figure 5.4 demonstrates a special case involving multiple dynamic objects simultaneously. The two moving objects in this scenario are a hand holding a paper box and the usual LEGO car. On the left, all detected keypoints in the frame are shown as red points. On the right, the result of the din-stat separation is displayed.

Apart from a few rogue points, the algorithm effectively filters out most static points, leaving primarily the truly dynamic keypoints. The separation results in two distinct clusters of keypoints, one on each dynamic object.

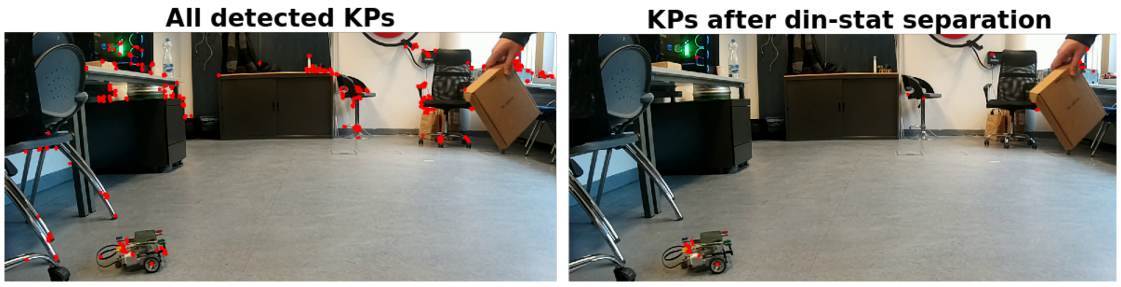

Figure 5.4: Operation of din-stat algorithm in presence of multiple dynamic objects.

Chapter 6

Evaluation

This chapter evaluates the proposed approach across a variety of controlled and real-world conditions. It begins with a description of the test environment and data preparation, including object segmentation via a trained model. Key experiments then examine the effects of distance, rotation, and environmental factors on feature matching performance. Additional analyses explore dynamic keypoint management, tessellation, and timing measurements, demonstrating both the algorithm's strengths and areas needing improvement. Ultimately, these results inform future refinements, ensuring a more robust and efficient solution.

6.1 Test Environment

The images used for testing were captured in one of the rooms of Building I at the university, using an OAK-D Pro W camera. The target object was a small LEGO car. Approximately 450 images were taken, capturing the object from various distances, angles, and perspectives to ensure comprehensive coverage of scenarios relevant for subsequent processing as we can see on Figure 6.1.

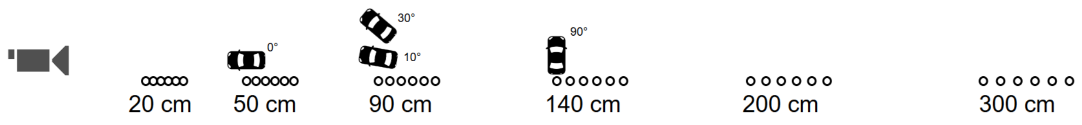

Figure 6.1: Visualization of the research and evaluation setup

The images were categorized based on the following parameters:

- Distance Variation: Images were captured at six distinct distances within seven main distance groups. The starting distances for these groups were 20, 50, 90, 140, 200, 300, and 500 cm, measured between the camera and the target object. Within each group, incremental changes in distance were applied, with step sizes of 1, 2, 3, 4, 5, 6, and 20 cm. For example, the first set of distances included 20, 21, 22, 23, 24, and 25 cm.

- Rotation: At each distance, multiple profiles of the target object were captured. The orientation directly facing the camera was defined as 0°. From this position, the object was rotated clockwise around an anchor point at its front in 10° increments, capturing images at angles of 10°, 20°, 30°, 40°, 50°, 60°, 70°, 80°, and 90°.

- Camera Movement: For this setup, the target object remained stationary at the 50 cm mark. Instead, the camera was shifted laterally to the right by 10, 20, 30, 40, and 50 cm. This movement resulted in apparent object rotations of approximately 11.31°, 21.801°, 30.964°, 38.66°, and 45°, while also introducing changes to the environment in the captured images.
- Small Rotation: Additional images were taken at finer rotation increments of 2° from the 50 cm distance. This was done because, as discussed later, rotations of 10° already pose significant challenges for subsequent processing.

The algorithm itself must be tested and evaluated on appropriate data. For this purpose, I captured videos of the LEGO car moving along various trajectories, including a straight line, a rotated L-shape, a half-circle, a full circle, and free roaming. These videos were recorded at multiple movement speeds to assess the algorithm's performance under varying conditions.

6.2 Object Segmentation Model

To enable proper testing without the need to manually draw a mask for the target object in every new scenario—a process that would have been both unsustainable and inconsistent given the hundreds of images involved—I trained a segmentation model based on Detectron2 using transfer learning. The result, shown in Figure 6.2, demonstrate that the generated mask is highly accurate. To ensure consistency during testing and evaluation, the generated masks were saved as separate images, with filenames referencing their corresponding original images.

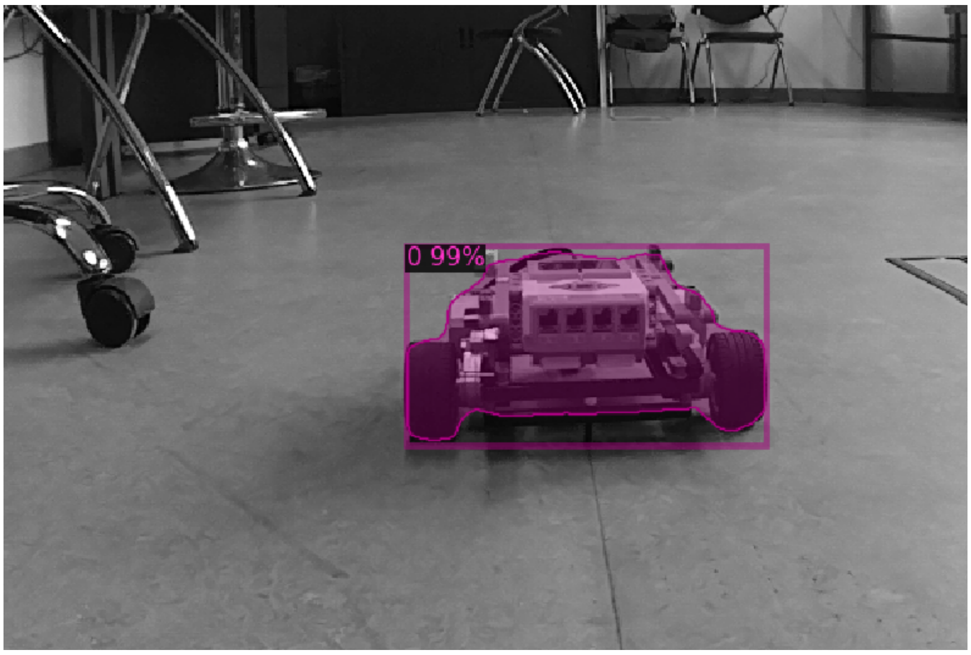

Figure 6.2: Operation of the Detectron2 segmentation model made with transfer learning

6.3 Measurements Demonstrating the Need for the Solution

As intuitively expected, the number of successful matches decreases as the distance between the camera and the object increases. Figure 6.3 illustrates the results of this measurement. The trend lines confirm the assumption, showing a clear decreasing trend as the distance grows. Given that 500 keypoints were detected in both images, the maximum possible number of matches was also 500.

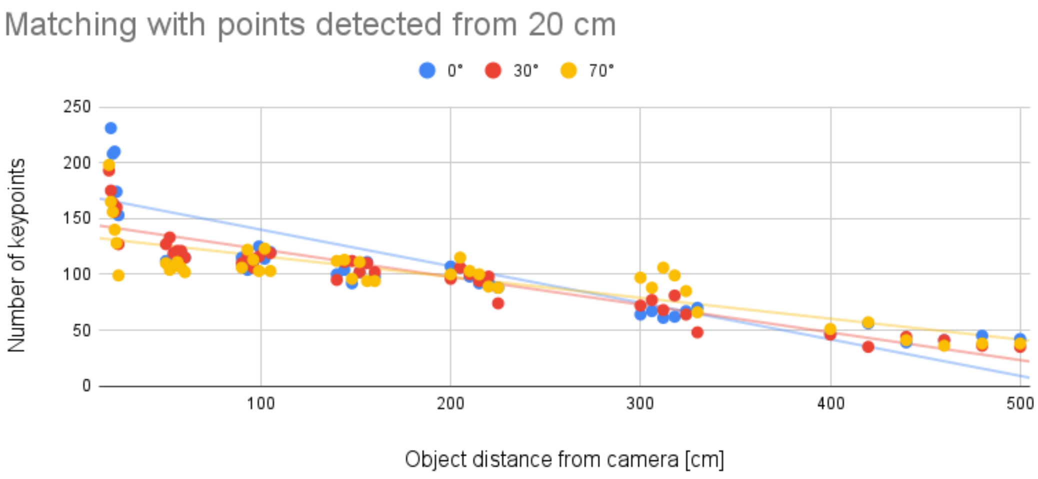

Figure 6.3: Number of matches between an image taken at 20 cm and other images at 0°, 30°, and 70° orientations.

Figure 6.4 presents how the number of keypoints detected on the object decreases as the target object becomes smaller in the image. For this measurement, the detection algorithm was configured to search for 500 keypoints. For example, when the toy car was positioned at 0° and 105 cm from the camera, 157 keypoints were detected on the object.

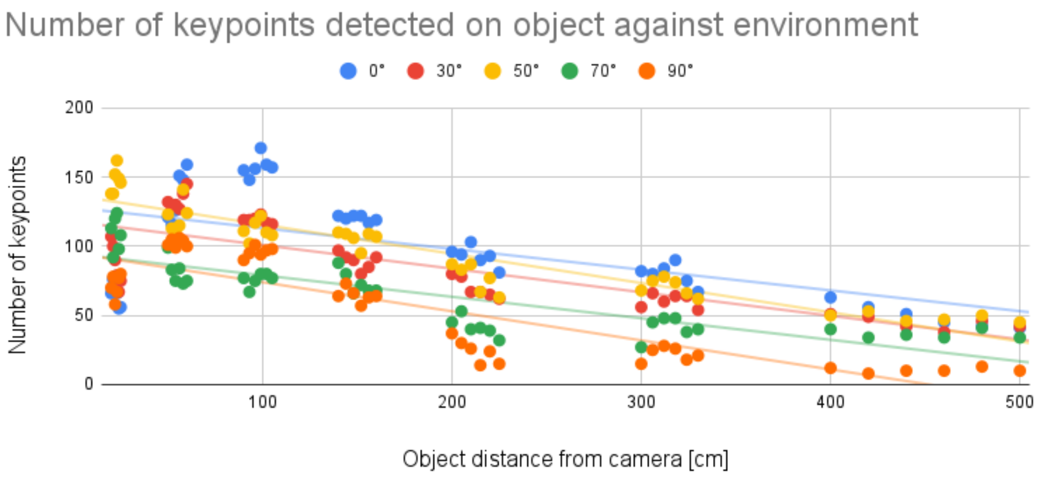

Figure 6.4: Number of keypoints on the object decreasing proportionally with distance.

Figure 6.5 depicts the number of keypoints that can be matched as the target object rotates around a fixed point while maintaining a constant distance from the camera. This measurement was taken at a distance of 99 cm from the object. The maximum number of matches, as derived from previous measurements, averages approximately 120. The trend line indicates a sharp drop in matches between 20° and 30° of rotation. Beyond this range, consistent matches where the same keypoint is identified on the object become practically nonexistent. In most cases, similar-looking points are matched, but they correspond to completely different locations on the car. As the rotation angle increases, the probability of correctly matching keypoints continues to decrease. If fortunate, a few accurate matches may still be found at 30°.

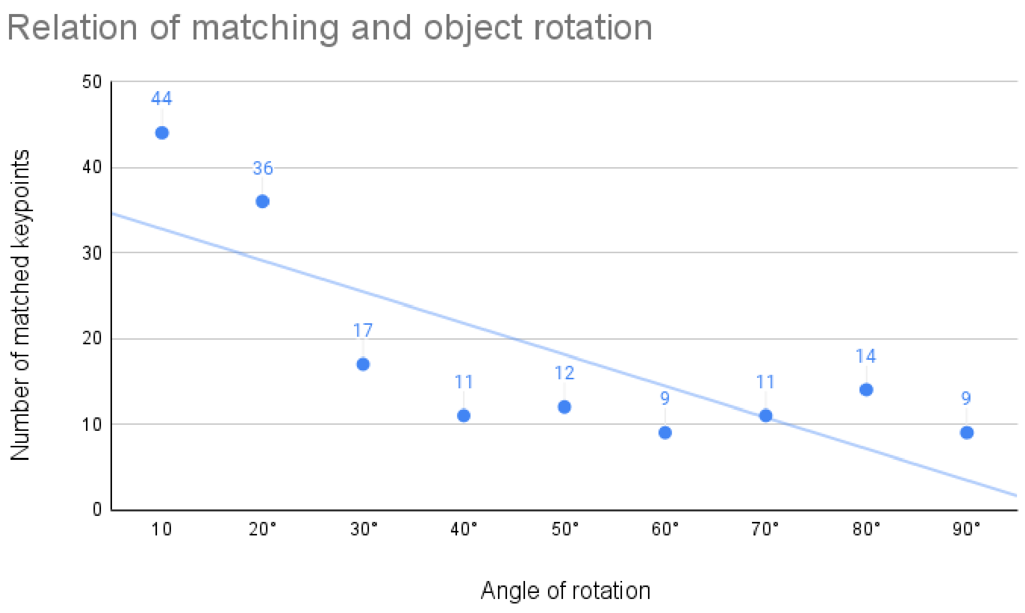

Figure 6.5: Number of matching keypoints decreasing proportionally with rotation.

I also conducted measurements in which two images were taken, with 500 keypoints detected in each, and then attempted to track the best matching 50 points across a sequence of approximately 14 images. The images were captured sequentially to simulate the slow distancing of the target object. If a keypoint could not be found in the next image, it was eliminated from the set of keypoints searched in subsequent images. Manual verification was later performed to determine whether the algorithm consistently identified the same point throughout the sequence.

The results revealed that, out of 500 initial keypoints, only 3-4 survived a slow retreat of 1 meter, even when the object orientation remained constant. This scenario, involving a gradual withdrawal, was the most favorable among all tested cases in terms of keypoint tracking.

From these measurements, it can be concluded that dynamic keypoint management is essential, as it is rare for a keypoint to be consistently matched across more than three consecutive images.

6.4 Measurements Leading to the Algorithm

Figure 6.6 presents a portion of the results from these measurements, specifically comparing ORB angle changes in ideal (top) and contaminated (bottom) scenarios. Since the differences between the cases are quite similar, it suffices to analyze just one of them. Based on the graphs, the following observations were made:

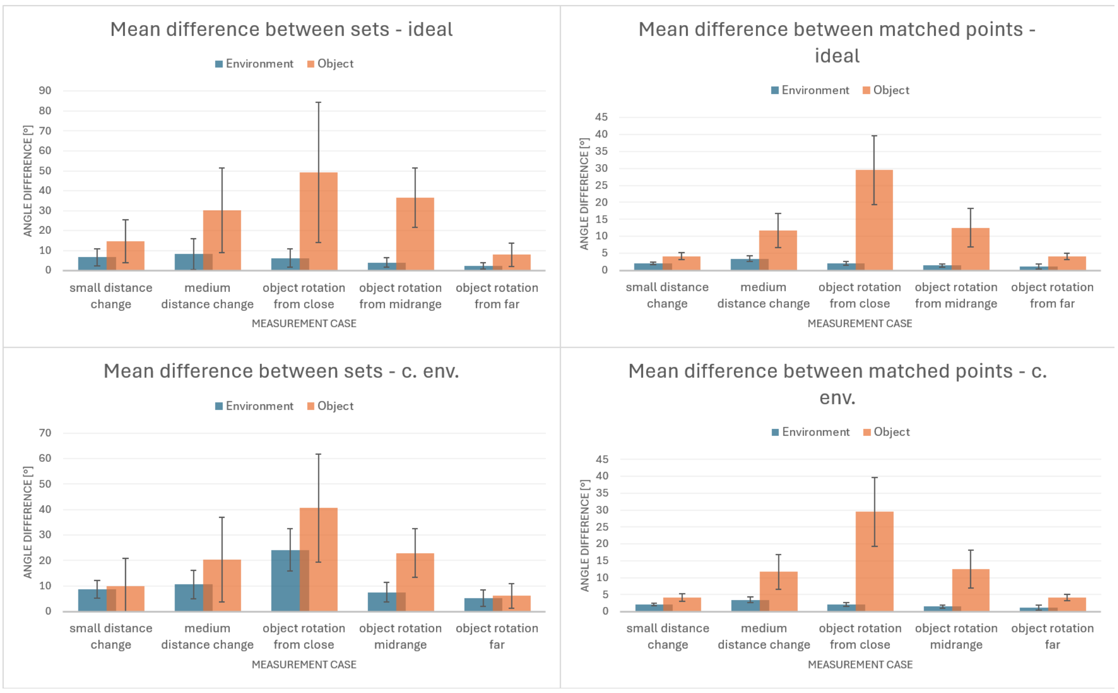

Figure 6.6: Average angle differences between ORB keypoints. The left side shows set-based averages, while the right side shows matching-based averages. The ideal scenario is depicted at the top, and the contaminated scenario is at the bottom.

- The object consistently exhibits greater orientation changes than the environment, whether due to distance increase or rotation. This holds true for both the general set-based and matched-point-based measurements. Based on this, the moving or dynamic object appears distinguishable from its static environment.
- As the distance to the object increases, the difference in changes between the object and the environment decreases in the case of rotation. This trend is not observed when simply increasing the distance between the camera and the object while keeping the rotation fixed, possibly because testing did not occur at sufficiently large distances.
- Rotation induces greater changes than simply increasing the distance, particularly at close distances.
- In the matching-based scenario, the difference values are more moderate (approximately half as much as in the set-based measurements).
- In the case of environmental contamination, the differences are further moderated. However, a significant gap between the object and the environment remains in the matched-point-based measurements. In contrast, for set-based measurements, the

average difference and standard deviation of the environment drastically increase, thereby greatly reducing the ability to distinguish it from the object.

The set-based approach did not yield fruitful results in distinguishing keypoints, especially compared to the matched-point-based approach. The matched-point-based approach proved to be more straightforward and easily applicable for the development of the tracker. For this reason, the data obtained from the matched-point-based measurements (primarily thresholds) were used in the creation of the algorithm, and I will primarily showcase graphs that highlight the results of these measurements.

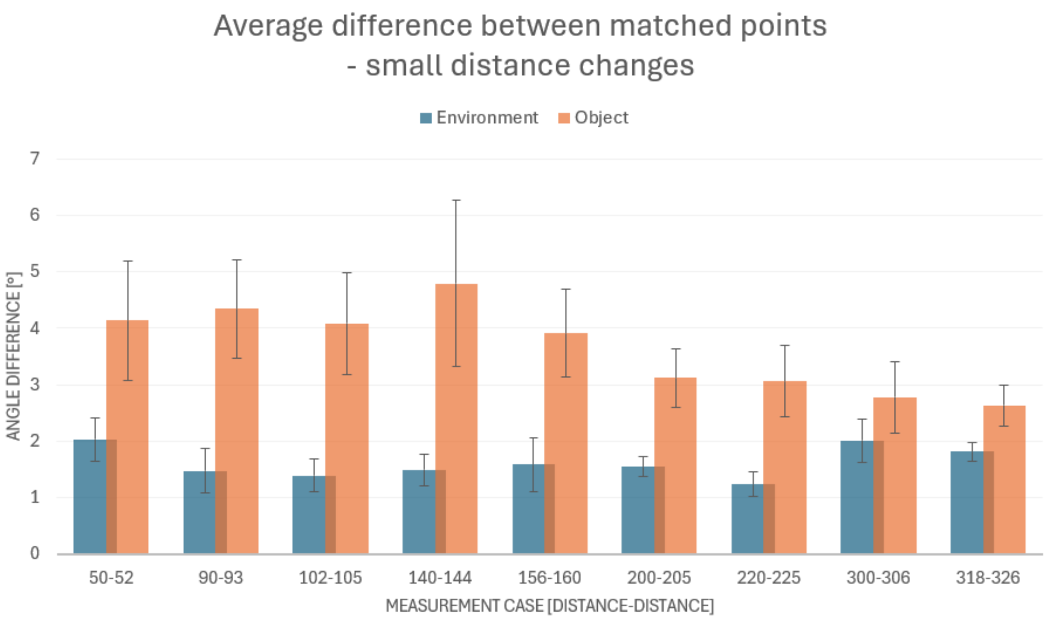

Figure 6.7: Average angle differences between matched ORB keypoints at different distances from the camera in the category of small distance changes.

Let us take a look at Figure 6.7, which zooms into the small distance change scenario of the matched-point-based approach under ideal conditions. Specifically, it highlights the first column pair of the top-right graph in Figure 6.6. This scenario represents the smallest observed difference between the object and the environment, consistent with our current understanding.

Despite the minimal difference in this case, there is still a significant enough gap between the object and the environment to establish distinct thresholds. These thresholds can indicate when the difference between two points suggests that they are not static and represent a true positive match. Ideally, in the case of a true positive match, such a difference would imply that the keypoints are indeed located on the object. However, it is currently unclear how to handle false matches—instances where one keypoint lies on the environment but is incorrectly matched with the object—as their intrinsic differences can vary significantly.

The figure also demonstrates that as the object moves farther from the camera, differentiating it from the environment becomes increasingly challenging, even under optimal conditions. Nonetheless, a discernible and exploitable difference persists, offering a solid basis for the development of effective thresholds.

The next figure, Figure 6.8, shows changes in the response intrinsic characteristic of ORB keypoints. For distance changes alone, this characteristic does not provide much meaningful insight; however, in the ideal case, it can indicate the object's rotation. While not sufficient on its own, it could be used to reinforce results when combined with other characteristics. This figure also highlights how a few keypoints that infiltrate the environment can distort the interpretation of characteristics. In the contaminated scenario, the environment exhibits uniform changes, while the object shows variability, making it challenging to extract actionable information.

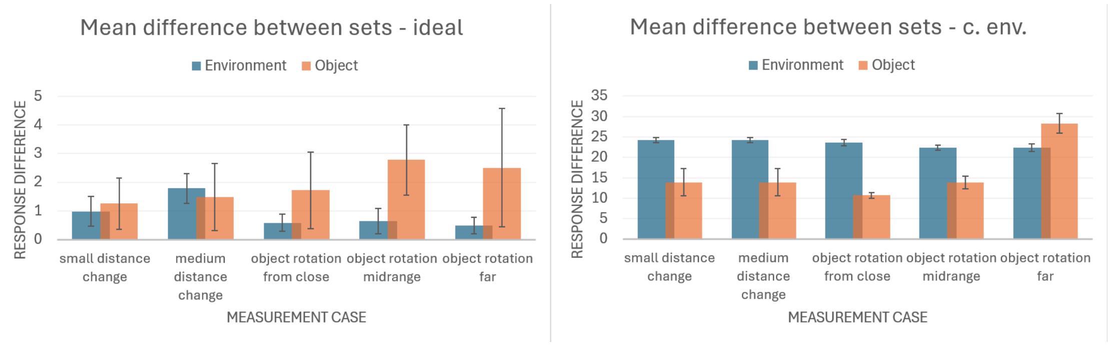

Figure 6.8: Mean response differences between ORB keypoints. Both graphs show set-based results, but the left one is of the ideal case, while the right one is with contaminated environment.

Finally, from this category, Figure 6.9 presents the results for the size characteristic of ORB keypoints. As the findings for the octave characteristic are almost identical, a separate discussion is unnecessary. The following observations were made:

- A similar relationship between rotation and distance was observed as with the angle characteristic: farther objects produce smaller deviations, though this is only evident in the matched-point-based scenario.
- In every scenario, there is a significant difference in the changes between the object and the environment, highlighting the potential for distinguishing dynamic objects.

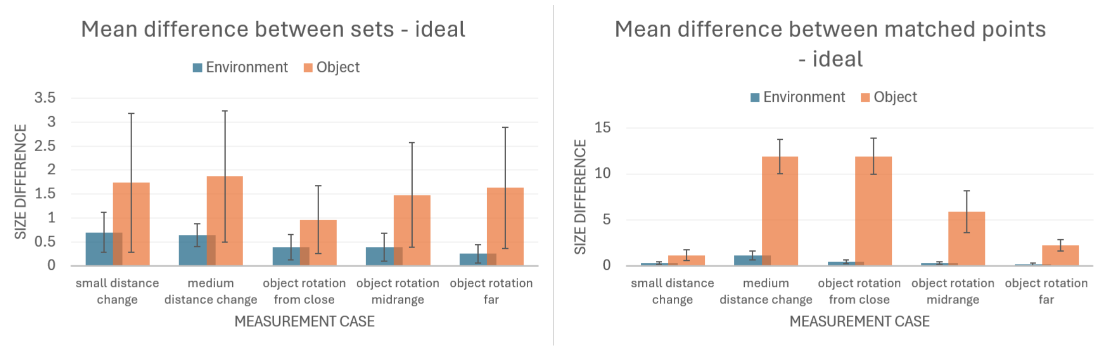

Figure 6.9: Average size differences between ORB keypoints. On the left are set-based averages, while on the right are matching-based averages.

6.4.1 Case of Camera Displacement

The previously presented measurements assume a practically static camera; however, this is rarely the case in real-world environments. Under these conditions, mostly chaotic behavior is observed: in some cases, the object exhibits greater variation, while in others, the environment does. Thus far, no reliable characteristic or combination of metrics has been identified that can accurately predict whether a changing feature is part of the object in these scenarios.

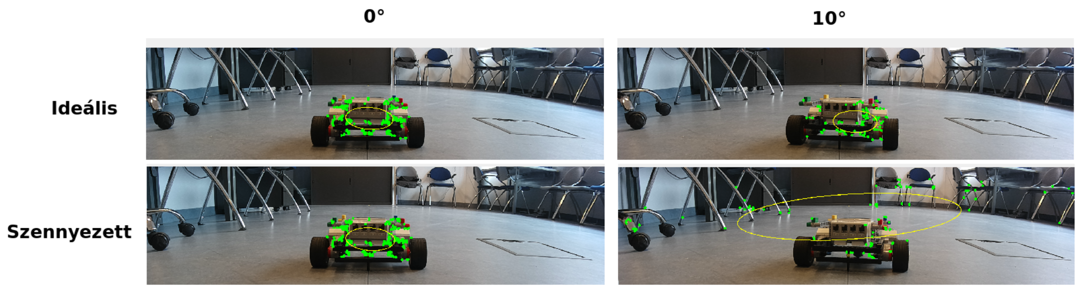

Figure 6.10: PCA results visualized as ellipses for images at 0° and 10° orientation from Table 6.1. The green dots indicate the point set used for PCA.

6.4.2 Tessellation Analysis

In keypoint tessellation, we do not necessarily focus on the 2D keypoint (KP) coordinates—since these coordinates are likely to shift across consecutive frames due to perspective changes—but rather on their relative structural differences. Metrics such as distances or angles between point pairs and eigenvalue ratios are used to assess whether the characteristics of the compared sets remain consistent or change over time. The construction of tessellations, whether based on relative or absolute coordinates, depends on camera movement. In cases where the camera movement can be considered constant, distance-based calculations may also be feasible. The metrics and methods outlined in Section 2.4.1 were used to evaluate the tessellations.

Condition	Average Distance	Standard Deviation	Aspect Ratio Comparison	Variance Coverage
0 - 10 - ideal	108.941	41.043	0.172	1.145
0 - 10 - cont.	380.838	274.497	2.814	0.055
0 - 30 - ideal	149.953	64.32	0.445	1.047
0 - 30 - cont.	412.809	254.569	2.669	0.052
0 - 60 - ideal	185.246	92.221	0.328	0.642
0 - 60 - cont.	415.402	246.9	2.595	0.053

Table 6.1: Tessellation metrics at 10°, 30°, and 60° rotations with an object distance of 50 cm.

Table 6.1 provides a subset of tessellation test results. The metrics, as described in Section 2.4.1, are used to quantify the quality and consistency of keypoints across various conditions. The corresponding images and visualizations for the first two rows of the table are shown in Figure 6.10. In the ideal case (top half of Figure 6.10), when the points of an object are compared with points from the same object in subsequent frames, a reasonable number of acceptable matches are found with relatively high accuracy. This result is reflected in metrics such as variance coverage, which is greater than 1. Conversely, in the contaminated case (bottom half of Figure 6.10), where the points from the first image are compared with all points in the second image, the PCA metrics reveal significant differences. These differences are visually evident from the larger ellipse size, indicating a less consistent point set.

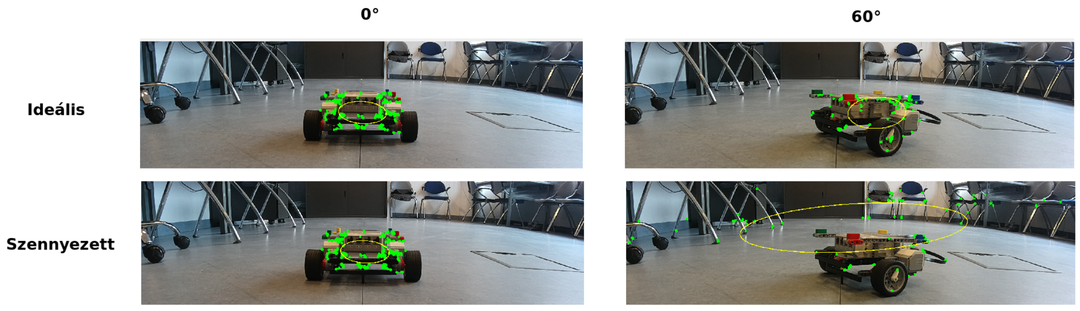

Figure 6.11: PCA results visualized as ellipses for images at 0° and 60° orientation from Table 6.1. The green dots indicate the point set used for PCA.

Based on these observations, tessellation analysis will primarily serve as a method to validate the quality of the obtained keypoint sets. This method is not currently utilized by the feature matcher at any stage of its process. However, it shows promise and is planned to be integrated into the system as a validation mechanism.

Figure 6.11 provides a similar visualization for a larger orientation change. While the variance coverage metric does not reach a value of 1, there is still a significant difference between the ideal and contaminated cases, both numerically and visually.

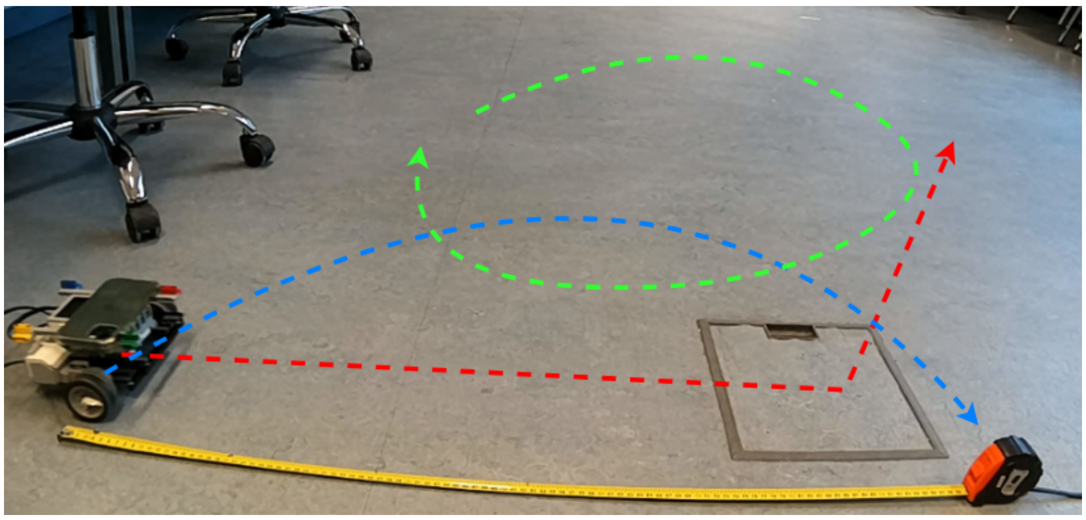

Figure 6.12: The investigated trajectories visualized on real footage. The red trajectory is of the fast and slow L, green is of the 'big circle', and the blue trajectory belongs to the fast and slow 'arc'.

6.5 Tested Trajectories

Figure 6.12 shows several of the trajectories used in the subsequent measurements, chosen for their ease of comparison across multiple parameters. The figure features three distinct, color-coded, and dotted trajectories, with arrows indicating their direction of movement.

The measuring tape visible at the bottom of the image is 1 meter long and positioned approximately 80 centimeters away from the camera. The farthest point of the 'big circle' trajectory is located around 2 meters from the camera.

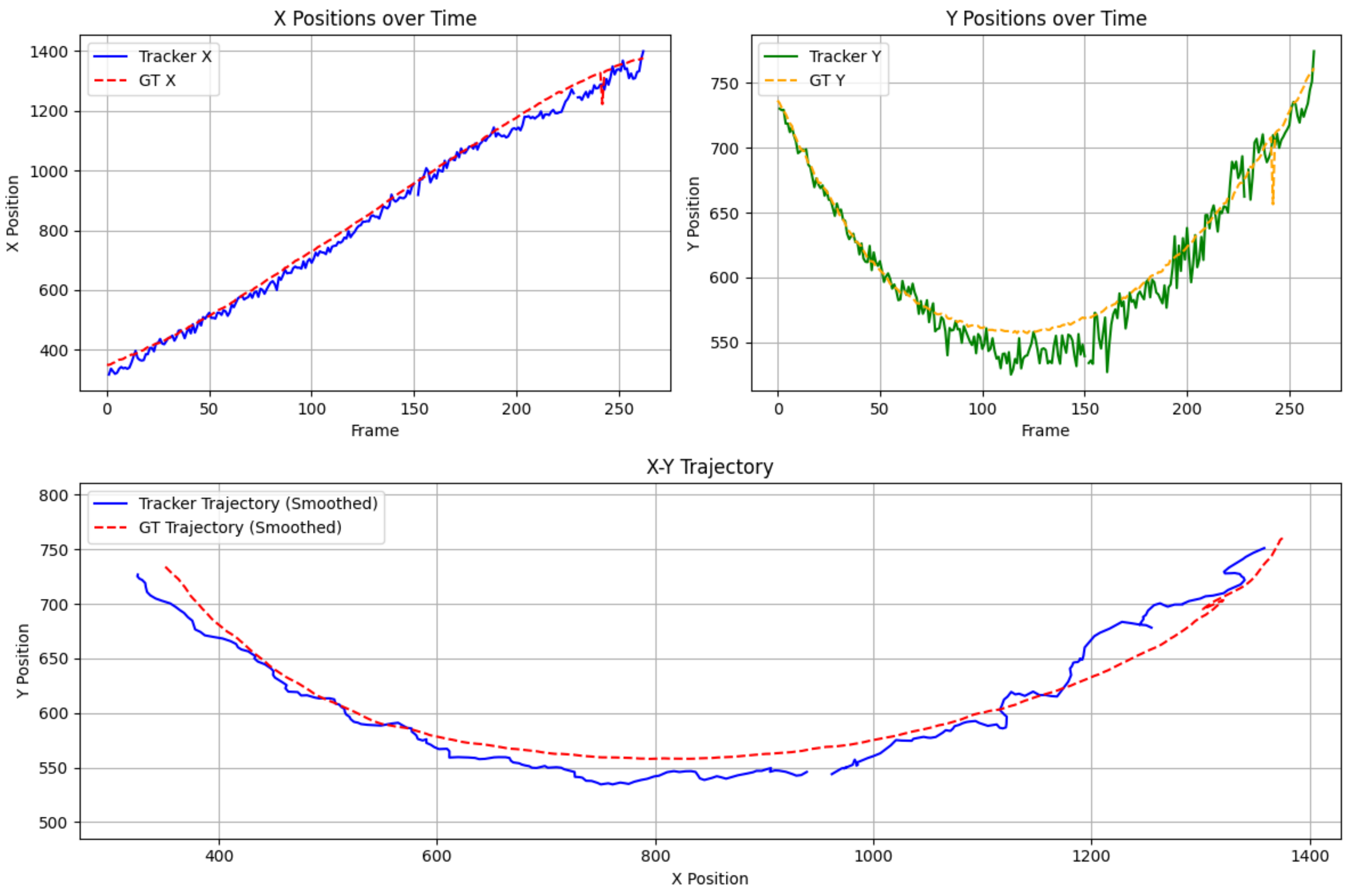

Figure 6.13: Comparison of individual and merged trajectories of the tracker and its ground truth on the 'slow_arc' scenario.

Figure 6.13 illustrates the accuracy of the feature matcher tracker on one of the prerecorded trajectories, referred to as the "slow arc". Here, "slow" indicates a movement speed of approximately $15\frac{cm}{s}$, while "arc" denotes a half-circle trajectory.

At the top of the graph, the X and Y pixel positions are plotted on the Y-axis, with the corresponding frame numbers on the X-axis. The bottom section of the figure displays the merged trajectory derived from the 2D points. Each graph contains two lines: one representing the tracker's output and the other representing the ground truth, obtained using object segmentation. It is important to note that the evaluation of trajectories was conducted using OpenCV, which uses a coordinate system with the origin point at the top-left corner. As a result, the trajectories depicted in Figures 6.13 and 6.15 have their Y-coordinates inverted compared to their natural orientation as shown in Figure 6.12.

To provide a sense of scale, one pixel in this specific measurements corresponds to approximately 0.9 millimeters. Based on this, the average distance between the tracker's results and the ground truth is approximately 2.2 centimeters. It is important to note that the average distance values shown in Figure 6.14, which were used to make this deduction, only include frames where both tracker data and ground truth data are available. If the tracker fails to provide data for a frame, the corresponding ground truth data is excluded from the calculation.

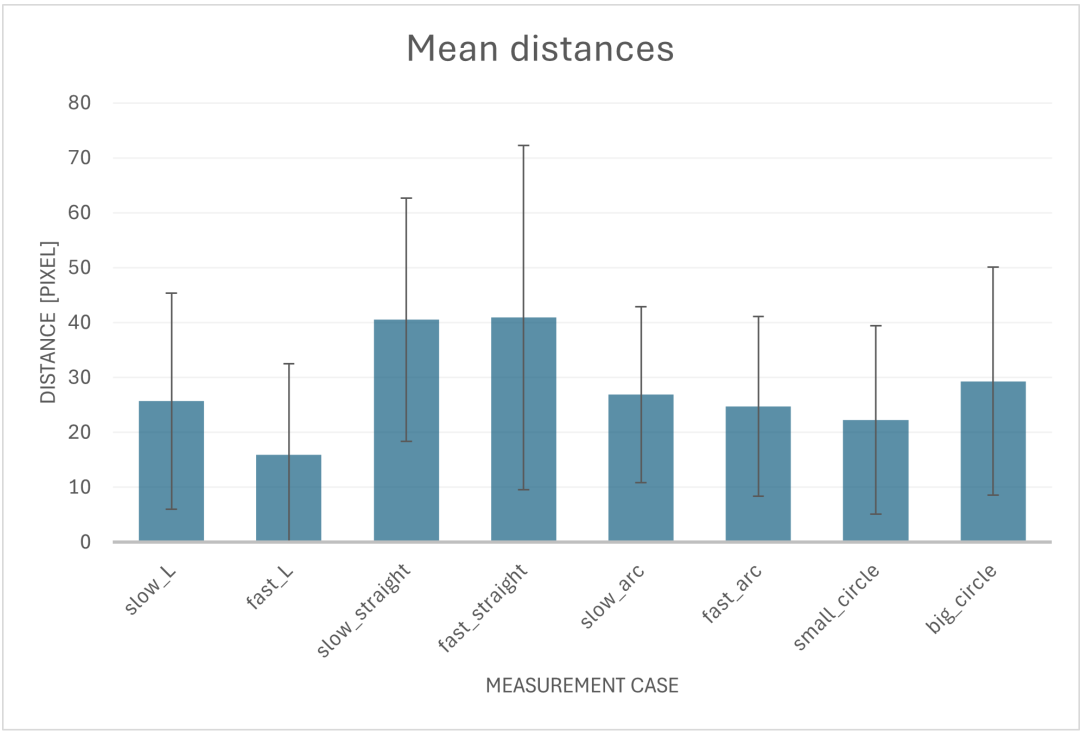

Figure 6.14: Average distance between the trackers result and the ground truth.

Not all scenarios were favorable to the tracker. In the top-left quadrant of Figure 6.15, we observe that when the movement speed increased to approximately $28\frac{cm}{s}$) and the object simultaneously turned while closing its distance to the camera, most of the track was lost. The exact reasons for this failure are still under investigation. One of the most likely explanations is the presence of motion blur, which occurs when factors such as high speed, specific perspectives, and relative positioning to the camera align. Interestingly, in the presence of strong motion blur, the novel din-stat separation drops most of the keypoints from the object. This results in the tracker failing to identify and calculate the 2D position, even though the din-stat separation previously appeared sufficient for identifying moving keypoint pairs. While motion blur also reduces the number of detected and subsequently matched keypoints, its impact on the din-stat separation is more pronounced, leading to a greater degradation in performance. This effect is even more evident in the 'fast_L' scenario (bottom-left quadrant of the figure). In this case, when the object moves perpendicularly to the camera at its maximum speed of approximately $31.25\frac{cm}{s}$, the tracker loses the object entirely. Conversely, when the object follows the same trajectory (bottom-right quadrant of the figure) but at a slower speed, tracking remains mostly continuous and closely aligns with the ground truth.

The challenges encountered on the straight part of the 'fast L' trajectory can be attributed to a combination of subtle differences in keypoint characteristics and the limitations of the current din-stat separation method. The angle differences between matched keypoints, as observed in the data, often fall below the threshold of 3, which was established as the minimum difference for reliable separation in previous measurements. This threshold, however, is not robust enough to distinguish between dynamic points on the object and points from the environment, leading to the rejection of a significant number of valid keypoints.

Other keypoint characteristics, such as size, remain unchanged in this scenario, rendering them ineffective for separation. While response values exhibit variations, their behavior needs to be further analyzed, particularly in video scenarios, to better understand their contribution to object tracking.

Interestingly, the difficulty faced by the tracker does not seem to be solely related to speed, as certain sections of the trajectory still show effective separation. This suggests that the fast straight trajectory differs sufficiently from the measurement scenarios used during development, exposing gaps in the algorithm's ability to generalize.

To address this issue, the threshold could be adjusted, and future work could focus on uncovering deeper relationships between movement types and parameter variations. A potential quick fix might involve lowering the threshold to 0.5, which still effectively discards most static points but retains a greater number of keypoints overall. This adjustment would shift some of the load and responsibility to other steps, such as RoI filtering and IDing, to eliminate invalid keypoints. Furthermore, exploring more adaptive methods for distinguishing between dynamic and static points could enhance the tracker's robustness and reliability across a wider range of scenarios.

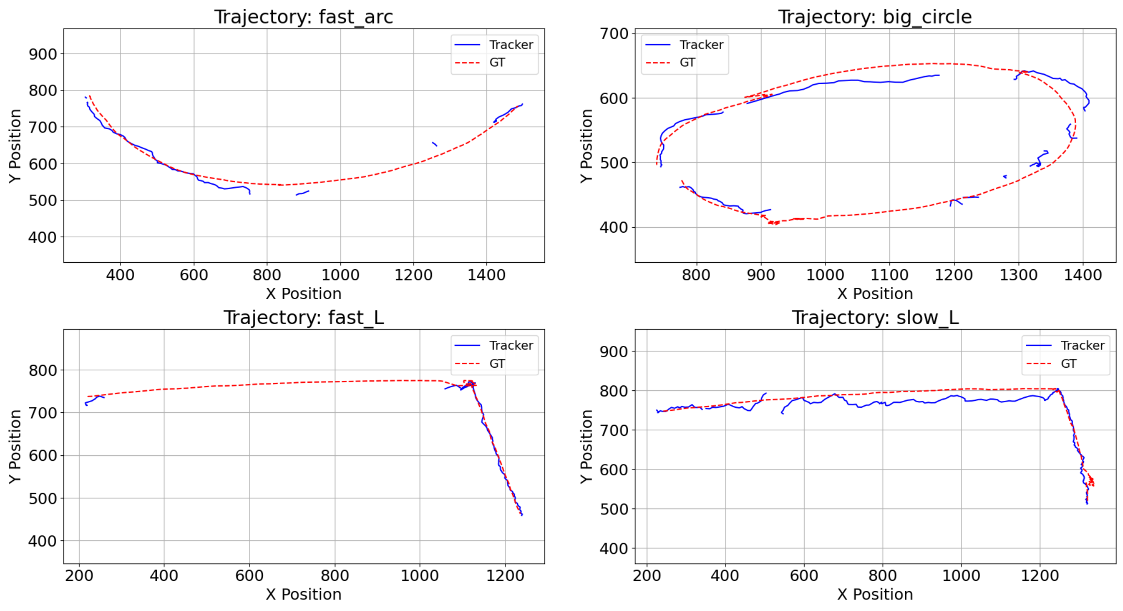

Figure 6.15: 2D trajectories of the measurement cases of fast_arc - top left, big_circle - top right, fast_L - bottom left and slow_L - bottom right.

Figure 6.16 highlights the significant impact of high movement speed and motion blur on the tracker's success rate. The "straight" scenario, involving a trajectory perpendicular to the camera, is particularly challenging in its high-speed version, as it combines factors that exacerbate the tracker's current limitations. This is evident in the results shown. Conversely, in scenarios classified as "slow," the percentage of frames with failed identification cases is reduced to less than 6%.

6.5.1 Timings

The ability for the algorithm to run efficiently is critical, as the ultimate goal is to achieve 60 FPS on a CPU. Despite this aspiration, most of the tests were conducted on frames with a resolution of 1920x1080 while searching for 1000 ORB feature points. This setup was chosen to facilitate testing and measurements. Higher resolution aids in identifying quality feature points, and searching for 1000 keypoints increases the likelihood of finding

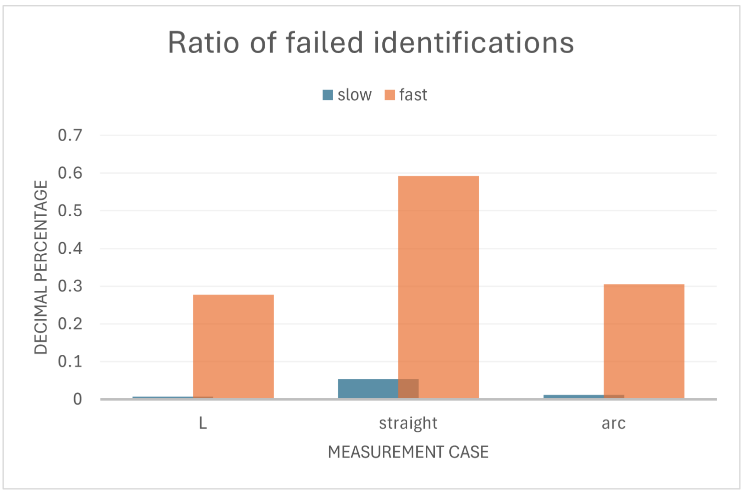

Figure 6.16: Decimal percentage of failed identification in fast vs slow measurement cases

a sufficient number on the object, simplifying the filtering and identification processes. However, this configuration is quite detrimental to timing performance.

Timing values, as shown in Figure 6.17, reveal that the most time-consuming step is currently ORB feature detection, which takes over 25 ms per frame. While this is a significant bottleneck, it is expected to improve with further optimization. Several approaches can help reduce the processing time. The most impactful variable is lowering the resolution of the input frames, which can be achieved either by capturing them natively at a lower resolution or by applying post-processing. Another option is to crop Regions of Interest (RoIs) from the frame and perform feature detection only on smaller sections of the image. Additionally, certain ORB parameters, such as limiting its pyramid levels, can further reduce computation time.

Lowering the resolution to 1280x720 already reduces detection time to approximately 12 ms per frame. However, if the goal is to achieve 60 FPS, which equates to 16.6 ms per frame for the entire pipeline, additional optimizations will be necessary.

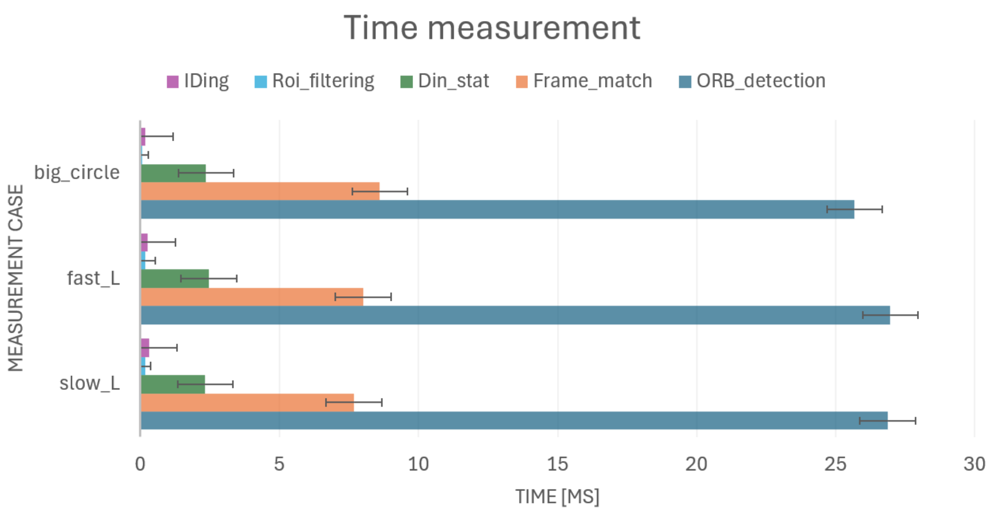

Figure 6.17: Timing measurements of different parts of the feature matcher

Chapter 7

Summary

This thesis has explored innovative approaches to visual object tracking tailored for Extended Reality (XR) applications, a field demanding high precision, real-time performance, and robust adaptation to dynamic environments. The research focused on integrating semantic knowledge into tracking frameworks to improve performance, particularly in XR contexts where latency and accuracy are critical.

1. **Comprehensive Overview of Tracking Technologies:** The study began by surveying the theoretical foundations of object tracking, including single and multi-object tracking (SOT and MOT), and state-of-the-art advancements such as tracking-by-detection and end-to-end methodologies. Metrics for evaluation and technologies like ORB feature extraction and tessellation analysis were also discussed.
2. **Proposed Feature Matching Framework:** A novel, semantically enhanced tracking framework was developed, leveraging object detection, ORB features and various filtering, analysis steps (e.g. principal component analysis).
3. **Methodological Innovation:** The framework introduced a novel way of classification of keypoints as static or dynamic, enhancing semantic filtering.
4. **Evaluation and Performance Analysis:** Performance evaluations revealed that the framework demonstrated reliable tracking in scenarios containing normal or relatively slow motion but encountered challenges in high-speed motion environments. These findings highlight the potential of integrating semantic insights into object tracking while emphasizing the need for further refinement to improve robustness and adaptability under dynamic conditions.

7.1 Limitations

Despite its strengths, the framework has limitations:

- Certain modules rely on manual tuning of parameters, which could be automated for broader applicability.
- The dynamic-static keypoint separation currently functions only with a static camera, as further research is required to identify an exploitable correlation among intrinsic parameters of ORB keypoints. Fast, perpendicular motion near the camera –

at a distance of approximately 1.2 meters and a speed exceeding $30\frac{cm}{s}$ – significantly degrades the algorithm's ability to retain the majority of valid keypoints.

7.2 Future work

A broader range of testing scenarios and datasets is necessary to better understand the algorithm's limitations and strengths. Collecting and analyzing more video footage under diverse environmental conditions, object types, and movement patterns will provide valuable insights for improving robustness and generalization.

Future efforts should focus on creating an adaptive and automated mechanism to adjust parameters dynamically, tailoring the algorithm to suit varying environmental conditions and object behaviors in real time.

Focus on implementing and evaluating keypoint handling in 3D space, as outlined in Section 4.5. This involves leveraging the concept of visible object sides to refine keypoint selection and processing. By incorporating 3D spatial information, the algorithm could improve accuracy in dynamic scenarios, better account for occlusions, and enhance overall robustness in identifying and tracking objects across frames. It is also the first step to occlusion handling.

Enhancing the efficiency of ORB keypoint detection is crucial for meeting real-time requirements. Parallel processing techniques, focused detection within Regions of Interest (RoIs), and determining an optimal resolution for detection (balancing keypoint quality and quantity) should be investigated. Additionally, optimizing the matching process to operate effectively with only the minimal necessary number of keypoints will further streamline the algorithm while preserving accuracy.

Bibliography

[1] Mehmet Aygün, Aljoša Ošep, Mark Weber, Maxim Maximov, Cyrill Stachniss, Jens Behley, and Laura Leal-Taixé. 4d panoptic lidar segmentation. *arXiv preprint arXiv:2102.12472*, 2021. URL `https://arxiv.org/abs/2102.12472`.

[2] Yifan Bai, Zeyang Zhao, Yihong Gong, and Xing Wei. Artrackv2: Prompting autoregressive tracker where to look and how to describe. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2024.

[3] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008(1):246309, 2008. ISSN 1687-5281. DOI: `10.1155/2008/246309`. URL `https://doi.org/10.1155/2008/246309`.

[4] Ignacio T. Berrios. Introduction to oc-sort. *Medium*, 2024. URL `https://medium.com/@itberrios6/introduction-to-ocsort-c1ea1c6adfa2`. Accessed: 2024-05-20.

[5] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016. URL `http://arxiv.org/abs/1602.00763`.

[6] Philippe Blatter, Menelaos Kanakis, Martin Danelljan, and Luc Van Gool. Efficient visual tracking with exemplar transformers. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1571–1581, 2023.

[7] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris Kitani. Observation-centric sort: Rethinking sort for robust multi-object tracking, 2023.

[8] Xiaoyan Cao, Yiyao Zheng, Yao Yao, Huapeng Qin, Xiaoyu Cao, and Shihui Guo. Topic: A parallel association paradigm for multi-object tracking under complex motions and diverse scenes, 2023.

[9] Fei Chen, Xiaodong Wang, Yunxiang Zhao, Shaohe Lv, and Xin Niu. Visual object tracking: A survey. *Computer Vision and Image Understanding*, 222:103508, 2022. ISSN 1077-3142. DOI: `https://doi.org/10.1016/j.cviu.2022.103508`. URL `https://www.sciencedirect.com/science/article/pii/S1077314222001011`.

[10] Jieyu Chen, Zhenghao Xi, Junxin Lu, and Jingjing Ji. Multi-object tracking based on network flow model and orb feature. *Applied Intelligence*, 52:12282–12300, 2022. DOI: `10.1007/s10489-021-03042-6`. URL `https://doi.org/10.1007/s10489-021-03042-6`.

[11] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description, 2018. URL `https://arxiv.org/abs/1712.07629`.

[12] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng. Strongsort: Make deepsort great again, 2023.

[13] Kurt Konolige Ethan Rublee, Vincent Rabaud. Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, 2011. ISSN 2380-7504. DOI: `10.1109/ICCV.2011.6126544`.

[14] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. ISSN 1573-1405. DOI: `10.1007/s11263-009-0275-4`. URL `https://doi.org/10.1007/s11263-009-0275-4`.

[15] Sanjana Vijay Ganesh, Yanzhao Wu, Gaowen Liu, Ramana Kompella, and Ling Liu. Fast and resource-efficient object tracking on edge devices: A measurement study, 2023.

[16] Juan Diego Gonzales Zuniga, Ujjwal Ujjwal, and Francois F Bremond. DeTracker: A Joint Detection and Tracking Framework. In *VISAPP 2022 - 17th International Conference on Computer Vision Theory and Applications*, online, France, February 2022. URL `https://hal.science/hal-03541517`.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018. URL `https://arxiv.org/abs/1703.06870`.

[18] Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, March 2015. ISSN 2160-9292. DOI: `10.1109/tpami.2014.2345390`. URL `http://dx.doi.org/10.1109/TPAMI.2014.2345390`.

[19] Richard Holloway, Frederick Brooks, Jr, Vernon Chi, Henry Fuchs, and Stephen Pizer. Registration errors in augmented reality systems. 07 1999.

[20] Masaya Kaneko, Kazuya Iwami, Torn Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Mask-slam: Robust feature-based monocular slam by masking using semantic segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 371–3718, 2018. DOI: `10.1109/CVPRW.2018.00063`.

[21] Sertis Vision Lab. Multi-object tracking: A review. `https://sertiscorp.medium.com/multi-object-tracking-a-review-6aaeea495209`, June 2022. [Online; accessed May 17, 2024].

[22] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, October 2018. ISSN 1556-4967. DOI: `10.1002/rob.21831`. URL `http://dx.doi.org/10.1002/rob.21831`.

[23] Peiliang Li, Tong Qin, and Shaojie Shen. Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving. *arXiv preprint arXiv:1807.02062*, 2018. URL `https://arxiv.org/abs/1807.02062`.

[24] Shuang Li, Hans Schieber, Nina Corell, Bernhard Egger, Jürgen Kreimeier, and Daniel Roth. GBOT: Graph-based 3d object tracking for augmented reality-assisted assembly guidance. *arXiv preprint arXiv:2402.07677*, 2024. URL `https://arxiv.org/abs/2402.07677`.

[25] Xiaoyu Li, Tao Xie, Dedong Liu, Jinghan Gao, Kun Dai, Zhiqiang Jiang, Lijun Zhao, and Ke Wang. Poly-mot: A polyhedral framework for 3d multi-object tracking, 2023.

[26] Xiaoyu Li, Dedong Liu, Lijun Zhao, Yitao Wu, Xian Wu, and Jinghan Gao. Fast-poly: A fast polyhedral framework for 3d multi-object tracking, 2024.

[27] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip H. S. Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. HOTA: A higher order metric for evaluating multi-object tracking. *CoRR*, abs/2009.07736, 2020. URL `https://arxiv.org/abs/2009.07736`.

[28] Wenhan Luo, Xiaowei Zhao, and Tae-Kyun Kim. Multiple object tracking: A review. *CoRR*, abs/1409.7618, 2014. URL `http://arxiv.org/abs/1409.7618`.

[29] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot challenge results. `https://motchallenge.net/results/MOT17/?det=All`, 2024. Accessed: 2024-05-20.

[30] M. M. Morsali, Z. Sharifi, F. Fallah, S. Hashembeiki, H. Mohammadzade, and S. Bagheri Shouraki. Sfsort: Scene features-based simple online real-time tracker, 2024.

[31] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, October 2015. ISSN 1941-0468. DOI: `10.1109/tro.2015.2463671`. URL `http://dx.doi.org/10.1109/TRO.2015.2463671`.

[32] Gaku Narita, Takashi Seno, Tomoya Ishikawa, and Yohsuke Kaji. Panopticfusion: Online volumetric semantic mapping at the level of stuff and things, 2019. URL `https://arxiv.org/abs/1903.01177`.

[33] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. URL `https://arxiv.org/abs/1804.02767`.

[34] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression, 2019.

[35] Ergys Ristani, Francesco Solera, Roger S. Zou, Rita Cucchiara, and Carlo Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. *CoRR*, abs/1609.01775, 2016. URL `http://arxiv.org/abs/1609.01775`.

[36] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping, 2020. URL `https://arxiv.org/abs/1910.02490`.

[37] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks, 2020. URL `https://arxiv.org/abs/1911.11763`.

[38] Matthias Schörghuber, Daniel Steininger, Yohann Cabon, Martin Humenberger, and Margrit Gelautz. Slamantic - leveraging semantics to improve vslam in dynamic environments. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3759–3768, 2019. DOI: 10.1109/ICCVW.2019.00468.

[39] Yoones Sekhavat and Jeffrey Parsons. The effect of tracking technique on the quality of user experience for augmented reality mobile navigation. *Multimedia Tools and Applications*, 77:1–34, 05 2018. DOI: 10.1007/s11042-017-4810-y.

[40] Martin Simon, Karl Amende, Andrea Kraus, Jens Honer, Timo Sämann, Hauke Kaulbersch, Stefan Milz, and Horst Michael Gross. Complexer-yolo: Real-time 3d object detection and tracking on semantic point clouds. *arXiv preprint arXiv:1904.07537*, 2019. URL https://arxiv.org/abs/1904.07537.

[41] Daniel Stadler and Jürgen Beyerer. BYTEv2: Associating more detection boxes under occlusion for improved multi-person tracking. In *Pattern Recognition, Computer Vision, and Image Processing. ICPR 2022 International Workshops and Challenges*, pages 79–94. Springer, 2023. DOI: 10.1007/978-3-031-37660-3_6. URL https://link.springer.com/chapter/10.1007/978-3-031-37660-3_6.

[42] Rainer Stiefelhagen, Keni Bernardin, Rachel Bowers, John Garofolo, Djamel Mostefa, and Padmanabhan Soundararajan. The clear 2006 evaluation. In Rainer Stiefelhagen and John Garofolo, editors, *Multimodal Technologies for Perception of Humans*, pages 1–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-69568-4.

[43] Michael Stoiber, Mohamed Elsayed, Andreas E. Reichert, Florian Steidle, Dongheui Lee, and Rudolph Triebel. BYTEv2: Associating more detection boxes under occlusion for improved multi-person tracking. In *Pattern Recognition, Computer Vision, and Image Processing. ICPR 2022 International Workshops and Challenges*, pages 79–94. Springer, 2023. DOI: 10.1007/978-3-031-37660-3_6. URL https://link.springer.com/chapter/10.1007/978-3-031-37660-3_6.

[44] Chao Sun, Ning Qiao, and Jian Sun. Robust feature matching based on adaptive orb for vision-based robot navigation. In *Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 282–287, 2021. DOI: 10.1109/ICRA48506.2021.9486546. URL https://ieeexplore.ieee.org/document/9486546.

[45] Jiaming Sun, Yiming Xie, Siyu Zhang, Linghao Chen, Guofeng Zhang, Hujun Bao, and Xiaowei Zhou. You don't only look once: Constructing spatial-temporal memory for integrated 3d object detection and tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3185–3194, 2021. DOI: 10.1109/ICCV48922.2021.00320. URL https://ieeexplore.ieee.org/document/9710513.

[46] ARTrackV2 Team. Artrackv2: Unified generative framework for visual object tracking. https://artrackv2.github.io/, 2024. Accessed: 2024-05-19.

[47] OpenCV Team. About. https://opencv.org/about/. [Online; accessed November 1, 2023].

[48] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3D Multi-Object Tracking: A Baseline and New Evaluation Metrics. *IROS*, 2020.

[49] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017. URL `http://arxiv.org/abs/1703.07402`.

[50] Shibiao Wu, Yao Fan, Shibin Zheng, and Hong Yang. Object tracking based on orb and temporal-spatial constraint. In *Proceedings of the 2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*, pages 597–600, 2012. DOI: `10.1109/ICACI.2012.6463210`. URL `https://ieeexplore.ieee.org/document/6463210`.

[51] Xiaohui Xie and Hong Wu. Orb tracking via random model and sample consensus. In *Proceedings of the 5th International Congress on Image and Signal Processing (CISP)*, pages 113–117, 2012. DOI: `10.1109/CISP.2012.6469590`. URL `https://ieeexplore.ieee.org/document/6469590`.

[52] Bayang Xue, Zhong Yang, Luwei Liao, Chi Zhang, Hao Xu, and Qiuyan Zhang. High precision visual localization method of uav based on feature matching. *Frontiers in Computational Neuroscience*, 16, 2022. DOI: `10.3389/fncom.2022.1037623`. URL `https://doi.org/10.3389/fncom.2022.1037623`.

[53] Bin Yan, Houwen Peng, Jianlong Fu, Dong Wang, and Huchuan Lu. Learning spatio-temporal transformer for visual tracking. *CoRR*, abs/2103.17154, 2021. URL `https://arxiv.org/abs/2103.17154`.

[54] Botao Ye, Hong Chang, Bingpeng Ma, Shiguang Shan, and Xilin Chen. Joint feature learning and relation modeling for tracking: A one-stream framework. In *ECCV*, 2022.

[55] Kefu Yi, Kai Luo, Xiaolei Luo, Jiangui Huang, Hao Wu, Rongdong Hu, and Wei Hao. Ucmctrack: Multi-object tracking with uniform camera motion compensation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(7):6702–6710, Mar. 2024. DOI: `10.1609/aaai.v38i7.28493`.

[56] Hao Yu, Qiang Fu, Zhen Yang, Liang Tan, Wei Sun, and Mingui Sun. Robust robot pose estimation for challenging scenes with an rgb-d camera. *IEEE Sensors Journal*, 19(6):2217–2229, 2019. DOI: `10.1109/JSEN.2018.2889820`. URL `https://ieeexplore.ieee.org/document/8554288`.

[57] Shaojie Zhang, Yinghui Wang, Jiaxing Ma, Wei Li, Jinlong Yang, Tao Yan, Yukai Wang, Liangyi Huang, Mingfeng Wang, and Ibragim R. Atadjanov. A feature matching method based on multi-level refinement strategy. *arXiv preprint arXiv:2402.13488*, 2024. URL `https://arxiv.org/abs/2402.13488`.