# From Perception to Actions: Autonomous Exploration, Synthetic Data, and Dynamic Worlds

**Dissertation**
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Elia Bonetto
aus Vicenza / Italien

Tübingen
2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

| | |
|---|---|
| Tag der mündlichen Qualifikation: | 23.01.2026 |
| Dekan: | Prof. Dr. Thilo Stehle |
| 1. Berichterstatter: | Jun. Prof. Dr.-Ing. Aamir Ahmad |
| 2. Berichterstatter: | Prof. Dr.-Ing. Hendrik P. A. Lensch |

The greatest ideas are the simplest. – William Golding, Lord of the Flies

# Abstract

This thesis explores innovative methods and frameworks to enhance intelligent systems' visual perception capabilities. Vision is the primary means by which many animals perceive, understand, learn, reason about, and interact with the world to achieve their goals. Unlike animals, intelligent systems must acquire these capabilities by processing raw visual data captured by cameras using computer vision and deep learning.

First, we consider a crucial aspect of visual perception in intelligent systems: understanding the structure and layout of the environment. To enable applications such as object interaction or extended reality in previously unseen spaces, these systems are often required to estimate their own motion. When operating in novel environments, they must also construct a map of the space. Together, we have the essence of the Simultaneous Localization and Mapping (SLAM) problem. However, pre-mapping environments can be impractical, costly, and unscalable in scenarios like disaster response or home automation. This makes it essential to develop robots capable of autonomously exploring and mapping unknown areas, a process known as Active SLAM.

Active SLAM typically involves a multi-step process in which the robot acts on the available information to decide the next best actions. The goal is to autonomously and efficiently explore environments without using prior information. Despite an extensive history, Active SLAM methods focused only on short- or long-term objectives, without considering the totality of the process or adapting to the ever-changing states. Addressing these gaps, we introduce iRotate to capitalize on continuous information-gain prediction. Distinct from prevailing approaches, iRotate constantly (pre)optimizes camera viewpoints acting on i) long-term, ii) short-term, and iii) real time objectives. By doing this, iRotate significantly reduces energy consumption and localization errors, thus diminishing the exploration effort — a substantial leap in efficiency and effectiveness.

iRotate, like many other SLAM approaches, leverages the assumption of operating in a static environment. Dynamic components in the scene significantly impact SLAM performance in the localization, place recognition, and optimization steps, hindering the widespread adoption of autonomous robots. This stems from the difficulties of collecting diverse ground truth information in the real world and the long-standing limitations of simulation tools. Testing directly in the real world is costly and risky without prior simulation validation. Datasets instead are inherently static and non-interactive making them useless for developing autonomous approaches. Then, existing simulation tools often lack the visual realism and flexibility to create and control fully customized experiments to bridge the gap between simulation and the real world. This thesis addresses the challenges of obtaining ground truth data and simulating dynamic environments by

introducing the GRADE framework. Through a photorealistic rendering engine, we enable online and offline testing of robotic systems and the generation of richly annotated synthetic ground truth data. By ensuring flexibility and repeatability, we allow the extension of previous experiments through variations, for example, in scene content or sensor settings.

Synthetic data can first be used to address several challenges in the context of Deep Learning (DL) approaches, e.g. mismatched data distribution between applications, costs and limits of data collection procedures, and errors caused by incorrect or inconsistent labeling in training datasets. However, the gap between the real and simulated worlds often limits the direct use of synthetic data making style transfer, adaptation techniques, or real-world information necessary. Here, we leverage the photorealism obtainable with GRADE to generate synthetic data and overcome these issues. First, since humans are significant sources of dynamic behavior in environments and the target of many applications, we focus on their detection and segmentation. We train models on real, synthetic, and mixed datasets, and show that using only synthetic data can lead to state-of-the-art performance in indoor scenarios.

Then, we leverage GRADE to benchmark several Dynamic Visual SLAM methods. These often rely on semantic segmentation and optical flow techniques to identify moving objects and exclude their visual features from the pose estimation and optimization processes. Our evaluations show how they tend to reject too many features, leading to failures in *accurately* and *fully* tracking camera trajectories. Surprisingly, we observed low tracking rates not only on simulated sequences but also in real-world datasets. Moreover, we also show that the performance of the segmentation and detection models used are not always positively correlated with the ones of the Dynamic Visual SLAM methods. These failures are mainly due to incorrect estimations, crowded scenes, and not considering the different motion states that the object can have. Addressing this, we introduce DynaPix. This Dynamic Visual SLAM method estimates per-pixel motion probabilities and incorporates them into a new enhanced pose estimation and optimization processes within the SLAM backend, resulting in longer tracking times and lower trajectory errors.

Finally, we use GRADE to address the challenge of limited and inaccurate annotations of wild zebras, particularly for their detection and pose estimation when observed by unmanned aerial vehicles. Leveraging the flexibility of GRADE, we introduce ZebraPose — the first full top-down synthetic-to-real detection and 2D pose estimation method. Unlike previous approaches, ZebraPose demonstrates that both tasks can be performed using only synthetic data, eliminating the need for costly data collection campaigns, time-consuming annotation procedures, or syn-to-real transfer techniques.

Ultimately, this thesis demonstrates how combining perception with action can overcome critical limitations in robotics and environmental perception, thereby advancing the deployment of intelligent and autonomous systems for real-world applications. Through innovations like iRotate, GRADE, and ZebraPose, it paves the way for more robust, flexible, and efficient intelligent systems capable of navigating dynamic environments.

# Zusammenfassung

Diese Arbeit untersucht innovative Methoden und Konzepte zur Verbesserung der visuellen Wahrnehmungsfähigkeiten intelligenter Systeme. Sehen ist für viele Tiere das wichtigste Mittel, um die Welt wahrzunehmen, zu verstehen, zu lernen, zu begreifen und mit ihr zu interagieren, um dadurch ihre Ziele zu erreichen. Im Gegensatz zu Tieren müssen intelligente Systeme diese Fähigkeiten durch die Verarbeitung von Rohdaten aus Kameras mithilfe von Computer Vision und Deep Learning erwerben.

Wir betrachten zunächst einen entscheidenden Aspekt der visuellen Wahrnehmung intelligenter Systeme: das Verständnis der Struktur und Organisation der Umgebung. Um Anwendungen wie Objektinteraktion oder erweiterte Realität in bisher unbekannten Räumen zu ermöglichen, müssen diese Systeme häufig ihre eigene Bewegung einschätzen können. Wenn sie in neuen Umgebungen betrieben werden, müssen sie außerdem in der Lage sein, eine Karte des Raums zu erstellen. Zusammen ergibt sich daraus das Kernproblem der simultanen Lokalisierung und Kartierung (Simultaneous Localization and Mapping - SLAM). Allerdings kann die Vorabkartierung von Umgebungen in Szenarien wie Katastropheneinsätzen oder der Hausautomation unpraktisch, kostspielig und nicht skalierbar sein. Daher ist es unerlässlich, Roboter zu entwickeln, die in der Lage sind, unbekannte Gebiete autonom zu erkunden und zu kartieren – ein Prozess, der als Active SLAM bekannt ist.

Active SLAM umfasst in der Regel einen mehrstufigen Prozess, bei dem ein Roboter auf der Grundlage der verfügbaren Informationen die nächsten und besten Aktionen auswählt. Das Ziel besteht darin, Umgebungen autonom und effizient zu erkunden, ohne auf Vorabinformationen zurückgreifen zu müssen. Trotz ihrer langen Geschichte konzentrierten sich Active SLAM-Methoden bisher nur auf kurz- oder langfristige Ziele, ohne den Gesamtprozess zu berücksichtigen oder sich an die sich ständig ändernden Zustände anzupassen. Um diese Lücken zu schließen, führen wir iRotate ein, um die kontinuierliche Vorhersage von Informationsgewinnen zu nutzen. Im Gegensatz zu den gängigen Ansätzen optimiert iRotate ständig die Kameraperspektiven, indem es auf i) langfristige, ii) kurzfristige und iii) Echtzeit-Ziele einwirkt. Auf diese Weise reduziert iRotate den Energieverbrauch und Lokalisierungsfehler signifikant und verringert somit den Erkundungsaufwand – ein erheblicher Sprung in Sachen Effizienz und Effektivität.

iRotate nutzt, wie viele andere SLAM-Ansätze auch, die Annahme, dass es in einer statischen Umgebung eingesetzt wird. Dynamische Komponenten in der Szene haben einen erheblichen Einfluss auf die SLAM-Leistung hinsichtlich Lokalisierung, Ortserkennung und Optimierung und sind einer breiten Einführung autonomer Roboter hinderlich. Dies ist auf die Schwierigkeiten bei der Erfassung diverser Ground-Truth-

Informationen in der realen Welt und die seit langem bestehenden Limitierungen von Simulationswerkzeugen zurückzuführen. Das direkte Testen in der realen Welt ist ohne vorherige Validierung in Simulationen kostspielig und riskant. Datensätze sind hingegen von Natur aus statisch und nicht interaktiv, was sie für die Entwicklung autonomer Ansätze unbrauchbar macht. Außerdem mangelt es bestehenden Simulationswerkzeugen oft an visuellem Realismus und Flexibilität, um vollständig angepasste Experimente zu erstellen und zu steuern, die die Lücke zwischen Simulation und realer Welt schließen. Diese Arbeit befasst sich mit den Herausforderungen der Gewinnung von Ground-Truth-Daten und der Simulation von dynamischen Umgebungen durch die Einführung des GRADE framework. Durch eine fotorealistische Rendering-Engine ermöglichen wir Online- und Offline-Tests von Robotersystemen und die Generierung reichhaltig annotierter synthetischer Ground-Truth-Daten. Durch die Gewährleistung von Flexibilität und Wiederholbarkeit ermöglichen wir die Erweiterung früherer Experimente durch Variationen, beispielsweise in Bezug auf Szeneninhalte oder Sensoreinstellungen.

Synthetische Daten können zunächst dazu verwendet werden, mehrere Herausforderungen im Zusammenhang mit Deep-Learning-Ansätzen (DL) anzugehen, z. B. eine ungleichmäßige Datenverteilung zwischen Anwendungen, Kosten und Grenzen von Datenerfassungsverfahren sowie Fehler, die durch falsches oder inkonsistentes Labeln in Trainingsdatensätzen verursacht werden. Die Kluft zwischen der realen und der simulierten Welt schränkt jedoch häufig die direkte Verwendung synthetischer Daten ein, sodass ein Transfer des Styles, Anpassungstechniken oder Informationen aus der realen Welt erforderlich sind. Hier nutzen wir den mit GRADE erzielbaren Fotorealismus, um synthetische Daten zu generieren und diese Probleme zu überwinden. Da Menschen eine wichtige Quelle für dynamisches Verhalten in Umgebungen und das Ziel vieler Anwendungen sind, konzentrieren wir uns zunächst auf ihre Erkennung und Segmentierung. Wir trainieren Modelle auf realen, synthetischen und gemischten Datensätzen und zeigen, dass die Verwendung ausschließlich synthetischer Daten zu state-of-the-art Leistung in Innenraum-Szenarien führen kann.

Anschließend nutzen wir GRADE, um mehrere Dynamic Visual SLAM-Methoden zu benchmarken. Diese basieren häufig auf semantischer Segmentierung und optischen Flusstechniken, um bewegte Objekte zu identifizieren und ihre visuellen Merkmale von der Pose-Schätzung und Optimierung auszuschließen. Unsere Auswertungen zeigen, dass sie dazu neigen, zu viele Merkmale abzulehnen, was zu Fehlern bei der *genauen* und *vollständigen* Verfolgung von Kameratrajektorien führt. Überraschenderweise beobachteten wir niedrige Verfolgungsraten nicht nur bei simulierten Sequenzen, sondern auch in realen Datensätzen. Darüber hinaus zeigen wir, dass die Leistung der verwendeten Segmentierungs- und Erkennungsmodelle nicht immer positiv mit der der Dynamic Visual SLAM-Methoden korreliert. Diese Fehler sind hauptsächlich auf falsche Schätzungen, überfüllte Szenen und die Nichtberücksichtigung der verschiedenen Bewegungszustände des Objekts zurückzuführen. Um dies zu beheben, führen wir DynaPix ein. Diese dynamische visuelle SLAM-Methode schätzt Bewegungswahrscheinlichkeiten pro Pixel und integriert sie in neue, verbesserte Prozesse zur Posenschätzung und

-optimierung innerhalb des SLAM-Backends, was zu längeren Trackingzeiten und geringeren Flugbahnfehlern führt.

Schließlich verwenden wir GRADE, um die Herausforderung der begrenzten und ungenauen Annotationen von wilden Zebras anzugehen, insbesondere hinsichtlich ihrer Erkennung und Schätzung der Pose, wenn sie von unbemannten Luftfahrzeugen beobachtet werden. Unter Ausnutzung der Flexibilität von GRADE führen wir ZebraPose ein – die erste vollständige Top-Down-Methode zur Erkennung von synthetischen zu realen Objekten und zur 2D-Posen-Schätzung. Im Gegensatz zu früheren Ansätzen zeigt ZebraPose, dass beide Aufgaben nur mit synthetischen Daten durchgeführt werden können, wodurch kostspielige Datenerfassungskampagnen, zeitaufwändige Annotationsverfahren oder Syn-to-Real-Transfertechniken überflüssig werden.

Letztendlich zeigen wir in dieser Arbeit, wie die Kombination von Wahrnehmung und Handeln kritische Limitierungen in der Robotik und der Umgebungswahrnehmung überwinden und letztlich den Einsatz intelligenter und autonomer Systeme für reale Anwendungen vorantreiben kann. Durch Innovationen wie iRotate, GRADE und ZebraPose ebnet sie den Weg für robustere, flexiblere und effizientere intelligente Systeme, die in der Lage sind, sich in dynamischen Umgebungen zurechtzufinden.

# Acknowledgements

> Picture a wave. In the ocean. You can see it, measure it, its height, the way the sunlight refracts when it passes through. And it's there. And you can see it, you know what it is. It's a wave. And then it crashes in the shore and it's gone. But the water is still there. The wave was just a different way for the water to be, for a little while. Just as the ocean, so too does life. There will be times of calm and times of turbulence, transformations, retreats, and beautiful waves, but ultimately, we keep moving forward.
>
> Adapted from *The Good Place, S4E13*

Looking at the last five years, I feel incredibly fortunate to have had in my life the combination of a supportive and flexible work environment, the inspiring atmosphere of the PS department, enriching opportunities, and the companionship of wonderful friends and family. As a first-generation scholar, and remembering the challenges I faced during my undergraduate studies, I am still in disbelief about having reached this point. I remember receiving the news that day in March 2020, and I'll be forever grateful for this opportunity. Surely, starting my PhD during the global COVID-19 pandemic presented unique challenges that demanded resilience and adaptability, and probably made some things harder than necessary. Perhaps also thanks to that, this journey has been transformative, teaching me the value of collaboration, persistence, balance, and reflection. It was a test of perseverance, but one that led to profound growth.

First, I want to thank my advisors, Prof. Aamir Ahmad and Prof. Michael J. Black, for the amazing environments they created. Your guidance has been a cornerstone of my growth, and I am grateful for the freedom you often allowed me — sometimes, perhaps, too much! Your mentorship, insightful guidance, and encouragement throughout this journey meant a lot. I am especially grateful for your support during moments when things didn't work out as planned, for the harsh, beneficial, and constructive feedback, and for the random yet invaluable insights dropped even during small talks in all-hands

meetings or lunches. You have been role models not only professionally but also personally, showing how to navigate challenges, research, and life with integrity, creativity, and determination. For your guidance, all the lessons learned, and the opportunities, I will always be profoundly thankful.

I thank Jörg Stückler for being on my thesis advisory committee and providing valuable feedback about my research projects and journey. I would also like to thank Prof. Hendrik P. A. Lensch and Martin V. Butz for reviewing this thesis and for participating in the oral exam.

I want to thank Nicole, Melanie, and Benjamin, you are truly the backbone of the Perceiving Systems group. Your administrative support, technical expertise, and willingness to help ensured that many hurdles were overcome seamlessly. Thank you for making countless tasks so easily doable.

To my colleagues and friends at the Max Planck Institute, back home, and beyond, thank you for the many moments that enriched this experience—discussions that sparked ideas, and fun moments that lightened the days. Thank you for the countless coffees, movie nights, BBQs, and hikes that added color to this journey. There were many hard moments, but so many nice ones, and your presence made both more meaningful and manageable. We all know that the journey from idea to implementation to publication is a difficult one, and I am especially grateful to all who helped me forget those challenges, even if just for a few minutes while drinking a coffee. Among them, I want to extend my thanks to my co-authors and to all members of the FRPG group with whom I had amazing experiences - especially performing experiments and work trips. Collaborating with you has been enriching, and your contributions have significantly shaped my journey.

I am also grateful for the opportunities that this program has given me to travel, attend conferences, and undertake internships. These experiences broadened my perspective, enriched my knowledge, and deepened my appreciation for the collaborative nature of scientific research.

A special thanks to my family — my mum and dad, Roberta and Luciano, and my big sis, Chiara. Your unconditional love and belief in me, even as I missed birthdays, holidays, and events, because of deadlines, internships, work, or travel. You gave me the strength to keep pushing forward. Your belief in me, unconditional support, love, and comprehension while seeing me abroad and far from home have been the foundation of my success here and made me feel safe, whatever happened. Thank you.

Finally, I will thank my partner, Laura, for your patience, understanding, constant encouragement, and presence in both the challenging and joyful moments. You have been my rock throughout this journey - even if you will never admit that, and you will never believe me on this. Despite the physical distance that often separated us, you made every moment special and made every second we saw each other in these years count. Thank you for your sacrifices and for bearing the time I took away from us, without ever faltering. I hope this will be worth it.

Thank you all. You have made this challenging journey worthwhile, transforming it into an unforgettable experience that will shape me and my life forever.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

## 1.1 Motivation

Humans rely heavily on vision to perceive, understand, and interact with their surroundings. Vision, however, goes beyond passive observation [9]. Our ability to continuously process and learn from visual data while leveraging experiences allows us to act safely and efficiently in diverse environments. With just a glance, we can, for example, easily construct mental maps of spaces, identify objects and people, intuitively shift our focus to what is important, and imagine (future) scenarios we have not yet observed [160]. While other senses can also contribute valuable information, vision uniquely enables us to understand spatial relationships, adapt to changes, and foresee the future.

In many ways, we seek to replicate with machines this ability to sense and interpret the external environment in real-time as it is crucial to enabling robotics applications, autonomous systems, and extended reality (XR) tools [45]. Generally, intelligent systems rely on exteroceptive sensors such as sonar [63, 122], LiDAR [24, 114, 259], and cameras [114, 187, 247] to perceive their surroundings. Cameras stand out for their cost-effectiveness, ability to capture detailed spatial and temporal information, and intuitive feedback for users and developers. This makes them one of the most used sensors for allowing intelligent systems to perceive the world.

Unfortunately, unlike humans, machines natively lack our innate understanding and intuitive perception of the environment and must learn to process and extract information from raw pixels and images. Geometry and computer vision (CV) techniques have traditionally been used to infer the 3D geometry of the scene [76, 95, 152] or estimate the camera motion [214, 217]. Recently, advancements in deep learning (DL) have allowed us, for example, to use images to detect and classify objects automatically in the scene [80, 84, 105, 177]. However, this is often insufficient to enable the interaction between intelligent systems and the environment. A cleaning robot navigating a household [108], an autonomous vehicle making split-second decisions on the road [24], or an XR system enhancing immersive experiences [109], all depend on a continuous cycle of perception, processing, learning, and acting on sensory data, just as humans and animals do [202].

Achieving this level of adaptability in intelligent systems remains a significant challenge in many scenarios. Current methods often lack the flexibility, generalization, contextual understanding, real-time processing, and decision-making capabilities we possess [72, 73, 221, 225]. For example, humans always maintain awareness of their surroundings, *constantly* knowing where they are and what is around them. We do so seamlessly and unconsciously, but it is what ultimately allows us to perform actions even in new, unseen environments. By contrast, intelligent systems often find themselves in new worlds. Therefore, to enable downstream applications and autonomy, they must leverage perception *first* to localize themselves, understand the environment, and recognize places - abilities enabled with Simultaneous Localization and Mapping (SLAM) [28, 162, 223] approaches.

SLAM often provides the groundwork for intelligent systems to operate effectively in

previously unseen environments. Visual SLAM (V-SLAM) approaches [114, 152, 221, 235] use cameras to both build a representation of the world and to be able to understand their location in it. This allows the creation of maps - necessary for enabling tasks such as navigation, object interaction, or others - when no information is available beforehand, as in common household environments. In robotics, SLAM can also be integrated with reasoning and action in Active SLAM methods [40, 116, 157, 240], where the system processes its state and the information received to explore environments autonomously. However, current systems often lack the adaptability and continuous focus-switching that humans intuitively perform, limiting their performance, flexibility, and scalability [156, 162, 183].

Humans also excel at using vision to predict the dynamics of their surroundings, effortlessly adapting to changes without necessarily compromising their capabilities [61, 160]. In contrast, SLAM systems often struggle when deployed in dynamic environments [14, 17, 24, 191]. This inability is a significant bottleneck in advancing and broad adoption of intelligent systems, as it constrains both their effectiveness and robustness. The problem is the result of practical and technological barriers. Testing physical robots in dynamic environments is dangerous and resource-intensive due to their limited control and decision-making capabilities. Simulators, instead, while offering a safer alternative, frequently fall short in replicating the realism and variability of actual environments [43, 44]. Moreover, despite an extensive history, V-SLAM approaches fail to capture the different levels of dynamism that an object can show [14, 235, 267].

A further complication arises from the inability of intelligent systems to learn to generalize visual information in the same way humans do. We can make associations effortlessly, recognizing patterns and adapting to new contexts based on prior experience. By contrast, current deep learning models rely heavily on vast, labeled datasets [59, 129, 213, 248]. While these datasets have driven remarkable progress, they are expensive, time-consuming, and sometimes impractical to create. Furthermore, systems trained on narrowly defined datasets often struggle to generalize, as their performance degrades when faced with the diversity and complexity of real-world tasks [119, 187, 224]. This gap underscores the need for innovative approaches that enable intelligent systems to specialize or generalize effectively.

Overcoming these challenges would unlock applications for intelligent systems, such as autonomous navigation in complex environments, assistive technologies for caregiving, conservation efforts in natural habitats, and enhanced immersion in AR/VR/XR applications. This thesis addresses these challenges by proposing innovative solutions that bridge perception and action, enabling intelligent systems to reason, learn, and adapt in unknown or dynamic environments. By leveraging insights from human perception and integrating them into system design, the contributions presented in this work aim to push the boundaries of intelligent system capabilities.

## 1.2 Summary of Content

Vision is fundamental to how humans and animals perceive, understand, and interact with the world. It enables continuous processing, learning, foreshadowing, and adaptive behaviors in diverse, unseen, and dynamic environments. Inspired by these capabilities, this thesis focuses on leveraging visual perception to bridge sensing with understanding and action to address critical challenges that limit intelligent systems from behaving in human-like ways in our world. Specifically, narrow and static datasets, resource-intensive data collection and evaluations, limited simulation tools, and fixed, inflexible objectives embedded in some applications are all aspects that severely constrain our ability to obtain intelligent systems capable of generalizing and adapting. To address these challenges, we target four main tasks. First, we aim to autonomously explore and map unknown environments, building on graph-based Visual SLAM methods (introduced in Section 1.3.3) while promoting more human-like behavior rather than narrowly focused objectives. Second, we develop a flexible and controllable simulation system designed to reduce the sim-to-real gap while supporting robotics applications. Third, we enhance camera state estimation in dynamic environments, improving system robustness by leveraging the continuously varying motions of objects. Finally, we explore the use of synthetic data to learn real-world information, bypassing the challenges of costly and impractical data collection.

**Human-Like Autonomous Exploration** In Chapter 2, we address the challenge of enabling robotics systems to explore and map unknown environments autonomously using visual data in scenarios where prior information is unavailable. This problem, known as Active V-SLAM [240], is fundamental for numerous real-world applications, particularly in environments where human intervention is impractical, hazardous, or inefficient. For instance, in disaster response scenarios autonomous exploration can help navigate unstable or dangerous areas without risking human lives [58]. In industrial settings, instead, such as building and facilities inspection or deep-sea exploration, Active V-SLAM enables robots to operate in conditions where sending human workers could be costly, unsafe, or impractical [29, 232]. Additionally, for home-centered applications like robotic assistants or automated cleaning systems, the ability to autonomously explore and map an environment efficiently is essential for scalability, reducing the need for human supervision or specialized setup [7]. Despite a long-lasting history, existing exploration methods often adopt only 'simple' approaches, optimizing either short-term or long-term objectives, neglecting the evolving nature of the exploration process itself [162]. This way of dealing with the problem hinders efficiency, adaptability, and real-world applicability, even assuming a static environment without moving entities. State-of-the-art approaches targeted either the closest frontier[1] or the one that maximized the information gain *but* without considering the exploration process [47]. Other methods only focused on the immediate vicinity of the robot, without considering the overall long-term ob-

[1] The boundary between explored and unexplored areas.

jective [19]. Instead, unlike conventional methods that treat these aspects in isolation, our approach continuously predicts information gain and dynamically optimizes camera viewpoints to maximize exploration efficiency and localization accuracy. The intuition is that humans do not navigate by following rigid, pre-planned strategies, and do not always look in the same, fixed direction. Instead, with iRotate we continuously adjust the robot's heading, shifting the visual focus and refining our actions based on immediate surroundings and long-term goals. iRotate is a multi-layered approach built on top of rhe RTAB-Map [114] framework. The top layer predicts the overall information gain across different paths leading to frontier points, ensuring a fully informed decision. The second layer refines the orientation of the camera along these paths, accounting for variability in information due to robot movements, evolving map optimizations, and potential errors in trajectory execution. Finally, the third layer leverages local feature information in real-time to enhance the robustness of the state estimate. This ensures that as the robot moves, it maintains an accurate understanding of its environment, effectively integrating long-term exploration strategies with short-term adaptability. Initially, our method is specifically designed for omnidirectional robots, leveraging their independent control over translation and rotation. However, this would limit the integration of the approach into several systems. Therefore, we introduce an additional degree of freedom by enabling independent camera rotation, allowing the robot to optimize its viewpoint without altering its trajectory. While this enhances visual information acquisition, it also creates additional estimation uncertainties, as it is not possible to know the exact relative orientation of the two without costly specialized sensors. To address this, we extend the robot's state estimation mechanism to include the camera state, jointly modeling their uncertainties to maintain accurate localization and mapping. Through extensive simulations and real-world robot experiments, we demonstrate that iRotate achieves comparable exploration coverage to state-of-the-art methods while significantly reducing overall map entropy. Notably, our approach reduces the total distance traversed by up to 39% compared to existing techniques, without increasing the total rotation of the robot's wheels, while traveling shorter paths. By unifying real-time adaptability with long-term exploration strategies, iRotate represents a significant advancement in Active Visual SLAM. It enhances efficiency and accuracy and lays the groundwork for more flexible and intelligent autonomous systems.

**Photorealistic Simulations: Bridging Virtual and Real Worlds** In Chapter 2 we assumed that the robot was working inside a static environment. While convenient and commonly adopted, this assumption fails to reflect real-world complexities. Dynamic elements, e.g. moving people, disrupt core V-SLAM mechanisms such as state estimation and map building, thus limiting their robustness and adoption. This assumption is often the result of several concurring factors. Testing robots in real-world scenarios provides clear and valuable insights, but it is often costly, risky, and impractical, especially when dynamic components in the scene might collide with the autonomous robot [211, 235, 243]. Additionally, real-world tests cannot be precisely repeated. To avoid these risks, researchers turn to datasets and simulations. However, datasets are con-

strained by fixed sensors, offer no interaction, and cannot support the continuous reasoning and changes brought by autonomous methods. Moreover, synthetic datasets (often tailored for CV applications [187]) frequently feature unrealistic backgrounds [59], making them less applicable to robotics. At the same time, traditional simulation tools lack either flexibility, low-level control, or support of realistic physics, rendering technologies, diverse robotic systems, or dynamic entities. All critical elements for developing robust autonomous systems. To address these limitations, Chapter 3 introduces a photorealistic simulation framework for Generating Realistic And Dynamic Environments - GRADE. GRADE is designed to simulate robots and collect data in dynamic environments that closely mimic real-world conditions. Built on NVIDIA Isaac Sim, GRADE leverages its path-tracing and physics engines for high-fidelity rendering and realistic interactions. GRADE extends Isaac Sim's capabilities by providing extensive tools to enhance simulation control, seamlessly load and command robots, and generate high-quality ground-truth synthetic data in various scenarios. A key innovation of GRADE is its experiment repetition mechanism, which allows for controlled variations of previously run simulations while preserving physics consistency. This feature facilitates robust benchmarking and iterative testing across diverse scenarios, making it invaluable for advancing autonomous systems. In this chapter, we also show that it is possible to use the data generated with GRADE alone to address the syn-to-real gap for the detection and segmentation of humans in indoor environments. With its scalability, interactivity, and high fidelity, GRADE accelerates the development of intelligent systems and lays the groundwork for the advancements discussed in subsequent chapters.

**On Dynamic Visual SLAM: Benchmarking, Challenges, and Pixels** Building on GRADE, Chapter 4 addresses a critical challenge in Visual SLAM: the inability of traditional systems to operate effectively in dynamic environments [14, 17, 191]. While V-SLAM approaches have achieved significant success in static settings, their performance deteriorates in the presence of moving objects or changing conditions. V-SLAM methods rely on visual features, i.e. distinct regions or points that can be extracted from the image, by matching them across frames to estimate camera motion. However, when these are dynamic, they lead to optimization errors and unreliable camera trajectory estimations. The classical way to address this issue involves feature rejection mechanisms typically based on optical flow [83] or object detection and segmentation methods [17, 131]. Dynamic V-SLAM approaches are then often evaluated only on a limited number of datasets that include dynamic elements. Therefore, we leverage GRADE and evaluate multiple state-of-the-art V-SLAM methods in simulated dynamic environments to establish a comprehensive benchmark. This highlighted the inability of these approaches to generalize across diverse environments with low trajectory errors and high tracking rates, a metric rarely reported in the literature. Interestingly, we observe low tracking times in real-world sequences as well. We believe these issues are a result of several factors. First, class-based approaches overly reject features in crowded environments and focus only on pre-determined classes. Moreover, hallucinations or noisy estimations can make these methods fail. To confirm this hypothesis, we use the models trained in the previous

section to re-evaluate the performance of the chosen Dynamic V-SLAM methods. With this, we show how even the best-performing trained methods do not necessarily yield the best SLAM metrics. Overall, these methods indiscriminately reject features without considering their actual moving state. The intuition lies in the fact that humans, as opposed to state-of-the-art Dynamic V-SLAM systems, can intuitively distinguish between static, moving, and potentially movable objects, continuously accounting for their varying motion dynamics. For instance, a parked car provides stable features that can be used to estimate camera motion and should probably not be rejected by a SLAM system. Inspired by this, we introduce DynaPix, a novel method that integrates per-pixel motion probabilities into the ORB-SLAM2 [152] SLAM pipeline. By estimating the likelihood of motion at each pixel, DynaPix enables more robust pose estimation and trajectory optimization, even in highly dynamic environments. This pixel-level granularity allows the system to selectively retain useful features, such as those from static or slow-moving objects, while ignoring those likely to cause inconsistencies. The experimental evaluation of DynaPix on both synthetic and real-world benchmarks demonstrates its superior performance over existing state-of-the-art methods, particularly in crowded and dynamic environments by significantly extending tracking time and reducing trajectory errors.

**Finding and Posing Zebras from UAVs using Synthetic Data** Chapter 5 builds upon insights from previous chapters, focusing on the challenges of collecting and annotating large datasets for perception tasks such as detection, segmentation, and pose estimation, particularly in dynamic or out-of-distribution scenarios. Traditional approaches rely heavily on manually labeled real-world data, which is often prohibitively costly, time-intensive, and impractical for many applications, especially in rare or complex environments. While synthetic datasets are increasingly employed to address these challenges, they frequently suffer from unrealistic renderings or hallucinated backgrounds, necessitating effective synthetic-to-real (syn-to-real) transfer techniques [59, 100, 119]. An example of such challenges is the detection and pose estimation of wild animals like zebras. These tasks face significant obstacles due to the scarcity of annotated data and the logistical difficulties of collecting it. Moreover, wildlife conservation efforts demand minimally invasive methods to avoid disturbing animals, making unmanned aerial vehicles (UAVs) a preferred tool for data collection and monitoring efforts. However, the aerial perspective introduces unique challenges, as it differs significantly from the ground-level viewpoints common in existing datasets. A consequence of this is that available detection methods fail to work in this scenario due to the different data distribution. While manual annotation is possible, it is error-prone, time-consuming, and impractical for large-scale applications. To address these obstacles, we leverage the capabilities of the GRADE framework by introducing ZebraPose, a novel synthetic-to-real framework designed for top-down zebra detection and 2D pose estimation. Leveraging YOLOv5 [102] and ViTPose [248], ZebraPose achieves accurate detection and pose estimation of zebras from aerial imagery, offering a groundbreaking solution for wildlife monitoring. Notably, ZebraPose is the first framework to perform both top-down detection and pose estimation using only synthetic data, without relying on fine-tuning or style

transfer techniques.

In summary, this research introduces key innovations to push intelligent systems closer to the robust, flexible, and intuitive performance seen in human perception-action systems. The development of iRotate provides a framework for efficient exploration by balancing long-term, short-term, and real-time objectives, enhancing the autonomous exploration of unknown environments. GRADE addresses the limitations of existing simulation tools by offering a photorealistic and customizable framework for dynamic scene generation, enabling realistic testing and synthetic data creation. The DynaPix method improves Dynamic Visual SLAM performance by incorporating per-pixel motion probabilities, leading to more accurate and reliable tracking in dynamic settings. Finally, ZebraPose demonstrates the power of synthetic data in wildlife monitoring, achieving high-quality zebra pose estimation using only synthetic data, and reducing the need for costly and difficult real-world data collection. Together, these contributions advance our understanding of intelligent system design and pave the way for more adaptable, efficient, and scalable solutions in robotics, autonomous vehicles, and environmental monitoring.

### 1.2.1 List of Publications

The contributions in this thesis mainly comprise work from the following publications.

#### Journal Papers

- Bonetto, E., Xu, C., and Ahmad, A. (2025). GRADE: Generating Realistic And Dynamic Environments for Robotics Research. In *Sage International Journal of Robotics Research (IJRR)*;
- Bonetto, E., Goldschmid, P., Pabst, M., Black, M. J., and Ahmad, A. (2022). iRotate: Active Visual SLAM for Omnidirectional Robots. In *Elsevier Robotics and Autonomous Systems (RAS), Volume 154*;

#### Conference Papers

- Bonetto, E., and Ahmad, A. (2026). ZebraPose: Zebra Detection and Pose Estimation using *only* Synthetic Data. In *IEEE/CVF Winter Conference on Applications of Computer Vision 2026 (WACV)*.
- Xu, C.$^\star$, Bonetto, E.$^\star$, and Ahmad, A. (2024). DynaPix SLAM: A Pixel-Based Dynamic SLAM Approach. The first two authors contributed equally to this work. In *2024 German Conference on Pattern Recognition (GCPR)*;
- Bonetto, E., and Ahmad, A. (2023). Synthetic Data-based Detection of Zebras in Drone Imagery. In *2023 European Conference on Mobile Robots (ECMR)*;

**Workshop and Extended Abstracts**

- Ahmad, A., Price, E., Goldschmid, P., Bonetto, E., Liu, Y., Khandelwal, P., Kerekes, V., Csoban, P. and Rubenstein, D. (2025). WildCap: Autonomous Non-Invasive Monitoring of Animal Behavior and Motion. In *1st German Robotics Conference 2025*.

- Bonetto, E., Xu, C., and Ahmad, A. (2023). Simulation of Dynamic Environments for SLAM. In *2023 IEEE International Conference on Robotics and Automation (ICRA) — Workshop: Active Methods in Autonomous Navigation*;

- Bonetto, E., Xu, C., and Ahmad, A. (2023). Learning from synthetic data generated with GRADE. In *2023 IEEE International Conference on Robotics and Automation (ICRA) — Workshop: Pretraining for Robotics*;

- Bonetto, E., Goldschmid, P., Black, M. J., Ahmad, A.(2021). Active Visual SLAM with Independently Rotating Camera. In *2021 European Conference on Mobile Robots (ECMR)*.

- Bonetto, E., and Ahmad, A. (2021) Towards Active Visual SLAM. In *2021 Deutsche Gesellschaft für Robotik (DGR days)*.

The following research was also conducted during the PhD, but is not covered by this thesis:

- Saini, N., Bonetto, E., Price, E., Ahmad, A., Black, M. J. (2022). AirPose: Multi-View Fusion Network for Aerial 3D Human Pose and Shape Estimation. In *Robotics and Automation Letters (RA-L), vol. 7, no. 2, pp. 4805-4812*;

- Tallamraju, R., Saini, N., Bonetto, E., Pabst, M., Liu, Y. T., Black, M. J., Ahmad, A. (2020). AirCapRL: Autonomous Aerial Human Motion Capture using Deep Reinforcement Learning. In *IEEE Robotics and Automation Letters (RA-L), vol. 5, no. 4, pp. 6678-6685*.

## 1.3 Background

This section provides a concise overview of the fundamental concepts that underpin the methods developed in this thesis. First, we briefly discuss the evolution of Simultaneous Localization and Mapping (SLAM). We then cover essential topics in 2D and 3D vision, visual odometry, and graph-based SLAM.

### 1.3.1 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) refers to the problem of enabling a system to build a map of an unknown environment while simultaneously estimating its position within that map [206]. Early approaches to it primarily relied on filtering-based methods, such as the Extended Kalman Filter (EKF-SLAM [205, 222]) and particle filters (FastSLAM [148]), where the robot's state and the map are represented probabilistically, and updates are performed incrementally as new observations arrive. However, these formulations often struggle with computational complexity. In the case of EKF-SLAM, for example, the covariance matrix of all landmarks grows quadratically, making real-time operation infeasible in large-scale environments [75, 209]. Particle filter-based methods represent the robot's belief as a set of sampled hypotheses, but remain sensitive to dimensionality and often require numerous particles to maintain accuracy [48, 202]. Moreover, due to their incremental nature, the integration of delayed or out-of-sequence measurements can be problematic, allowing them to (generally) estimate only the current robot pose but not revisit and refine the previous ones, a process known as smoothing [174, 222]. To solve these issues, graph-based formulations were introduced. As explained in detail in Section 1.3.3, graph-based SLAM reframes the problem as an optimization and smoothing task rather than a sequential filtering process. However, while this allows for both incremental updates and smoothing of the recent poses, optimizing the whole graph in large scenes can be computationally expensive as all the edges in the graph have to be updated. Different optimization techniques and approaches have been developed to mitigate this, such as differentiating between the local and global optimization steps [25, 152]. This, combined with the lower overall errors achieved through the smoothing process, has made this technique a de facto standard.

As previously mentioned, various exteroceptive sensors can be used to perceive the environment and construct a map representation. Among them, cameras stand out as one of the most cost-effective and widely available options [50, 153]. They allow us to estimate motion and reconstruct the environment using only image data via Visual SLAM (V-SLAM) approaches. V-SLAM methods are generally divided into feature-based and direct [1]. Feature-based approaches detect and track distinctive keypoints, or features, across frames, using them as landmarks for motion estimation and place recognition. These methods are computationally efficient and robust to illumination changes but may struggle in textureless or dynamic environments [1,5]. In contrast, direct methods operate on all the raw pixels by leveraging photometric consistencies. While direct methods can perform well in low-texture regions, they are typically more sensitive to lighting variations and computationally demanding [158]. Considering all these factors, graph-based V-SLAM methods have become the most widely used SLAM approaches today.

### 1.3.2 2D and 3D Vision

Since camera data plays a central role in V-SLAM, a solid understanding of 2D and 3D vision is essential for developing accurate and robust visual perception systems. At the core of any vision system is the 2D image — a matrix of pixel intensities representing the scene as it is observed from a specific viewpoint. However, the world is inherently three-dimensional, and understanding the geometry and imaging process allows us to comprehend how the 3D structure of a scene falling within the field of view (FOV) of the camera is then projected onto a 2D image plane [76, 217].

Different camera models can describe how this projection works. The pinhole camera model is one of the most common and the one we will use throughout this thesis. In this model, a 3D point $\mathbf{X}_w = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}^\top$ in a global coordinate system is mapped to a pixel $\mathbf{p} = \begin{bmatrix} u & v \end{bmatrix}^\top$ through a perspective projection. This is governed by the camera's intrinsic and extrinsic parameters, which define the relationship between the camera, the world, and the captured image. For simplicity, here we assume a distortion-free model.

The intrinsic parameters characterize the internal properties of the camera and are encapsulated in the calibration matrix $\mathbf{K}$. This matrix transforms a point in camera coordinates into normalized image coordinates, up to a scale factor. Its parameters include:

- **Focal Length** $(f_x, f_y)$: Defines the scale of projection along the image axes and is typically expressed in pixels.
- **Principal Point** $(c_x, c_y)$: The image coordinates where the optical axis intersects the image plane.
- **Skew Coefficient** $(s)$: Represents the angle between the image axes (often assumed to be zero in modern cameras).

The intrinsic matrix is commonly represented as:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.1}$$

The extrinsic parameters, on the other hand, define the position and orientation of the camera relative to the global coordinate system. These consist of:

- **Rotation Matrix** $(\mathbf{R})$: A $3 \times 3$ orthonormal matrix that defines the orientation of the camera relative to the global coordinate system. It satisfies the property $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$, ensuring that rotations preserve distances.
- **Translation Vector** $(\mathbf{t})$: A $3 \times 1$ vector that specifies the camera's position in the world coordinate system.

Together, the extrinsic parameters form the extrinsic transformation matrix, which is used to project a point from the world coordinate system to the camera coordinate system through:

$$\mathbf{X}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R}\mathbf{X}_w + \mathbf{t} \tag{1.2}$$

where $\mathbf{X}_c$ is the representation of $\mathbf{X}_w$ transformed in the camera frame.

We then use

$$\mathbf{p}' = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \mathbf{K}\mathbf{X}_c \tag{1.3}$$

to project camera frame coordinates $\mathbf{X}_c$ onto the image plane. By using homogeneous coordinates $\tilde{\mathbf{P}}_w = \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$, we can then write a unified equation:

$$\mathbf{p}' = \mathbf{K}[\mathbf{R}|\mathbf{t}]\tilde{\mathbf{P}}_w. \tag{1.4}$$

The final image coordinates $\mathbf{p}$ are obtained by normalizing $\mathbf{p}'$ via:

$$u = \frac{u'}{w'}, \quad v = \frac{v'}{w'}. \tag{1.5}$$

Often, the whole process is indicated through:

$$\pi(\mathbf{C}, \mathbf{X}_w) = \mathbf{p} \tag{1.6}$$

where $\mathbf{C}$ is the camera pose and $\mathbf{X}_w$ the 3D point we are projecting to $\mathbf{p} = [u, v]$.

This perspective projection model is central to computer vision and SLAM, as it defines how images encode depth and structure information. In SLAM, this can be used to estimate both the camera motion, i.e. the evolution of the extrinsic parameters with respect to an origin frame, and to infer the depth of the scene through the triangulation of corresponding points between frames. However, trying to estimate both at the same time using only images allows us to do so only up to an unknown scale factor.

Various techniques can be employed to resolve this ambiguity. Stereo cameras, for instance, provide depth information by leveraging a known metric baseline between two viewpoints, allowing depth estimation through disparity computation. Similarly, an Inertial Measurement Unit (IMU) can provide metric scale information when integrated into the system. Alternatively, depth sensors such as infrared-based structured light cameras or LiDAR can directly capture depth measurements, enabling metric reconstructions.

### Visual Odometry

Visual Odometry (VO) is responsible for estimating the motion of the camera between frames. The process typically involves identifying points of interest and tracking them across frames. Such points are called features. To achieve robust feature tracking, various keypoint detectors and descriptors are employed, including ORB (Oriented FAST and Rotated BRIEF) [185], SIFT (Scale-Invariant Feature Transform) [139], and SURF (Speeded-Up Robust Features) [11]. ORB is widely used due to its efficiency and robustness to illumination changes, while SIFT and SURF offer scale and rotation invariance at the cost of higher computational complexity. Once features are detected, they are matched across frames using appropriate distance metrics. These matched correspondences enable the estimation of camera motion by solving epipolar constraints.

The transformation from a 3D point $\mathbf{X}$ and its 2D projection $\mathbf{p}$ on the image frame is summarized by Equation (1.6). Given two consecutive frames, the geometric relationship between corresponding image points is governed by the fundamental matrix [65] $\mathbf{F}$ (for an uncalibrated camera) or the essential matrix [137] $\mathbf{E}$ (when intrinsic camera parameters are known), satisfying the epipolar constraint:

$$\mathbf{p}_2^T \mathbf{E} \mathbf{p}_1 = 0. \tag{1.7}$$

By decomposing $\mathbf{E}$ using singular value decomposition (SVD), the relative camera motion, expressed as rotation $\mathbf{R}$ and translation $\mathbf{t}$, can be recovered up to scale. Other techniques, such as homography estimation, can be used as an alternative to epipolar geometry, especially in planar scenes where the latter often fails.

Despite its effectiveness, Visual Odometry (VO) suffers from drift accumulation over time, which leads to trajectory estimation errors. Furthermore, feature correspondences become ambiguous when objects in the environment move or change, causing mismatches and additional inaccuracies in the estimated trajectory.

## 1.3.3 Graph-based V-SLAM

In the context of SLAM, we assume the system operates in an unknown environment. In general, its trajectory up to the current time $t$ is represented as:

$$\mathbf{x}_{1:t} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\} \tag{1.8}$$

where $\mathbf{x}_i$ denotes the system's state at time $i$. The state typically includes the position and velocity of the system. For simplicity, the initial state $\mathbf{x}_0$ is often assumed to be at the origin, i.e.

$$\mathbf{x}_0 = (0,0,0), \tag{1.9}$$

which serves as a reference frame for subsequent motion estimates. As the system moves, it acquires odometry measurements:

$$\mathbf{u}_{1:t} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_t\} \tag{1.10}$$

and perceives the environment through observations:

$$\mathbf{z}_{1:t} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t\}. \tag{1.11}$$

Using these inputs, the system incrementally builds a representation of the environment, denoted as $\mathbf{q}$. The SLAM problem then consists of jointly estimating both the system's trajectory and the environment's representation, given the acquired measurements:

$$p(\mathbf{x}_{1:t}, \mathbf{q} \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t}). \tag{1.12}$$

V-SLAM methods primarily rely on images to perceive the environment and build a map representation, but not necessarily to acquire odometry measurements. The odometry step often integrates multiple information sources, such as GPS, IMU, and VO, typically fused through EKF [202, 222]. However, some systems operate solely on VO. The map in V-SLAM can take various forms, depending on the system's needs and the environment being mapped. It may be a dense representation, which captures detailed spatial information, or just the set of features and landmarks, e.g. in SfM maps. Alternatively, other forms like octomaps [88] or gridmaps [60] can be used, where the environment is represented by cells, each containing information like the presence or absence of objects. These different types of maps offer varying trade-offs between computational complexity, memory usage, and accuracy, allowing V-SLAM systems to adapt to different scenarios [264]. In this thesis, we will mostly employ 2D gridmaps. Let $M \subset \mathbb{R}^2$ represent a bounded 2D grid map of the environment, where for each cell $\mathbf{m} = [x_m\ y_m]^\top \in M$ we have an occupancy probability $p_o(\mathbf{m}) \in [0,1]$. $M$ can be then subdivided in *unknown* space, $M_{unk} \subset M$, and explored area, $M_{exp}$, given that $M_{unk} \cup M_{exp} = M$ and $M_{unk} \cap M_{exp} = \varnothing$. The cells in the explored area can then be classified as *free* ($M_{free}$) or *occupied* ($M_{occ}$) under the condition that $M_{free} \cup M_{occ} = M_{exp}$, and set $p_o(\mathbf{m}) = 0.5$ if $\mathbf{m} \in M_{unk}$. The SLAM framework will process the input data and transition cells from the unknown to the explored sets while updating occupancy probabilities to every cell belonging to $M_{exp}$. Some cells of this map are then further identified as frontiers, i.e. cells located at the boundaries between *free* and *unknown* spaces large enough for the robot to traverse. These definitions can be expanded, without loss of generality, to 3D approaches.

### Factor Graphs Optimization

Graph-based V-SLAM formulates the SLAM problem as a factor graph, where the system maintains a set of camera poses and feature positions while optimizing their consistency given available sensor constraints [202, 222]. The SLAM system constructs a factor

graph, where nodes correspond to camera poses and perception points. The edges in this graph encode spatial constraints derived from sensor measurements, such as odometry and feature-based correspondences. These constraints are modeled as probability distributions that define the relationships between connected nodes. In this way, the system can leverage Markov assumptions and employ graph-based optimization techniques.

Optimization in SLAM is further complicated by the fact that many variables of interest do not live in Euclidean vector spaces but rather on differentiable manifolds. For example, robot poses and landmark orientations are typically represented as elements of the Lie groups SE(2) or SE(3). To handle this, modern SLAM formulations make use of manifold optimization, employing minimal local parameterizations and retractions to map between the manifold and its tangent space [87]. This allows nonlinear least-squares solvers such as Gauss–Newton or Levenberg–Marquardt to be applied consistently to problems involving rotations and rigid-body transformations. Frameworks such as g2o [113] and Ceres Solver [3] provide generic mechanisms for optimization on manifolds. In practice, the optimization problem is formulated as a nonlinear least squares problem, where the objective is to find the configuration of poses that minimizes the accumulated error. This approach, commonly referred to as pose graph optimization, primarily refines the trajectory and corrects the accumulated drift. If desired, this approach can be used to optimize other observations as well, like planes, cuboids, or features. Modern solvers such as `g2o` [113] and Ceres [3] have been widely adopted to optimize these graphs efficiently, making large-scale mapping feasible. The optimization problem is commonly defined as:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} C(\mathbf{x}), \tag{1.13}$$

where $C(\mathbf{x})$ represents the cost function that quantifies the inconsistency between the estimated states and the sensor constraints. In the case of vision-based SLAM, this error is often defined in terms of the *reprojection error*, which measures the discrepancy between observed and predicted feature projections and is optimized through a Bundle Adjustment (BA) algorithm. The cost function to be minimized is then:

$$\begin{aligned} C &= \sum_{i,j} \rho\left((\mathbf{p}_{ij} - \pi(\mathbf{x}_i, \mathbf{X}_j))^T \Omega_{i,j}^{-1} (\mathbf{p}_{ij} - \pi(\mathbf{x}_i, \mathbf{X}_j))\right) \\ &= \sum_{i,j} \rho(\mathbf{e}_{i,j}^T \Omega_{i,j}^{-1} \mathbf{e}_{i,j}), \end{aligned} \tag{1.14}$$

where $\mathbf{e}_{i,j} = (\mathbf{p}_{ij} - \pi(\mathbf{x}_i, \mathbf{X}_j))$ is the reprojection error. Here, $\rho$ is the cost function, e.g. the Huber robust cost function, $\mathbf{p}_{ij}$ is the observed 2D projection of the 3D point $\mathbf{X}_j$ in the $i$-th frame, and $\Omega$ is the covariance associated to the measurement. Note, however, how dynamic scenes will also impact this step of a V-SLAM approach, in addition to the aforementioned VO procedure. This is because the reprojection error might no longer be computed over immovable features observed from multiple views.

Graph-based optimization in V-SLAM typically combines local and global strategies to balance computational efficiency and accuracy. Local optimization refines the most recent camera poses in real-time to ensure consistency within a short temporal window. Meanwhile, global optimization is triggered by specific events, such as loop closures, which correct long-term drift and enforce global consistency. Local optimization thus allows for on-line and real-time operations, while the system is not always blocked by the global optimization step, which can be slow. Other techniques, like sub-map merging, can also be applied to alleviate the computational complexity.

### Loop Closure Detection

Loop closure events happen whenever the system recognizes in the currently sensed data a previously visited location. Their detection plays a crucial role in mitigating drift as the framework can use these events for enforcing consistency in the estimated trajectory by matching the current pose with the old one and using that as a strong prior. In visual SLAM they are often detected by leveraging visual features. To efficiently match visual features across images and detect revisited scenes *Bag-of-Words* (BoW) [46] models are commonly used. Once a loop closure is identified, global optimization is performed to redistribute errors across the pose graph, significantly improving overall accuracy. However, loop closures must be validated carefully, as incorrect matches can introduce severe inconsistencies. To prevent erroneous updates, many SLAM systems reject loop closures if the correction introduces excessive deviation from the current graph structure. Clearly, moving objects and features might prevent loop closure from happening, as the content of the frame will no longer match what was observed before.

### Performance Metrics

Naturally, various evaluation metrics are employed to assess the accuracy and performance of V-SLAM systems, ranging from trajectory estimation accuracy to map reconstruction quality and robustness analysis. These metrics provide quantitative insights into the effectiveness of a SLAM system under different conditions. One of the most commonly used trajectory evaluation metrics is the *Absolute Trajectory Error* (ATE), which measures the global accuracy of the estimated trajectory by comparing it against the ground truth. Given $N$ estimated camera (robot) poses $\hat{\mathbf{t}}_i$ and their corresponding ground truth poses $\mathbf{t}_i$, ATE Root Mean Squared Error (RMSE) is computed as:

$$\text{ATE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| \mathbf{t}_i - \hat{\mathbf{t}}_i \right\|^2}. \tag{1.15}$$

A lower ATE value indicates a more accurate trajectory reconstruction and can also be used to evaluate VO methods. Beyond trajectory accuracy, the quality of the reconstructed map is another critical evaluation factor. The map completeness, for exam-

ple, quantifies the proportion of the environment reconstructed and is often considered alongside the normalized map entropy and balanced accuracy (BAC). Indeed, for each explored location of a map $\mathbf{m} \in M$ obtained from a SLAM system (be that in 2D or 3D), we can know if that cell is unexplored or occupied/free (and with which probability). If we indicate this probability with $p_o$ we can then define the cell entropy as:

$$H[\mathbf{m}] = -(p_o(\mathbf{m}) \cdot \log_2(p_o(\mathbf{m})) + (1 - p_o(\mathbf{m})) \cdot \log_2(1 - p_o(\mathbf{m}))). \tag{1.16}$$

The entropy of the map is then defined as

$$E_{\text{map}} = \sum_{\mathbf{m} \in M_{exp}} H[\mathbf{m}], \tag{1.17}$$

and the normalized entropy as

$$\overline{E_{\text{map}}} = \frac{E_{\text{map}}}{|M_{exp}|}. \tag{1.18}$$

Naturally, as this is a metric of the final output, it is computed over the explored area. Higher entropy values signify higher uncertainty in the reconstruction. Moreover, note that this can be restricted without loss of generality to any arbitrary number of cells, e.g. to compute the entropy of the observed area of the map. The balanced accuracy is instead defined as the average of recall obtained on each class, i.e. occupied and free cells, and is used to account for the imbalance between these two sets.

# Chapter 2

# Human-like Autonomous Exploration

Active SLAM (A-SLAM) methods enable robots to simultaneously map and self-localize in unknown environments by actively choosing actions that maximize information gain. However, unlike humans, who dynamically shift their focus based on environmental changes and task priorities, these methods fail to adapt their perception strategy throughout the exploration process, resulting in inefficient information gathering. As previously mentioned in Sections 1.1 and 1.2, state-of-the-art methods suffer from a critical limitation: they fail to consider the entirety of the exploration process and all actions that will be or are being taken by the robot while moving toward the goal. By focusing only on specific locations and time instants, while neglecting the information that would be available *throughout* the process or its changing nature, they fail to maximize and optimize information gain over the whole experiment's duration. In practice, they often overlook the continuous nature of exploration, leading to suboptimal planning and inefficient use of sensor data.

To solve this, we present a novel Active SLAM method, iRotate, which leverages vision-based perception for efficient autonomous exploration. iRotate introduces a multi-layered active control strategy that *continuously* adjusts camera orientation to enhance information gain (Section 2.5). The top layer selects informative goal locations and pre-computes highly informative view directions. The subsequent layers instead actively re-plan and refine the previously made choices while executing the trajectory itself, exploiting the continuously updated map and local feature information. Our approach further accounts for the presence of obstacles in the field of view and the robot's own location with respect to the goal through dedicated utility formulations.

Initially designed for omnidirectional robots, our method leverages their independent translation and rotation control. Thus, we also introduce a novel independent rotational joint to allow a simultaneous and decoupled rotational movement of the robot and the camera. This also required a new state estimation technique (Section 2.4.3) to jointly model the robot and camera states toward mitigating additional estimation uncertainties. This enables optimal heading control *without* requiring the robot's base rotation, extending our method to non-holonomic robots.

Through extensive simulations and real-world experiments, we demonstrate that iRotate achieves comparable coverage to state-of-the-art methods while reducing map entropy and energy consumption. Notably, it shortens the total traversed distance by up to 39% without increasing the wheels' total rotational movement, despite the continuous orientation changes. Furthermore, we show that our method generalizes effectively to non-omnidirectional robots, expanding its applicability.

The chapter is structured as follows: we introduce the problem in Section 2.1, followed by a discussion of the state-of-the-art in Section 2.2. The problem formulation and notations are presented in Section 2.3. We then cover kinematic models and state estimation techniques in Section 2.4 and our iRotate framework in Section 2.5. Finally, we present evaluation experiments in simulated and real world environments in Section 2.6 and conclude in Section 2.7.

## 2.1 Active Visual SLAM

Mobile robots assisting humans in everyday tasks are increasingly present in workplaces and homes. Examples include autonomous vacuum cleaners, lawnmowers, warehouse transporters, and automated inspection or patrolling systems. These robots often operate in new, unknown, and possibly changing environments, requiring them to both *perceive* and *adapt* to their surroundings. To ensure robust operation in higher-level tasks like navigation, obstacle avoidance, and environment interaction, they generally must possess two key capabilities: i) self-localization and ii) real time environment mapping [202, 222, 225]. While localization can rely on a prebuilt map, a robot encountering an environment for the first time and without access to any prior information must perform both tasks simultaneously using SLAM (Section 1.3.3). Moreover, once deployed, robots often *still* need to perform SLAM continuously to account for environmental changes, refine maps, and enhance localization robustness.

As discussed in Section 1.3, a widely adopted approach is V-SLAM [28], where cameras are used to perceive the surroundings (Section 1.3.3). However, many V-SLAM methods operate passively, e.g. [30, 50, 114, 148, 182]. This means that the robot follows predefined control inputs from an operator and merely builds a map while tracking its location without any autonomous decision-making. However, to enable large-scale deployment in diverse scenarios, the initial SLAM session must be efficient and unsupervised, eliminating the need for human oversight. Active SLAM has then been introduced to enable autonomous exploration and navigation in unknown environments while concurrently constructing a map. The term "Active" signifies that the robot makes real-time decisions about its movements and goals based on collected information, its current state, and sensor readings, thereby optimizing map estimation, exploration speed, and localization accuracy. The overarching goal of Active SLAM is to develop a framework that enhances autonomous exploration while balancing speed and precision [38, 203].

Most Active SLAM algorithms follow a common workflow [28] – i) selection of multiple candidate goals, ii) path generation towards them, iii) computation of a utility value for each, and iv) execution of the maximum utility one. Candidate goals can be either simple reachable robot locations or full poses (position and heading) and are often found alongside map frontiers, i.e. regions that lie between the explored and unexplored areas of the environment, enabling long-term exploration strategies. Alternatively, methods like receding horizon (RH) [157] or next-best-view (NBV) [107] planners, often referred to as short-term approaches, seek goals in the vicinity of the current robot location and immediately try to reach it [162]. For each goal, a utility is computed as a function of the expected information gain, often using the number of unexplored cells or the estimated map's entropy predicted to be observed from the considered locations. However, RH and NBV methods consider only the next pose and tend to result in incomplete explorations, as the robots can end up in local minima [19, 162]. Frontier-based approaches instead typically consider only the last reachable location to compute the process utilities [136, 162], disregarding the totality of the movement that would bring the robot to

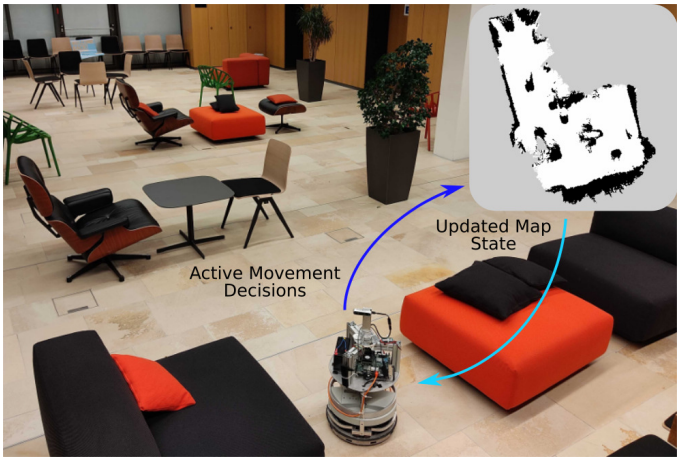



Figure 2.1: Our omnidirectional robot actively maps a reconstructed office reception area.

that position. Moreover, the camera's *orientation* in the final goal position is usually *not* optimized to maximize the information gain but is just the result of the interpolated output of the used path planning method. Exceptions like [47, 193], integrate this optimization only in the last reachable location, without considering the full extent of the path the robot would take to achieve that.

In this chapter, we focus on the autonomous exploration of indoor environments using an unmanned ground vehicle (UGV). We propose a novel Active SLAM method that leverages a camera to perceive the robot's surroundings. Our approach is inspired by a key observation: humans naturally turn their heads and eyes toward areas of interest while navigating — whether to avoid nearby obstacles, focus on long-term destinations, or study and memorize their surroundings. Similarly, a robot can actively control its camera heading during path execution to maximize environmental coverage. This includes observing previously unseen areas and refining already explored regions. By actively optimizing coverage, the robot can accelerate information acquisition and reduce overall map uncertainty more efficiently. We hypothesize that this strategy significantly decreases the total distance required for environment mapping, as confirmed by our experimental results. A shorter travel distance directly translates to lower energy consumption and increased operational autonomy—critical factors in many applications. Furthermore, this active behavior should extend to the planning phase, ensuring that the system evaluates the *total* utility of each potential path before execution.

Building on our insights, we propose a novel Active V-SLAM method comprising three layers of activeness:

1. The paths toward available goals, i.e. a set of frontier points, are divided into equally spaced waypoints. At each waypoint, the optimal heading direction to maximize a utility function is computed while considering subsequent frustum overlaps between them. Therefore, we can understand and exploit all the informa-

tion contained in the map *before* performing any movement (see Section 2.5.3). This allows us to have a long-term planning factor in our method.

2. While executing the chosen path, selected based on the total foreseen utility, an intermediate refinement is done every time a waypoint is reached. Going beyond simple sampling performed in classical NBV/RH techniques, we recompute the next waypoint's optimal heading based on the updated occupancy map. This allows us to dynamically account for variations in the map entropy and new information coming, for example, from loop closure events or the robot's movements (see Section 2.5.5).

3. Finally, we continuously refine the desired orientation of the camera using a weighting function between the previously selected optimal heading and the real time 3D features distribution. This increases the visibility of the features to help loop closure events while the robot moves between waypoints, incorporating in the system the real time information perceived by the robot (see Section 2.5.6).

We also introduce and experimentally compare three variations of the utility function, primarily based on the Shannon entropy formulation (Equation (1.16)). These variations refine how the robot selects actions by improving waypoint evaluation, incorporating obstacle awareness, and balancing exploration with re-observation, ensuring a continuously optimized mapping process. First, we use a weighted average between waypoints' utilities instead of a classic weighted sum. Second, we explicitly account for the presence of obstacles in the field of view (FOV). Third, we propose a balance between the exploration and re-observation behaviors of the robot. This allows us to differently balance the two main aspects in Active V-SLAM, i.e. the exploration and the refinement. Overall, our Active V-SLAM approach is active and continuously optimized throughout the *entire* mapping process, rather than at fixed intervals.

We first implement our method using an omnidirectional platform, which allows simultaneous rotational and translational movements. This flexibility enables our continuous control of the camera heading without it being constrained by holonomic or non-holonomic limitations. However, this design restricts the applicability of our framework to such platforms. Indeed, as commonly done, our camera is rigidly attached to the robot's body, allowing us to keep the relative transformation between the camera's optical center and the robot frame constant. However, this also introduces several drawbacks. First, it reduces the robot's freedom of movement. For non-omnidirectional vehicles, when a final specific orientation is desired, a fixed camera pose can increase time and energy consumption, as the robot must rotate in place or follow longer trajectories to adjust its viewpoint direction. This challenge affects not only iRotate but also other best-view planners like [157], as the platform's motion constraints influence trajectory planning. Second, robots typically have low rotational velocities and accelerations due to safety constraints, which prevent collisions and instability. This limitation slows camera reorientation when it depends entirely on the robot's movement. Third, moving the entire

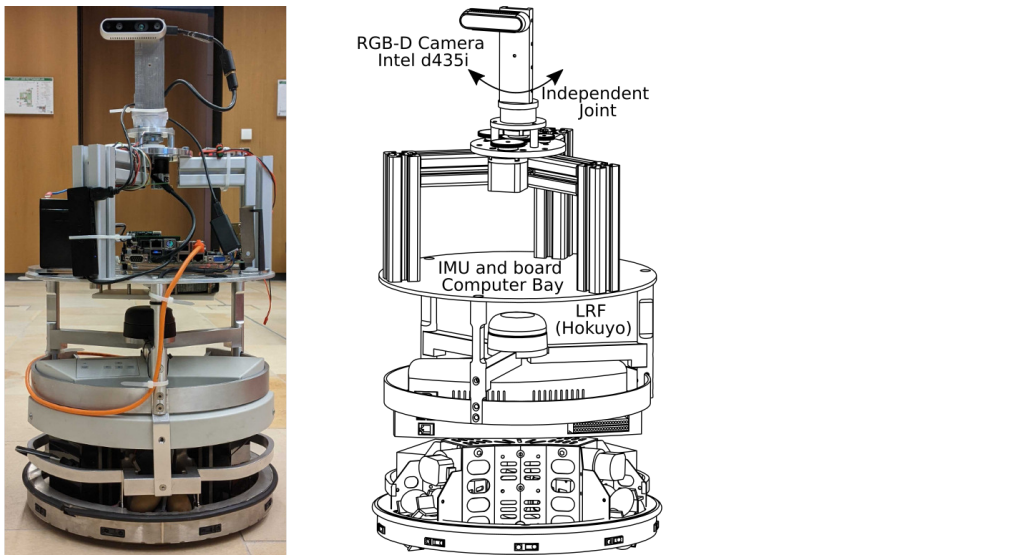



Figure 2.2: Our robot platform – Festo Didactic's Robotino with additional structure and hardware. Real robot (left), Gazebo model (right).

robot to adjust the camera view consumes significantly more energy than rotating only a lightweight camera on a small joint. This inefficiency becomes particularly relevant in energy-constrained applications.

Therefore, to adapt iRotate to semi-[1] and non-holonomic[2] platforms, we propose a new hardware system to allow independent control of the orientations of both the camera and the robot. This is achieved through a limitless rotating joint placed at the robot's top, over which we fix the camera itself. In practice, we are introducing an additional degree of freedom, increasing our flexibility in the mapping and control steps. However, this joint will also affect the overall performance of the SLAM system, as the relative orientation between the robot and the camera will be approximated and a source of estimation errors. Methods that utilize similar approaches usually disregard this problem and either assume continuous perfect calibration, use high-end encoders[3], or ignore the correlation between the two systems. These assumptions are both usually costly and not always achievable or verifiable throughout the whole experiment, e.g. even a small collision might affect the calibration between the camera and the platform. Instead, we express the camera orientation with respect to the robot's base through an estimate and an uncertainty value and then combine the camera and the robot's pose estimates in a unified representation. This is then provided to the Active SLAM framework itself, enabling us to include all the information with the corresponding uncertainty without modifying the underlying system, thus keeping our method independent of it.

[1] In this context, can move forward, backward, and sideways but rotate only in-place.
[2] Can only move forward or backward while turning but cannot move sideways.
[3] Devices that measure and provide feedback on the motor's position and speed.

Therefore, alongside the iRotate Active SLAM framework, we also:

1. Introduce a novel hardware architecture to address the aforementioned restrictions of having a fixed relative camera orientation.

2. Propose a novel state estimate formulation for this hardware architecture, to lower estimation errors and increase SLAM performance.

3. Expand iRotate to different semi- and non-holonomic platforms without any loss in performance while also resulting in more accurate maps and reduced energy consumption when applied to the omnidirectional one, thanks to the combined base-camera movements.

We validate our approach through both simulation and real-world experiments using the Festo Didactic Robotino (Figure 2.1 and Figure 2.2), a three-wheeled omnidirectional robot equipped with an onboard RGB-D camera and computational units. The independently rotating camera joint is controlled via a single low-cost incremental encoder. All experiments are conducted in consistent simulation environments, using the same base robot platform and parameters. Our results demonstrate that our method achieves comparable coverage to state-of-the-art approaches while significantly reducing overall map entropy. Notably, our approach reduces the traversed distance by up to 39% compared to other methods without increasing the total wheel rotations. The decoupled movement of the robot and camera further minimizes both wheel rotations and overall robot motion, leading to lower mapping uncertainty and reduced energy consumption while maintaining higher accuracy. Furthermore, our method generalizes to different robotic platforms, including holonomic (non-omnidirectional) and non-holonomic robots, without performance degradation. Code and implementation details are openly available at https://github.com/eliabntt/active_v_slam.

## 2.2 Related Works

### 2.2.1 Goals Selection and Active Control

Active SLAM algorithms aim to balance the autonomous exploration of new regions with the exploitation of previously acquired information, thereby reducing overall map uncertainty [33, 136, 203]. Selecting optimal goals is a critical step in Active SLAM, directly influencing the efficiency and effectiveness of exploration. State-of-the-art approaches often find them alongside frontiers [33, 47, 227, 249], seeking fast long-term exploration. Others, like [193] and [228], rely on persistent randomized rapidly exploring random tree (RRT) structures combined with short-term NBV or RH approaches. In [40, 116], the robot instead chooses the next set of actions based on the current global state using some randomly placed attraction particle. The *final* goal can be taken as-is or further optimized by using, for example, random sampling (as in [157]). In V-SLAM, the optimal

robot orientation for each nearby candidate pose [193] can be instead computed to improve the information observed by the camera. This optimization of the final goal can be combined either with long-term planning strategies, as in [47], or in an RH fashion where the chosen location will be reached immediately, like in RH-NBV planners [157, 193]. However, for RH approaches, there is a risk of getting stuck in the vicinity of a location in local minima. As a consequence, they often fail to explore the environment if the system is not carefully designed [136, 195] due to the local scope of these methods. Furthermore, when focusing on frontiers, as with iRotate, the path toward the final goal is typically not optimized or even considered toward the choice of the most informative final location. Moreover, the chosen goal remains fixed irrespective of what happens while the robot is reaching it [47]. Yet, the continuous refinement and information collection carried out by the SLAM framework and loop closure events might substantially impact and drastically change the condition under which the initial goal was selected. Therefore, blindly reaching the selected goal without considering the path toward it, or any further optimization or update step, might be inefficient. This can be solved, for example, by continuously re-updating the gains only of nearby reachable goals and saving them on a persistent tree, as proposed by Schmid et al. [193]. However, this is a costly operation.

Different from Schmid and other previous approaches, we propose a more flexible and adaptive approach by separating the heading computation into three distinct time scales: one long-term at the global path level, one short-term for the next reachable waypoint, and one in real time. Unlike previous methods, we follow a frontier-based exploration approach while *combining* it with ideas from RH and NBV techniques, thus benefiting from their specific characteristics. Initially, we find candidate frontiers, compute paths toward them, and pre-optimize the robot orientation along some waypoints along these paths using our utility formulation. This ensures that the camera headings for both i) each waypoint and ii) the final goal are pre-computed to maximize the predicted information gain. This results in a long sequence of NBV-like checkpoints that allows us to compute the *overall* utility of each possible path. Once complete, the best path is selected based on the total utility, and the robot can start to move. This constitutes our first active level (Section 2.5.3). During the execution of the chosen sequence of waypoints, which will remain fixed, we then apply a refinement similar to RH techniques. For each subsequent waypoint, to ensure a constant adaptation to the newly obtained information, we re-optimized the heading using the same utility function used in the previous steps in our second active level (Section 2.5.5). Therefore, we combine the benefits of NBV planners (path heading optimization), RH techniques (next viewpoint optimization), and frontier-based exploration (long-term goal).

Finally, real time features and landmark following have been widely studied since the seminal work of Davison et al. [49]. Tracking those elements is often an effective way to help visual odometry systems and/or the (re)localization steps in SLAM methods. This is usually enforced either through an external independent component as in [68], or integrally in the optimization method as in [116]. In contrast, we do not *directly* consider the effect of keeping the features within the field of view, nor do we use previously

observed features. Instead, we influence the real time desired camera orientation based on the *latest* features distribution and the computed optimal heading of the next waypoint. Our key insight is that we can increase the robustness 'blindly' by balancing both the feature tracking and the information gained within the system — all in real time in our third active level (Section 2.5.6).

### 2.2.2 Goal and Path Utility Computation

Computing the utility for all points along all available paths toward all available goals is usually computationally intractable. Moreover, as shown in [28, 208], an accurate method to simulate uncertainty evolution and loop closures to precisely predict the information gain remains to be found. This challenge is due to the multitude of effects that act simultaneously in a SLAM framework, such as relocalization events, newly discovered obstacles, imprecise robot control, or unforeseen drifts. Therefore, approximations need to be introduced like in [33, 157, 228] where the will-be-observed unexplored cells or the current map entropy are directly used as utility value to predict the information gain. As previously said, the goal that carries the highest score is to be selected as the current iteration objective. This total utility is usually computed as a linear combination of several metrics [28, 162] applied to the subsampled path's waypoints[4]. These metrics usually include factors like map consistency, future observed entropy, path length, and approximate robot state uncertainty. The balance between exploration and exploitation is then either performed implicitly within the utility formulation [33] or based on some hand-crafted heuristics and threshold values [40, 116]. Due to the inherent difficulty of accurately computing uncertainty evolution within a graph-based SLAM framework, and like previous approaches [157], we utilize the available map information to calculate the utility of each location, while discounting it for the distance that the robot would need to travel to reach it.

To compute map-based information-gain metrics, different functions have been proposed using Shannon [47] and Rényi [33] entropy formulations, the exploration volume [19, 195], or the quality of observations [193]. The one used in iRotate is based on the Shannon entropy (Equation (1.16)). For each candidate location and orientation, we compute the predicted information gain by evaluating the portion of the 2D grid map observed by the camera. Notably, visual features essential for loop closures and localization often cluster near obstacles. These areas are typically harder to map than free space, making their observation more valuable. Additionally, the relevance of exploring new areas versus refining previously mapped ones depends on the robot's distance from the frontier—exploration becomes less critical as the robot moves farther from the frontier, and vice versa. To dynamically balance this trade-off, in Section 2.5.2, we propose and evaluate two additional utility formulations: one based on the presence of obstacles within the FOV and another that accounts for the robot's position relative to the final goal.

[4]Each path has at least one: the final goal.

Moreover, unlike previous approaches such as [47, 227], which consider only the final waypoint, we incorporate the contribution of every waypoint along the path, following the insights from [157,228]. However, while these methods typically rely on interpolated trajectories or sampled orientations, our approach first evaluates the full 360-degree orientation span at each waypoint, selects the optimal orientation considering the frustum overlaps, and then computes the final path utility, thus ensuring a more comprehensive assessment. Finally, although the distance between the starting position and each waypoint is commonly used to estimate control-related uncertainty, a simple weighted sum of utilities may not adequately reflect the overall path contribution. Such an approach risks overemphasizing isolated score peaks instead of capturing the cumulative impact of the entire trajectory. To overcome this limitation, we introduce and evaluate a weighted average formulation that better represents the aggregated contribution of every *step* along the path.

### 2.2.3 Independent Robot-Camera Movement

Notably, the concept of optimizing the camera orientation has been used in many scenarios such as person tracking and surveillance [32, 52, 94], NBV planning and Active SLAM [47, 68, 193], 3D reconstruction [161] and others. Nonetheless, despite the clear advantages that controlling the camera's direction brings, existing approaches typically lack one or more of the following key aspects: (i) independent camera orientation with respect to the robotic platform, (ii) simultaneous control of robot and camera movements, and (iii) consideration of the uncertainty introduced by estimating the camera's gaze. In the simplest case, a camera freely moving in the 3D space can be considered *as* the 'robotic' system itself, e.g. [32, 52], where the pan-tilt movement made by the camera is performed from a fixed global location. In those, even if the pan-tilt(-zoom) system of the camera is actively controlled, there is no other movement involved. Other approaches, like [257], utilize the pan-tilt unit only during the system's calibration. After that, i.e. while the robot is moving and localizing itself, the camera orientation is fixed at all times. A notable exception is [161], where the authors propose a system of two RGB pan-tilt units used to gather *only* the color information for a 3D reconstruction system. In this study, the cameras can also reach *only* a set of pre-defined poses. Their use of high-accuracy encoders ensures precise and repeatable movements that are, however, lacking any form of real activeness, SLAM capabilities, or simultaneous control of the robot and the camera movements.

Most (Active) V-SLAM approaches rely upon static cameras attached to the robotic system. Thus, even if an active heading control is employed, it is the robot that rotates and not the camera. For example, we initially leverage the omnidirectional capabilities of a three-wheeled ground vehicle, while in [157], the high mobility of an aerial drone is exploited to compute informative headings. While alternative solutions might be considered, such as omnidirectional cameras, camera arrays, or dedicated rotational mechanisms, each presents inherent challenges that can hinder performance. For ex-

ample, camera arrays suffer from synchronization, calibration, and alignment problems. Omnidirectional/fisheye cameras introduce distortion, are not well suited for indoor environments, and usually lack the presence of a corresponding depth sensor [34]. In this context, it was the seminal work by Davison [49] that pioneered active feature tracking in SLAM by using one of the first independent rotating heading mechanisms. This and subsequent SLAM methods based on the work of Davison focus on the features' tracking to reduce localization uncertainty applied to hand-held cameras. In contrast, the camera in iRotate is not used to track features 'passively', but lies within an Active SLAM framework that continuously optimizes viewpoints towards not only improving localization but also mapping quality and exploration. Fintrop et al. in [68] propose an actual active gaze control for SLAM using artificial landmarks but do not introduce either a full Active SLAM approach or a connection between the camera and the robot's state estimates. Similarly to us, Manderson et al. [142] actively control the gaze of the system. However, the navigation is not performed autonomously but by an external operator while the state estimation is done only for the camera frame. Both of these methods are thus more related to an active control of the camera, rather than being fully-fledged Active SLAM approaches. Even [128], where the camera system is controlled separately with respect to the robot, gaze prediction is agnostic about the robot's movements and uses a greedy policy in the control, therefore lacking any link between the two and almost performing the two tasks separately.

To overcome this limitation and make iRotate working across different platforms, we introduce an independent camera rotational joint that decouples the camera's viewpoint optimization from the robot's navigation. This design allows the camera and the robot to rotate independently, enhancing both flexibility and scalability. Moreover, unlike Davison's work, which assumes known transformation matrices between the cameras and the robot, our method continuously estimates these transformations, ensuring greater adaptability. However, dedicated rotating hardware usually lacks the necessary precision to achieve reliable extrinsic calibration, leading to inaccuracies in the estimated camera pose. This, in turn, affects the alignment between the camera, the robot base, and the generated map, ultimately degrading mapping accuracy due to inaccurate reprojections. Additionally, the rotation mechanism itself introduces control challenges, such as mechanical drift and latency, further complicating precise localization and map consistency. More advanced systems involving independent and simultaneous camera control are often found in humanoid robotics, as shown in [144], where the pan-tilt motion of the head-mounted camera aids localization and target tracking. However, these approaches also do not explicitly account for the coupling between the robot's navigation and the camera's orientation. In such cases, the camera, rather than the entire robotic platform, remains the primary focus within a partially known environment, becoming the device with respect to which the map is being built, as opposed to the robot itself. Here, we address this limitation by introducing a novel state estimation method, as described in Section 2.4.3, to consider the effects that both robots and camera rotations have. We further hypothesize that, by allowing the independent camera rotation, the system will be capable of balanc-

ing the movement of both the robot and the independent rotational joint more efficiently. This is important since a lower rotational movement of the wheels translates into less energy that has to be used to move the robotic platform, improving energy efficiency. Ultimately, to the best of our knowledge, there was no method at the time combining a full Active SLAM approach with an active camera movement independently of the robotic platform and considering the system as a whole as we do.

### 2.2.4 Learning-based Active SLAM

In recent times, learning applied to SLAM has gained momentum. Several methods applying deep learning and deep reinforcement learning emerged, like [37,118,239]. Some of those apply to learning both the SLAM backend and to the Active SLAM procedure itself [37]. Others, like [118, 239], instead build on top of already deployed SLAM frameworks, e.g. EKF-SLAM. However, works based on RL approaches usually limit the applicability to different robotic platforms. In general, comparing these approaches is hard. Every RL agent is trained over specific traits, including, but not limited to, platform constraints (e.g. in the velocity space) and different sensor sets and ranges (e.g. using 2D LiDAR for mapping). Moreover, changing the SLAM framework responsible for building the map might harm the system itself due to the different distribution of the extracted information. Therefore, a comparison with these methods would be unfair since any observed difference in the performance would not be linked to the method itself but possibly to many factors (e.g. learning procedure, constraints, control parameters).

## 2.3 Problem Description and Notations

Let there be a ground omnidirectional robot, traversing on a 2D plane a 3D **static** environment, whose pose is in the local frame is given by $\mathbf{x}_R = [x_R \; y_R \; \theta_R]^\top$ and velocity by $\dot{\mathbf{x}}_R = [\dot{x}_R \; \dot{y}_R \; \dot{\theta}_R]^\top$. With $\mathbf{x}_G = [x_G \; y_G \; \theta_G]^\top$ and $\dot{\mathbf{x}}_G$, we indicate the robot's position and velocity in the global frame. Let the robot be equipped with an RGB-D camera with an associated maximum sensing distance $\delta_{thr}$ and horizontal field of view (H-FOV), a 2D laser range finder (LRF), and an IMU. The robot runs a V-SLAM method online and in real-time, constructing a 3D representation of the world which is then projected to a 2D probability grid map. The robot's goal is to quickly, efficiently (i.e. with minimal energy consumption or traveled distance), and autonomously map all the observable points in $M_{unk}$ as *free* or *occupied*. We assume that the robot begins exploration from a collision-free state. Building on the definitions we introduced in Section 1.3.3, all map cells belong to $M_{unk}$ at the beginning of the experiment. The problem is considered solved when the $M_{occ} \cup M_{free} \cup M_{hid} = M$ is verified, where we define $M_{hid}$ as the space that cannot be mapped, for example, unreachable locations. The robot must also keep the map's uncertainty as low as possible and have a final low trajectory error.

At any given time instant, the robot's state, map state, the set of observed visual features, and the graph $\mathbf{G}$ of previously visited locations generated by the V-SLAM backend are available to the robot. A node $\mathbf{n} = [x\ y\ \theta]^\top \in \mathbf{G}$ is defined by its coordinates in the map frame and by the orientation of the robot. While solving this problem, the robot must also be able to avoid collisions. Global and local path planning steps have to be performed without any knowledge of the environment or any landmarks in it.

We first propose an Active V-SLAM approach to solve the above-described mapping problem. The crux of our approach is to explore the environment by generating and executing paths that maximize information acquisition by the robot's sensors. More precisely, at a given instant, the robot i) identifies a suitable exploration goal position given by $\mathbf{x}_E = [x_E\ y_E\ \theta_E]^\top$, ii) based on the so-far acquired information up to that instant, generates a collision-free path, which consists of contiguous poses $\sigma = \{\mathbf{x}_R, ..., \mathbf{x}_E\}$, with maximum possible future information gain, and iii) executes the path by continuously re-planning its heading direction to incorporate all acquired information along the way and keep in its FOV as many visual features as possible. The exploration is considered concluded after a certain amount of time expires, as the robot is unaware of the total amount of area to be explored. Here, we use Shannon entropy as a measure of the map's uncertainty (see Equation (1.17)).

To enable the independent rotation, the camera is then mounted on a new joint that allows a full 360° independent rotation along the robot's z-axis, i.e. it has independent pan movement. We denote the orientation of the camera with respect to the robot as $\gamma_C^R$ and its rotational velocity as $\dot{\gamma}_C^R$. Therefore, the orientation of the camera with respect to the world frame is $\psi = \theta_G + \gamma_C^R$ and its velocity is $\dot{\psi} = \dot{\theta}_G + \dot{\gamma}_C^R$. When the camera is mounted in such a joint, we also equip the robot with an additional IMU unit. One of the IMUs will be attached to the camera, while one will be fixed on the robot's base. The goal then is to use the camera's independent rotation to allow a simultaneous turn of both the camera and the robot to reduce the overall energy consumption, thanks to the increased freedom of movement. As previously said, with this approach, we seek to apply our Active SLAM method, i.e. iRotate to different, non-omnidirectional, robotic platforms, achieving, at least, comparable performance. Finally, we will study the effect that the merging of the estimation of the camera and the robot poses has, seeking better performance in terms of the final map's entropy, accuracy, and trajectory error with respect to the corresponding non-combined alternative.

## 2.4 Kinematic Models and Uncertainty

### 2.4.1 The Robot Platform

Our robot, a Festo Didactic's Robotino 1 (Figure 2.2), has a 3-wheel omnidirectional drive base. We augmented the robot hardware with an extended physical structure containing the Hokuyo A2M8 2D Laser Range Finder (LRF), an RGB-D camera with an

embedded IMU (Intel RealSense D435i), and a single-board computer (Intel® Core™ i7-3612QE CPU @ 2.10GHz). For the non-omnidirectional trials, besides the rotational hardware to control the rotational joint and the encoder, an additional IMU unit was attached to the robot base near the central rotation axis. A virtual model of the same robot was created for the simulation experiments (Figure 2.2 right).

### 2.4.2 Kinematic Description

In a three-wheeled robot, we can identify a distance $D$ between the center of the robot and each wheel. The radius of the wheel is depicted with $r$, and the angular position of the wheel can be defined with $\alpha_i = i \cdot \frac{2}{3}\pi$ with $i \in [0,1,2]$, as they are equally spaced. With this information, the kinematic model can be obtained. $\mathbf{T}_R$ represents the rotation matrix between the global and the local frame of the robot:

$$\mathbf{T}_R = \begin{bmatrix} \cos(\theta_G) & \sin(\theta_G) & 0 \\ -\sin(\theta_G) & \cos(\theta_G) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.1}$$

Moreover, $\mathbf{S_L}$ denotes the static transformation from the robot's frame to each wheel:

$$\mathbf{S}_L = \begin{bmatrix} -\sin(\alpha_0) & \cos(\alpha_0) & D \\ -\sin(\alpha_1) & \cos(\alpha_1) & D \\ -\sin(\alpha_2) & \cos(\alpha_2) & D \end{bmatrix}. \tag{2.2}$$

The wheel velocities $\omega = [\omega_1, \omega_2, \omega_3]$ can be obtained with the following nonlinear relationship:

$$\omega = \mathbf{GRR} \cdot \mathbf{S}_L \cdot \mathbf{T}_R \cdot \dot{\mathbf{x}}_G, \tag{2.3}$$

where **GRR** is the gear reduction ratio matrix, i.e. a $3x3$ diagonal matrix whose diagonal elements are all $(\text{wheel\_radius})^{-1}$. This thus represents the full kinematic description of our omnidirectional robot.

With the introduction of an independent rotating joint usable to control the camera orientation, we create an additional degree of freedom. Then, by being $\dot{\gamma}_G$ the rotational velocity of the camera with respect to the robot in the global frame, we can control the system with $\dot{\mathbf{x}}_G^{'} = [\dot{x}_G, \dot{y}_G, \dot{\theta}_G, \dot{\gamma}_G]^T$ and modify the just introduced matrices as follows.

$$\mathbf{T}_R^{'} = \begin{bmatrix} & & & 0 \\ & \mathbf{T}_R & & 0 \\ & & & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \tag{2.4}$$

The last row links together the robot's and camera's rotations. Indeed, when the robot performs a rotation, that movement is transferred to the camera through the joint.

In both $\mathbf{S}_L$ and **GRR**, the differences are minimal. Indeed, with

$$\mathbf{S}_L^{'} = \begin{bmatrix} & \mathbf{S}_L & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

there is no transformation needed since the camera does not contribute to the robot's base movements. $\mathbf{GRR}'$ only introduces an element on the diagonal matrix for the corresponding gear ratio.

Thus, we can define our system with

$$\omega' = \mathbf{GRR}^{'} \cdot \mathbf{S}_L^{'} \cdot \mathbf{T}_R^{'} \cdot \dot{\mathbf{x}}_G^{'} \tag{2.6}$$

We assume that no ground truth data is available at any time. For the robot's pose estimate, we use a sensor fusion method based on EKF [149] to obtain position and velocity along with their uncertainty. It is obtained by using ICP odometry [164], the IMU, and the wheel's odometry with an a-periodic refinement provided by the loop closures events, which is used only for the $x, y$ components for stability reasons. This estimate is then used within the V-SLAM system without any further modification.

### 2.4.3 Merging State Estimates

Once we introduce the independent rotational joint, we need to disambiguate between the orientation of the camera and the orientation of the robot itself, as the two are linked but independent. Toward this end, we introduce an additional IMU unit alongside the encoder attached to the joint itself. The two IMUs are placed respectively on the camera and the robot base. The IMU attached to the base of the robot will detect *only* the rotational movements of the robot, while the camera one has no way to distinguish what is turning. Since for the camera's state estimation, we are interested in the *relative* yaw with respect to the robot's base, we cannot use either of the two IMUs directly. Therefore, we first time-synchronize the two IMU sources and compute the instantaneous relative measurements. We do not assume any alignment between the two IMUs. Note that, by being both IMUs rigidly attached to the robot's body, any translation offset that is there does not affect the angular velocity measured by them. Indeed, one would only need to account for the relative orientation of the two, which is assumed to be constant. The error of these measurements is not taken as the plain sum of variances, which would be rather low, but fixed at 0.01. This is because we recognize that perfect synchronization is not usually possible, and IMUs will suffer drift and possibly have spikes in their readings. The computed 'differential' IMU is then fused in a second EKF-based estimator with the encoder measurements, successfully obtaining an estimate of the relative orientation of the camera with respect to the base of the robot.

The SLAM framework is designed to use *only* the robot's pose and uncertainty within the built graph, while the camera relative pose is taken as a ground truth transformation by the system. In other words, it is *not* possible to account for *both* the robot and the camera's uncertainties. While this is not a problem whenever the camera is statically fixed, the independent joint rotation will continuously vary this transformation. Nevertheless, we propose a novel solution to this problem. Instead of considering the camera and the robot as separate entities, we merge the two state estimates at any time instant. In this, the position estimate is *exactly* the one obtained before, while the orientation is obtained through the sum of both the camera and robot's yaw estimates. The same applies to velocities. This state estimate can be indicated by $\mathbf{x}_{sum} = [x_G, y_G, \psi]$ with the corresponding velocity $\dot{\mathbf{x}}_{sum}$ while the overall variance is computed as the sum of the singular ones and can be directly used within any SLAM framework without any modification. In this way, we successfully account for the uncertainty introduced by the camera's independent movement. In the next sections, we will use $\theta$ to represent the heading; however, it can be substituted with $\psi$ to denote the equivalent orientation parameter without loss of generality.

## 2.5 Our Active SLAM Approach

In this section, we thoroughly describe our Active SLAM framework, which we named iRotate. Following the workflow of a Active SLAM method, we first describe our goal selection strategy in Section 2.5.1 and formulate how we compute the utility attainable from a given location and orientation in Section 2.5.2. Following these introductory sections, we introduce how we compute, refine, and select the best global paths towards those goals in Section 2.5.3, i.e. our first active level. The next step in our framework is to carry out the selected path through local planning and execution Section 2.5.4, with the introduction of the second Section 2.5.5 and third Section 2.5.6 active levels carrying out, respectively, the RH-like heading update and online refinement. Finally, we describe the finite state machine (FSM) that manages the integration between all these modules in Section 2.5.7.

### 2.5.1 Goal Selection Strategy

Reachable frontier map cells are the ones located on the edge between the explored and unexplored spaces, have no obstacles in the neighboring cells, belong to a region big enough for the robot to traverse, and for which there exists a traversable path connecting them with the current robot location. For each set of frontier cells, we need to extract a candidate goal. For this, we use the frontier extraction algorithm from [141] and modify it in the following way. At first, frontier clusters are identified. Next, candidate goal locations are selected at the centroids of those clusters. If a centroid is not reachable, a greedy search is performed in its proximity to identify the next reachable frontier cell in

that cluster. These are then marked as possible candidate goals. Further, if any of the candidate goals was the selected goal of the previous iteration, it is discarded, assuming it was unreachable. We then generate paths toward the remaining candidate goals, pre-optimize the camera headings, and compute their overall utilities using our first active layer. Indeed, contrary to other works, we do not select the closest frontier nor the one with the *singular* highest obtainable utility. Instead, the final goal position is associated with the *path* with maximum utility considering all the available candidates *after* the optimization.

Further, different from the most common approach for frontier-based autonomous exploration, we do not consider the task complete if frontiers are not available [33, 47, 136, 249]. In such cases, our robot either does a complete in-place rotation or tries to move to a randomly picked previously visited location, until the allotted time for the experiment has elapsed. This allows the robot to refine the map, activate new loop closures, and, eventually, 'enable' new frontiers. The robot will reach the selected goal with newly generated paths, i.e. not necessarily tracking back its previous movements, while still actively mapping the environment using all the activeness levels presented later. This selection is completely random as we do not search for high-entropy, feature-rich areas, or known old loop closure locations. We consider this a good trade-off between stopping the exploration and manually selecting a goal. While the described strategy is similar to what is employed in [208], the work of Stachniss et al. presents several differences. Indeed, they consider these alternative actions *while* performing explorations, believe the mapping uncertainty is minimal when no unknown areas are left, and terminate the experiment as soon as no gain is expected in the pose uncertainty. Differently, we recognize that the uncertainty is not given to be minimal in such situations, and new frontiers might be found after, for example, a loop closure or a refinement of a map portion. Moreover, forcibly selecting previously found loop closures might also result in an overall drifting or falling into local optima if not carefully tuned [28]. Another approach might be selecting only high entropy areas. However, in such cases, the system might suffer excessive drift or may not even be able to reach the goal locations at all, e.g. when placed in a tight spot. Moreover, since we try to reduce the entropy of the map by continuously controlling the heading of the robot with our three active levels, high-entropy areas might be directly covered while following the random objective. Finally, different from [116] and similar related works, we do not have any other behavior implemented that acts *concurrently* to the exploration to which we can resort, nor do we have an underlying RRT representation like [157, 195]. However, while the latter suffers from the necessity of carefully tuned parameters to successfully terminate the exploration [195], the former is based on maintaining a global set of features, which is not our case.

### 2.5.2 Pose Utilities Formulations

We use the utility as a measure of information gain for an attainable robot pose $\mathbf{x}$ (position and orientation) on the map that lies along a path $\sigma$, leading to the goal location $\mathbf{x}_G$.

Given a pose $\mathbf{x}$, we obtain a set of $N$ map cells $S_{\mathbf{x}} = \{\mathbf{m}_0, ..., \mathbf{m}_N\}$ that are visible from the pose $\mathbf{x}$. A cell $\mathbf{m}$ is deemed visible if there exists a ray that goes from it to the queried pose $\mathbf{x}$ without intercepting any obstacle and lies within the camera's FOV and whose distance is smaller than $\delta_{thr}$, hypothesizing the robot at pose $\mathbf{x}$. A cell is considered an obstacle if $p_o(\mathbf{m}) > 0.7 = p_{thr}$. At every cell $\mathbf{m} \in S_{\mathbf{x}}$, we associate a utility value $\mu[\mathbf{m}|\mathbf{x}, \sigma]$. The total utility $U_{[\mathbf{x},\sigma]}$ of a pose $\mathbf{x}$ along a path $\sigma$, is given as

$$U_{[\mathbf{x},\sigma]} = \sum_{\mathbf{m} \in S_{\mathbf{x}}} \mu[\mathbf{m}|\mathbf{x}, \sigma]. \tag{2.7}$$

We compare three different ways to compute $\mu[\mathbf{m}|\mathbf{x}, \sigma]$. The first way, $\mu_1$, is to define it as the plain Shannon entropy, which represents the available amount of information, i.e. $\mu_1[\mathbf{m}|\mathbf{x}, \sigma] = H[\mathbf{m}]$. With this, we can capture directly the maximum amount of information retrievable from a given cell. In the second way, $\mu_2$, we introduce the presence of obstacles directly in the utility computation.

$$\mu_2[\mathbf{m}|\mathbf{x}, \sigma] = \begin{cases} H[\mathbf{m}] & \text{if } p_o(\mathbf{m}) < p_{thr} \\ H[\mathbf{m}] + \kappa_1 & \text{if } p_o(\mathbf{m}) \geq p_{thr} \ , \end{cases} \tag{2.8}$$

where $\kappa_1$ is a constant (set to 1 in our experiments). The insight is twofold in this case. First, we know that visual features are usually found around obstacle regions, as it is where we can observe contrast and variability against the background. At the same time, those are harder to map correctly with respect to empty areas. However, the entropy of those cells goes rapidly toward 0 due to Shannon's formulation. Therefore, adding this constant term to account for obstacle presence will favor robot headings towards that kind of cell, improving the likelihood of observing more features and 'forcing' it to observe objects to refine their mapping. Thus, differently from $\mu_1$, we *explicitly* favor the observation of obstacle cells.

The third way of computing utility, $\mu_3$, is based on the following idea. Along the path, a robot can balance the observation of new, unmapped cells and the re-observation of previously mapped ones. Generally, if a robot is far with respect to a frontier point, the amount of unknown area it can explore locally is very limited but might still have a big impact on the utility function. As unmapped cells have $p_o(\mathbf{m}) = 0.5$ we have that $H[\mathbf{m}] = 1$ if $\mathbf{m} \in M_{unk}$, i.e. the maximum value of $H$. With $\mu_3$ we aim to emphasize the refinement of already mapped cells rather than exploring the new area while the robot is far with respect to the final goal and vice versa. With this, we seek to differentiate the objective dynamically between re-observation (refinement) and exploration, by differently weighting the two based on the distance between the robot and the frontier point. To this end, we first define

$$\lambda_{[\mathbf{x},\sigma]}[\mathbf{m}] = \begin{cases} (1 - j[\mathbf{x}, \sigma]) & \text{if } \mathbf{m} \in M_{unk} \\ j[\mathbf{x}, \sigma] & \text{if } \mathbf{m} \in M_{free} \bigcup M_{occ} \ , \end{cases} \tag{2.9}$$

as our weighting function, where $j$ itself is a function of the distance between the queried pose $\mathbf{x}$ and the final goal pose $\mathbf{x}_G$ given as

$$j = \max(\xi_l, \min(\xi_h, \xi_l ||\mathbf{x} - \mathbf{x}_G||_2)). \tag{2.10}$$

Constants $\xi_l$ and $\xi_h$ are parameters to tune re-observation priorities (set to 0.2 and 0.8, respectively, in our experiments). Note that it is always verified that $j \in [0.2, 0.8]$ and the further we are from $\mathbf{x}_G$, the nearer we are to $\xi_h$. Using this weighting function, $\mu_3$ is given as

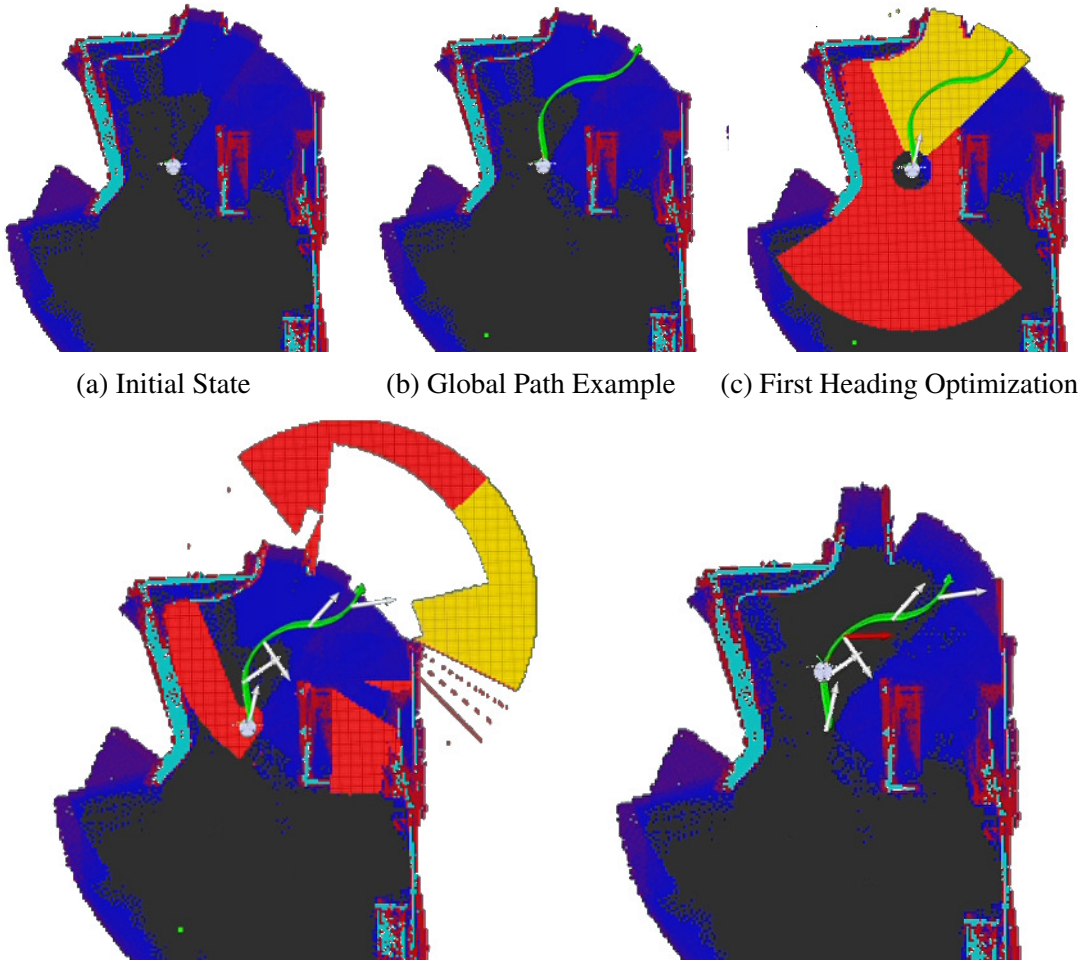
$$\mu_3[\mathbf{m}|\mathbf{x}, \sigma] = \begin{cases} \lambda_{[\mathbf{x},\sigma]}[\mathbf{m}]H[\mathbf{m}] & \text{if } p_o(\mathbf{m}) < p_{thr} \\ \lambda_{[\mathbf{x},\sigma]}[\mathbf{m}]H[\mathbf{m}] + \kappa_1 & \text{if } p_o(\mathbf{m}) \geq p_{thr}. \end{cases} \tag{2.11}$$

By using $\mu_3$, we increase the weight of the utility related to the entropy of the already explored cells while the robot is far from the final frontier point. Then, while the robot moves toward the exploration zone (the frontier point), the weight is shifted toward the utility related to the unmapped cells. We built this on top of $\mu_2$, meaning that we keep the obstacle weight for the reasons explained above. However, different from both $\mu_1$ and $\mu_2$, here we dynamically balance between re-observing the already explored area and mapping new space.

### 2.5.3 Global Path Generation, our *First Active Level*

For every available candidate goal, we generate a path with an A$^\star$ planner [82] applied to the global occupancy grid. Every path $\sigma$ obtained in this way is reduced to a set of waypoints $W = \{\mathbf{w}_0, ..., \mathbf{w}_E\}$, where a waypoint is defined as $\mathbf{w}_i = [x_i \; y_i \; \theta_i]^\top$, $\mathbf{w}_0 = \mathbf{x}_G$, $\mathbf{w}_E = \mathbf{x}_E$. The maximum distance between waypoints is limited to a fixed value (1 m in our experiments) and is computed as the linear sum of Euclidean distances between the subsequent cells belonging to the path. This discretization is done for several reasons. First, this lowers the overall computational requirement of the complete Active SLAM approach. Second, it facilitates the robot's dynamic constraints by allowing the robot sufficient time to rotate such that the required heading of the subsequent waypoint is achievable and unnecessary in-place rotations are avoided.

The A$^\star$ planner natively returns a heading for every point of the path based on some internal heuristics. A naive but non-active approach would be to interpolate the ones belonging to the waypoints following the movement direction to obtain the predicted and desired headings. However, this would limit the observation capabilities of the system, not fully exploiting what is available at this stage of the Active SLAM system. Differently, instead of having a series of mostly overlapping interpolated headings along the path, we compute a new and *enhanced* series of headings. Those are computed such that the utility of the path is optimized with respect to the information that is currently available.



(a) Initial State (b) Global Path Example (c) First Heading Optimization

(d) Computed Optimal Headings (e) Best View Update

Figure 2.3: A representation of the first and second levels of activeness. The initial state, shown in (a), depicts the current voxels' occupancy probability based on the RViz [103] standard color coding, i.e. the free cells are black, likely-free blue, likely-obstacles red, and obstacles light-blue, while in white it is unexplored space. In (b), we show the frontier points as green dots and a path toward one of them. In (c) and (d), we depict our **first active level**, i.e. the optimal headings with the frustum overlap. We show with a red circular area the set of visible cells from a given location, in yellow the ones covered by the optimized orientation, we omit the overlapping cells, and depict the final optimal headings as white arrows. In the final image (e), we can clearly see the difference between the pre-computed optimal view direction (white arrow) and the one by the **second active level**, computed only once the previous waypoint is reached (red arrow).

Note that, differently from the case of non-omnidirectional platforms mounting *fixed* mounted cameras, we do not need to recompute the approach path since our robots have no limitations in their movement capabilities, either thanks to the omnidirectionality or the independent rotational joint. Therefore, for our *first-level activeness*, we compute the optimal heading for every waypoint. In this, for every waypoint $\mathbf{w}_i = [x_i \; y_i \; \theta_i]^\top$ we replace $\theta_i$ by $\theta^\star$, where the latter maximizes the waypoint's utility value, i.e.

$$\theta^\star = \underset{\theta_i \in [0,2\pi]}{\operatorname{argmax}} U_{[\mathbf{w}_i,\sigma]} \tag{2.12}$$

However, instead of directly using Equation (2.7) to compute $U_{[\mathbf{w}_i,\sigma]}$, we update the map cell set $S_{\mathbf{w}_i}$ in a way such that the cells visible from the waypoints before $\mathbf{w}_i$ are not included in $S_{\mathbf{w}_i}$, i.e. we account for the frustum overlap of the subsequent camera views. The final path utility is then given by

$$U[\sigma] = \frac{\sum_{i=0}^{N} e^{-\rho \cdot \delta_i} \cdot U_{[\mathbf{w}_i,\sigma]}}{\sum_{i=0}^{N} e^{-\rho \cdot \delta_i}}, \tag{2.13}$$

where $\delta_i$ is the path-distance between the waypoint $\mathbf{w}_i$ and the current robot location and $\rho$ is a discount factor set to 0.25 in our experiments. Instead of the classical approach, like the one used in [157] that considers a plain weighted sum, we consider a weighted average between all waypoints' utilities. This choice enables us to capture a more comprehensive contribution of the whole path in terms of overall information gain. With a weighted sum, the system favors paths that exhibit spikes in the utility, caring only about a high cumulative reward rather than how that reward is obtained and how much it will gain at every step. Instead, the weighted average approach considers how much the robot can gain along the *whole* path, seeking routes that are more informative on average irrespective of their length. Finally, the highest utility path is selected, and its waypoints are used by the local path execution module, as described next. A depiction of the global path planning and the waypoints' heading optimization can be found in Figure 2.3.

### 2.5.4 Path Execution and Robot Control

We use a Nonlinear Model Predictive Control (NMPC) based method for the local path execution. The NMPC formulation is initially based on the kinematic description of a three-wheeled robot introduced in Section 2.4. We can indicate the state $\mathbf{x}_t$ is the robot's position and orientation at a given time instant $t$. The robot is controlled in the velocity space using the vector $\mathbf{u}_t(n) = [u_{t,x} \; u_{t,y} \; u_{t,\theta}]^T$, with $n$ as the horizon time step. The objective is to reach the desired goal location and orientation, i.e. the next waypoint $\mathbf{w}_i$. The location is defined within the first active level (Section 2.5.3), while the desired orientation is provided by the second and the third level of activeness (see Section 2.5.5 and Section 2.5.6). To ensure safe operations, the robot must also be able to avoid ob-

stacles and stay within its control bounds. Thus, the NMPC is formulated as follows

$$\mathbf{x}_t(0)^\star \dots \mathbf{x}_t(N)^\star, \mathbf{u}_t^\star(0) \dots \mathbf{u}_t^\star(N-1) = \underset{\mathbf{u}_t(0)\dots\mathbf{u}_t(N-1)}{\textbf{arg min}} \quad (J) \tag{2.14}$$

$$\texttt{s.t. } \mathbf{x}_t(n+1) = f(\mathbf{x}_t(n), \mathbf{u}_t(n)) \tag{2.15}$$

$$\begin{aligned} &-\mathbf{u}_{max} \leq \mathbf{u}_t(n) \leq \mathbf{u}_{max} \\ &-v_{tr,max} \leq v_{t,tr}(n) \leq v_{tr,max} \\ &d_t^k(n) \geq d_{min}, \ \forall\ k\ \in \mathbf{obs}(n) \end{aligned} \tag{2.16}$$

Where $f$ is the nonlinear kinematic equation (see Section 2.4), $v_{t,tr}(n)$ represents the *overall* translation velocity, i.e. $(u_{t,x}^2 + u_{t,y}^2)^{0.5}$.

Note that while A$^\star$ provides an obstacle-free path, it is necessary to include obstacle constraints in the local re-planning and execution to avoid collisions with any newly discovered obstacle cell. We thus consider $\mathbf{obs}(n)$ to be the vector containing all the predicted obstacle positions and $d_t^k(n)$ to be the distance between the robot and the $k$-th obstacle at the $n$-th time step. We keep $u_{\{x,y,\theta\},max}, v_{tr,max}, d_{min} \in \mathbb{R}^+$. The robot then must minimize the distance to the target, the control effort, and an additional cost term for being near an obstacle. The objective $J$ can be defined as follows

$$\begin{aligned} J = &(\mathbf{x}_t(N) - \mathbf{w})^T \mathcal{Q}_x (\mathbf{x}_t(N) - \mathbf{w}) + \\ &+ \sum_{n=0}^{N-1} \Big[ (\mathbf{x}_t(n) - \mathbf{w})^T \mathcal{Q}_x (\mathbf{x}_t(n) - \mathbf{w}) + \mathbf{u}_t(n)^T \mathcal{R} \mathbf{u}_t(n) + \\ &\mathbf{z}_t(n)^T \mathcal{Q}_{obs} \mathbf{z}_t(n) \Big] \end{aligned} \tag{2.17}$$

$\mathbf{z}_t(n) = [h_t^1(n) \ \dots \ h_t^k(n)]^T$ indicates a 'force' factor that drives the robot away from the obstacles. This is computed, for each one of the $k$ obstacles, as $h_t^k(n) = -(1 - e^{0.1/d_t^k(n)^2})$. In addition to the minimum distance constraint, we add this term to further discourage the robot from going near them to better cope with noise due to the SLAM procedure and possible errors in their localization. With $\mathcal{Q}_x$, $\mathcal{R}$, and $\mathcal{Q}_{obs}$ defined as the applied positive diagonal weighting matrices.

To control the robot, we create and generate the NMPC formulation with the ACADO framework [89]. We first control the robot in an omnidirectional modality. Then, we modify the NMPC formulation to account for the independent rotation of the camera. In this, the controlled orientation considered in the framework is the *final global* orientation of the *camera*. The penalty factor associated with the camera's rotation is the same as that of the robot's rotation used in the previous case. We use this formulation to test

both a simultaneous rotation of the camera and the robot and a scenario where only the camera is allowed to rotate. Note that with the latter, we are effectively using a semi-holonomic robot that is capable of moving only along the $x$ and $y$ axes and linearising the whole system. Finally, to simulate a non-holonomic robot, we impose a hard constraint in the NMPC formulation setting $v_y = 0$. In all experiments, we keep the same maximum rotational speed of the camera, of the robot, and also of the whole system.

### 2.5.5 Local Waypoint Refinement, our *Second Active Level*

The second active level is carried out immediately before sending a waypoint as a goal to the NMPC. We exploit the observation that the SLAM backend continuously updates our knowledge *while* the robot is moving. Indeed, the information considered during the previous global planning and optimal heading generation could have changed drastically. This can be, for example, due to loop closure events, intermediate robot movements, newly mapped obstacles, or further map refinements or corruptions. Following this observation, each time the robot reaches a waypoint $\mathbf{w}_i$, we re-compute the optimal heading of the subsequent waypoint $\mathbf{w}_{i+1}$. We do so by using the same procedure described previously to compute optimal headings, but this time we leverage the updated map information. We then obtain a refined $\tilde{\theta}^{\star}_{i+1}$ that will be used as our new refined optimal heading. In this way, we ensure that we include the most recent and updated version of the occupancy grid not only when we do the global long-term planning, but also every time we reach a sub-goal. A representation of this can be seen in Figure 2.3.

### 2.5.6 Real Time Heading Refinement, our *Third Active Level*

Our third active level, carried out *while* the robot is moving in *between* two consecutive waypoints. Since the V-SLAM back-end computes a set of 3D features for every time step, the key idea is to keep in the robot camera's FOV as many 3D visual features as possible. We do so by weighting the optimal heading direction computed in the previous step with a newly computed orientation that would help retain these features in the FOV. Our insight is that this would help the relocalization and loop closure modules, as the camera is more stable and eventually focuses on areas with more visual information. To this end, every time a node is added to the graph, we obtain the 3D locations of all the features extracted from the current image. In this way, only the features relative to the latest added node are being used. Using this, we can compute the orientation that the robot should have in the next waypoint, $\beta_{i+1}$, to keep them in the FOV. This piece of information was not available during the previous step as it is only computed in real time. Therefore, the planning carried out during the first and the second active levels could not consider this information during the computation of the predicted information gain. Also, even if a global features map were to be maintained, and disregarding the computational complexity, new and better features might anyway be found within the newly taken images.

Then, at any given time instant $t$ while traversing between waypoints $\mathbf{w}_i$ and $\mathbf{w}_{i+1}$, we have at our disposal i) $\tilde{\theta}^{\star}_{i+1}$, the desired optimal angle of the next waypoint $\mathbf{w}_{i+1}$ as computed by the *second-level activeness*, and ii) $\beta_{i+1}$. At every time step, we re-compute $\beta_{i+1}$ with a 3D ray-tracing technique on the updated octomap by using the FOV of the robot camera, the maximum distance of the features from it, and the next waypoint's 2D position. Let $\delta_t$ be the distance between the current robot position and the next waypoint $\mathbf{w}_{i+1}$, always at time $t$. In real time, we can finally compute the current desired orientation for the robot, $\gamma_t$, as

$$\gamma_t = \frac{\tilde{\theta}^{\star}_{i+1} \cdot e^{\kappa_2 \cdot \delta_t} + \beta_{i+1} \cdot e^{\kappa_3 / \delta_t}}{e^{\kappa_2 \cdot \delta_t} + e^{\kappa_3 / \delta_t}}, \tag{2.18}$$

where $\kappa_2$ and $\kappa_3$ are some weighting constants. The NMPC's objective is now created with the goal of to tracking the continuously updated waypoint $\mathbf{w}'_{i+1} = [x_{i+1}\ y_{i+1}\ \gamma_t]^\top$, whose position component is the same as $\mathbf{w}_{i+1}$ and the orientation component is constantly updated to the newest $\gamma_t$. Varying the parameters $\kappa_2$ and $\kappa_3$ allows varying the importance of keeping 3D visual features within the FOV versus achieving the previously desired orientation $\tilde{\theta}^{\star}_{i+1}$. For instance, in our experiments, we set $\kappa_2 = -6$ and $\kappa_3 = -0.5$ to keep the features within the FOV as long as the robot is far enough with respect to the next waypoint. As it approaches the following waypoint, the weight of the previously computed heading becomes more prominent. Indeed, since $\lim_{\delta_t \to 0^+} e^{\kappa_2 \cdot \delta_t} = 1$ and $\lim_{\delta_t \to 0^+} e^{\kappa_3 / \delta_t} = 0$ we have that $\lim_{\delta_t \to 0^+} \gamma_t = \tilde{\theta}^{\star}_{i+1}$. However, we acknowledge that features are not always available. In such scenarios, the system sets $\beta_{i+1}$ to the *current* heading direction of the robot to ensure a smooth transition between the two, following the same weighting procedure. A visual representation of this function with the chosen parameters is provided in Figure 2.4. Overall, the *third-level activeness* increases the chance of loop closures and place recognition (especially while revisiting areas), thus improving map quality and lowering trajectory errors.

### 2.5.7 Coordinating The Modules

A finite state machine manages the execution of the three Active SLAM layers. A simplified scheme of the FSM is depicted in Algorithm 2.1. The NMPC continuously publishes its state (*WAITING*, *OPERATING*, *GOAL REACHED*, or *RECOVERY*). This is the main signal influencing what the FSM does. If the NMPC is waiting for commands (*WAITING, GOAL REACHED* states), the FSM will then check if a goal (waypoint) is available and send it in the form of a desired location and optimal orientation. If the current list of waypoints is empty, i.e. there is no goal already available, it will request one. This is done by our first level of activeness (Section 2.5.3) and consists of both i) goal selection and ii) global path generation steps. The pre-optimized waypoints are then stored in a queue, and the first one is executed immediately. If, instead, the current list of waypoints is not empty, the FSM will refine the heading of the next sub-goal, i.e. our second

**Algorithm 2.1** Pseudocode of the finite state machine (FSM) coordinating our proposed Active SLAM layers.

```
Result: Exploration
state = get NMPC state

if state = WAITING or state = GOAL REACHED then
    if waypoints is not empty then
        // Second-level activeness
        optimize next heading
        // 'In' NMPC Third Active Level
        send a waypoint to NMPC
        popfront(waypoints)
    else
        endpoints = require frontier points
        if endpoints is empty then
            if map graph is empty then
                endpoints = 360 rotation in place
            else
                endpoints = random graph node
            end
        end
        // First-level activeness
        waypoints = getOptimalSolution(endpoints)
        // 'In' NMPC Third Active Level
        send first waypoint to NMPC
        popfront(waypoints)
    end
else if state == RECOVERY then
    recovery easy
    if recovery easy fails then
        recovery hard
    end
else if state == OPERATING then
    do nothing
```

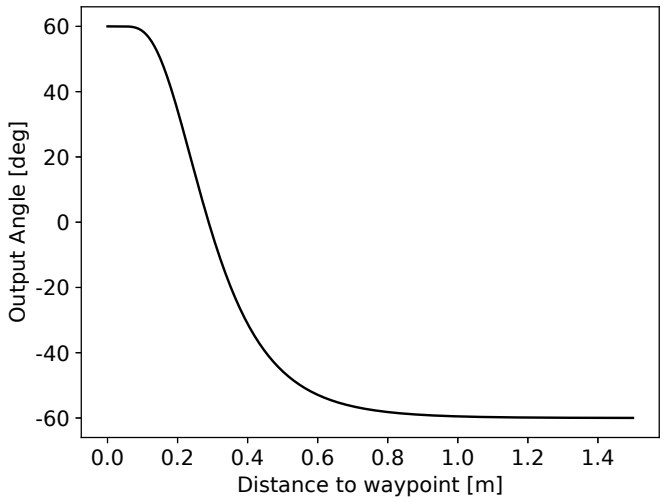



Figure 2.4: Example of a computed $\gamma_t$ using $\tilde{\theta}^{\star}_{i+1} = 60^{\circ}$, $\kappa_2 = -6$, $\beta_{i+1} = -60^{\circ}$, $\kappa_3 = -0.5$, and $\delta_t \in [0, 1.5]$ m (distance to waypoint).

level of activeness (see Section 2.5.5), and send the optimized version to the NMPC. If the NMPC is actively working (*OPERATING*), the FSM will stay idle. Between each control loop, we take care of balancing the feature tracking and the current orientation objective via our third level of activeness (see Section 2.5.6) directly within the NMPC control code. Specifically, this is obtained by computing the heading based on Equation (2.18) and passing the result online as the heading objective of the NMPC while keeping the location of the waypoint fixed. To ensure full autonomy of the system, a recovery behavior is introduced. If the robot finds itself stuck and unable to move, it will send a *RECOVERY* signal to the FSM. First, it tries to solve a situation via common recovery behaviors like in-place rotation and going away from the nearest obstacle (*recovery easy* in the code). Then, if that fails, by leveraging the multi-session mapping capability of the SLAM framework, we start a new mapping session (*recovery hard* in Algorithm 2.1). The current pose of the robot will be considered the new origin, and the FSM will act as previously described. Once a loop closure is found with the previous map, the two will be merged and rectified. The first behavior that the robot exhibits will always be a 360° in-place rotation to explore the immediate vicinity of the platform. The loop will continue until an external termination signal is triggered.

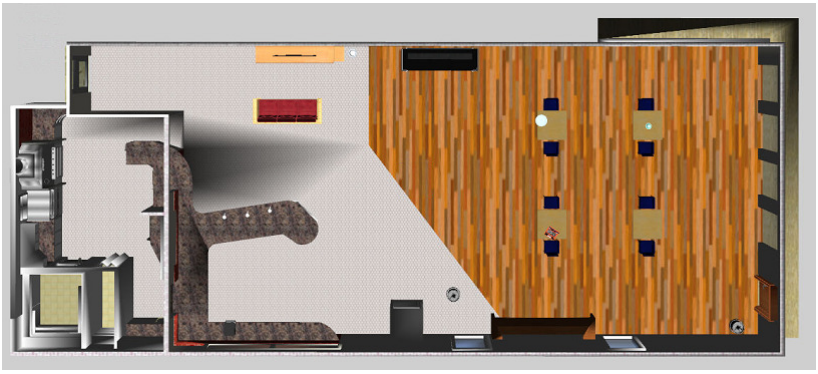
## 2.6 Experiments

### 2.6.1 Setup and Implementation

To validate the proposed algorithm, we first use the Gazebo simulator [110]. We test our method in two different environments: i) AWS Robomaker Small House World [179]

(Figure 2.5a), which is $\sim 180$ m$^2$, and ii) a modified version of the Gazebo's Café environment (Figure 2.5b) using 3DGems' [176] assets, which is $\sim 200$ m$^2$.

(a) AWS's Small House



(b) Gazebo's edited Café

Figure 2.5: Top-down view of the two simulated environments

We use real hardware-like parameters for all the sensors in the simulation. The LRF has an angular resolution of 1° and provides a complete 360° sweep. The camera is used at a resolution of 848x480 pixels. Its horizontal FOV is 69.4° [79], and its maximum sensing depth is limited to 4 meters. For all experiments, the maximum translation speed of the robot is set to 1 $\frac{\text{m}}{\text{s}}$, while the maximum angular speed is 1 $\frac{\text{rad}}{\text{s}}$. The ground truth maps were obtained using *pgm_map_creator*[5] and edited to remove unwanted artifacts. Since Gazebo does not support components with zeroed friction coefficients in the perpendicular direction, it is not possible to directly simulate three-wheel omnidirectional robots. Therefore, we disable wheel collisions and augment the model with six independent virtual joints (one for each DOF) to enable control. The wheels are rotated (without friction or collisions) using the kinematic model introduced above.

[5] https://github.com/hyfan1116/pgm_map_creator

As introduced in the previous section, we use the widely adopted ACADO framework [89] to set up the optimization and qpOASES [66] as a solver. We set a 20 steps horizon, each consisting of 0.1 s. We consider *ten* obstacles in our implementation, and their position is treated as online data in the framework. The obstacles are detected through the *costmap_converter*[6] package, which provides us with the estimated obstacle's velocity and convex hull. The nearest vertex is used for each one of the detected obstacles. The obstacles are then sorted based on their status (dynamic or static) and their distance to the robot. Priority is given first to the dynamic ones, and then to the nearest static ones, with respect to the state predicted at each time step by the previous iteration of the NMPC. Our initial state is the most recent estimate of the robot's pose provided by the odometry estimation system. Note that, despite the system itself can, in principle, manage dynamic obstacles, we will assume that the robot operates in a **static** environment in our experiments.

We use RTAB-Map [114] as our V-SLAM back-end. RTAB-Map is a graph feature-based Visual SLAM framework. The 2D occupancy map is generated by the 2D projection of the 3D octomap built through our RGB-D sensor and used for navigation and evaluation, allowing the mapping of the obstacles that would be otherwise 'hidden' to the 2D LRF.

### 2.6.2 Evaluation Metrics

The constructed map has a cell size of 0.05 m and is compared, for the simulated experiments, with respect to the ground truth map using the balanced accuracy score (BAC) on the three classes: free, occupied, and unknown. We also evaluate the root mean squared absolute trajectory error (ATE RMSE, see Equation (1.15)) and the number of loop closures triggered per meter of distance traveled by the robot. The map entropy is also monitored throughout the entire experiment and presented as normalized with respect to the actual explored area (Equation (1.18)). Wheels' total rotation per meter of trajectory traveled is computed to further show the energy-saving benefits of our method by indicating the actual movement carried out throughout the experiment. Finally, the total area explored is monitored both in time and against the meters traveled by the robot. While the first is an indication of exploration speed, the latter is employed to compare energy efficiency between the algorithms and capture the overall movement that the robot takes to discover a comparable amount of area. All results are presented with the mean and standard deviation computed among all the trials.

For both simulation and real robot experiments, one might notice drops in the exploration amount and spikes in the normalized entropy. Those are related to two main factors. The first one is related to the RTAB-Map system itself since, once a loop closure is triggered, the whole map gets rectified and updated with subsequent possible degradation and corruption. The second one is related to the recovery procedure. In this case,

[6] https://github.com/rst-tu-dortmund/costmap_converter

the current explored area and its normalized entropy are like the ones at the beginning of the experiment, until a loop closure with the previous session is found. We do not use a cumulative sum of the explored area among the two sessions since we have no way to ensure that the 'newly' explored space is different with respect to the previous one. Once a loop closure is found with the previous map, the two get automatically merged, 'restoring' these two values.

### 2.6.3 Methods Naming and Description

#### Omnidirectional Robots

Since our Active SLAM is extremely modular, in our evaluations we perform various comparisons to analyze the effect of the modifications we introduce with iRotate. These include each active level, utility formula (including the weighted average), independent joint rotation, and state estimate mechanism.

First, we consider 2 additional paths' utility computation methods. These are i) OL: using only the optimized goal position instead of all the waypoints on the path, and ii) INTER: the orientation of the robot is simply interpolated along the trajectory to the goal position, and the total utility is computed as a plain sum of the utilities. Both of these, in practice, neglect our *first-level of activeness*. Active SLAM approaches using these for utility computation are prefixed by OL and INTER, respectively. Approaches using our complete procedure of global path generation (Section 2.5.3) are prefixed with A.

| | Active Levels Enabled | | | Interpolated headings | Only Last Waypoint | Sim. Time | Utility |
|---|---|---|---|---|---|---|---|
| | First | Second | Third | | | | |
| A | ✓ | ✓ | ✓ | | | 600 s | $\mu_1$ |
| A_1 | ✓ | | | | | 600 s | $\mu_1$ |
| A_L | ✓ | ✓ | ✓ | | | 750 s | $\mu_1$ |
| A_S | ✓ | ✓ | ✓ | | | 600 s | $\mu_1$ |
| A_O | ✓ | ✓ | ✓ | | | 600 s | $\mu_2$ |
| A_DW_O | ✓ | ✓ | ✓ | | | 600 s | $\mu_3$ |
| INTER_0 | | | | ✓ | | 600 s | $\mu_1$ |
| OL_0 | | | | | ✓ | 600 s | $\mu_1$ |
| OL_1 | ✓ | | | | ✓ | 600 s | $\mu_1$ |
| OL_1_3 | ✓ | | ✓ | | ✓ | 600 s | $\mu_1$ |
| OL_2 | | ✓ | | | ✓ | 600 s | $\mu_1$ |
| OL_2_3 | | ✓ | ✓ | | ✓ | 600 s | $\mu_1$ |

Table 2.1: Summary of the methods used for the comparisons using the omnidirectional platform. For A_S we use $\mu_1$ with a weighted sum of utilities, instead of a weighted average. The time is reduced to 300 s in the real world experiments.

OL_0 is similar to [47], since it chooses the long-term goal based on the heading optimization for the last waypoint. Waypoints' headings here are not being optimized since, as explained also in [47], their method generates a very low number of intermediate steps in contrast to our one-meter-apart approach. INTER_0 represents a non-optimized approach toward the frontier, similarly to [33]. Different from them, we use a plain Shannon's entropy formulation and an RGB-D sensor to perform the mapping operation (not a 2D laser sensor). Note that the information gathered along the interpolated trajectory does not modify in any way the path or the planned robot headings, making this an ideal comparison as it also represents a greedy approach. Therefore, we consider OL_0 and INTER_0 as our baselines.

Based on the combination of activeness levels and path utility computation approach, we also performed experiments with the following methods – A (our complete approach), A_1, OL_0, OL_1, OL_1_3, OL_2, OL_2_3, and INTER_0. The numbers in the suffix signify the activeness levels enabled, with 0 referring to none of them enabled. For simplicity, if no number is specified, all active levels are enabled. The difference between A_1 and OL_1 is that on A_1 we select the bath based on the total utility of the path, while with OL_1 we *first* select the path based on the last waypoint, and then pre-optimize the waypoints' headings. We also experimented with a longer time duration version of our complete approach, named A_L, and a classic weighted sum approach named A_S. Finally, we also provide experiments with 2 additional methods, where our complete approach is used, but instead of using the plain Shannon entropy scheme ($\mu_1$), we use $\mu_2$ and $\mu_3$, as introduced in Section 2.5.2. These methods are named A_O ($\mu_2$) and A_DW_O ($\mu_3$). A recap of all the methods, the enabled active levels, and other relevant characteristics is provided in Table 2.1. Examples of the generated maps and paths with the most relevant methods are shown in Figure 2.12.

| | A_M | A | HH_M | HH | OC_M | OC | Y0_M | Y0 |
|---|---|---|---|---|---|---|---|---|
| Omnidirectional | ✓ | ✓ | ✓ | ✓ | | | | |
| Camera rotating | | | ✓ | ✓ | | | | |
| Semi-holonomic | | | | | ✓ | ✓ | | |
| Non-holonomic | | | | | | | ✓ | ✓ |
| Merged estimate | ✓ | | ✓ | | ✓ | | ✓ | |

Table 2.2: Summary of the robot configurations used in our experiments. In the row, we indicate the characteristics that the robot might possess. In the columns, we indicate the short names of the considered methods. The tick in a cell signifies that the method named has that characteristic.

### Non-Omnidirectional Robots

The considered baseline for the non-omnidirectional evaluations is A, i.e. the complete iRotate approach running on the omnidirectional robot. All approaches considered here have all three levels of activeness enabled.

We compare the baseline with our proposed merged state estimate (Section 2.4.3), which will be called A_M. Since we are modifying the provided system uncertainty, we use this evaluation to assess if there is any difference between the two methods when we do *not* move the camera. Even though we expect no difference between the two, performing this benchmark will allow us to obtain a baseline for the state estimate merging strategy.

Then, we compare our baseline against the different control alternatives introduced in Section 2.5.4. We named these:

- HH: *both* the camera and the base can rotate at the same time with the *same* maximum specific and combined speeds;
- OC: Only the Camera is allowed to rotate. The robot can only perform translational movements. This is a *semi-holonomic* scenario;
- Y0: the robot is behaving as a *non-holonomic* robot, while the camera is allowed to rotate, i.e. $v_y = 0 \ \frac{\mathrm{m}}{\mathrm{s}}$.

In the first case, we are increasing the overall complexity of the system by adding another controlled variable. However, at the same time, we expect to obtain an optimized movement since the NMPC should automatically effectively combine the two rotations. For fairness, while in principle it would be possible for us to tweak the NMPC to favor this, we do not explicitly change any parameter. With the other two scenarios, we are showing how the Active SLAM approach is usable even with other platforms by removing the strict omnidirectional requirement. Finally, we also want to study the effects that our proposed state estimate merging has on those methods. As in the baseline case, the experiments that are *not* suffixed with M represent the standard approach while the ones with M use our proposed state estimate. A summary of the characteristics for each of these configurations is reported in Table 2.2.

## 2.6.4 Simulation Experiments

Simulation results are the average over 20 successful trials of ten minutes each for each method with the same starting location on the map. Variability in trials emerges thanks to the different trajectories that the robot takes in each one of the runs. Since RTAB-Map's update rate is not fixed but varies based on the movement of the robot, the plots are presented after a bucketing procedure in time windows of 2 seconds, eventually filled with previous values.

| | BAC | | ATE RMSE | | Per meter loops | | Per meter wheels' rotation | |
|---|---|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std | mean | std |
| A | 0.771 | 0.022 | 0.059 | 0.028 | 4.965 | 0.596 | 65.781 | 4.532 |
| A_L | 0.789 | 0.019 | 0.047 | 0.023 | 5.149 | 0.597 | 68.116 | 4.944 |
| INTER_0 | 0.772 | 0.024 | 0.038 | 0.022 | 4.303 | 0.393 | 59.134 | 2.672 |
| OL_0 | 0.780 | 0.030 | 0.060 | 0.058 | 4.198 | 0.580 | 60.615 | 4.942 |
| OL_1 | 0.802 | 0.018 | 0.051 | 0.025 | 3.838 | 0.640 | 61.755 | 4.546 |
| OL_1_3 | 0.792 | 0.028 | 0.078 | 0.046 | 4.986 | 0.469 | 69.080 | 5.928 |
| OL_2 | 0.785 | 0.031 | 0.084 | 0.054 | 3.984 | 0.867 | 67.288 | 13.683 |
| OL_2_3 | 0.776 | 0.027 | 0.066 | 0.057 | 5.120 | 0.650 | 68.506 | 5.868 |
| A_S | 0.767 | 0.021 | 0.064 | 0.062 | 5.146 | 0.491 | 66.285 | 5.829 |
| A_1 | 0.776 | 0.026 | 0.063 | 0.031 | 3.211 | 0.440 | 63.705 | 6.018 |
| A_O | 0.772 | 0.030 | 0.041 | 0.011 | 5.302 | 0.587 | 66.313 | 8.356 |
| A_DW_O | 0.765 | 0.022 | 0.046 | 0.024 | 5.144 | 0.514 | 64.594 | 4.731 |

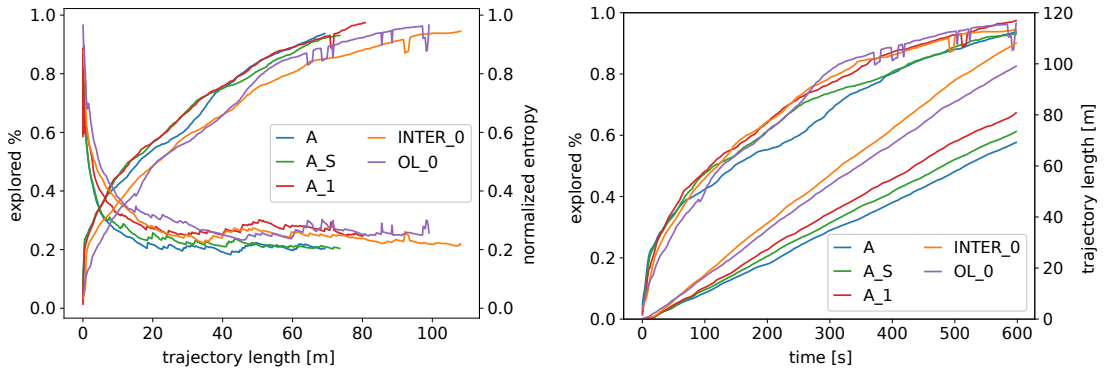
Table 2.3: Results obtained in the Café environment using the omnidirectional configuration. We report the average and standard deviation over 20 successful trials for the balanced accuracy (BAC), the absolute trajectory error (ATE, in m), the number of loop closures per meter travelled, and the amount of rotation of each wheel (in $\frac{\text{rad}}{\text{s}}$).

| | BAC | | ATE RMSE | | Per meter loops | | Per meter wheels' rotation | |
|---|---|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std | mean | std |
| A | 0.818 | 0.015 | 0.051 | 0.020 | 4.692 | 0.382 | 63.043 | 2.540 |
| A_L | 0.815 | 0.024 | 0.052 | 0.021 | 4.880 | 0.704 | 66.649 | 9.372 |
| INTER_0 | 0.823 | 0.011 | 0.034 | 0.013 | 3.791 | 0.386 | 56.721 | 2.886 |
| OL_0 | 0.808 | 0.016 | 0.063 | 0.045 | 4.057 | 0.658 | 61.687 | 20.745 |
| OL_1 | 0.816 | 0.029 | 0.052 | 0.037 | 3.402 | 0.525 | 62.389 | 9.463 |
| OL_1_3 | 0.815 | 0.021 | 0.058 | 0.023 | 4.665 | 0.426 | 68.163 | 19.557 |
| OL_2 | 0.820 | 0.019 | 0.079 | 0.062 | 3.495 | 0.470 | 66.980 | 12.378 |
| OL_2_3 | 0.818 | 0.016 | 0.067 | 0.045 | 4.849 | 0.642 | 65.048 | 9.937 |
| A_S | 0.788 | 0.026 | 0.101 | 0.077 | 4.722 | 0.526 | 67.236 | 9.299 |
| A_1 | 0.805 | 0.023 | 0.060 | 0.031 | 2.923 | 0.477 | 59.382 | 2.851 |
| A_O | 0.821 | 0.015 | 0.048 | 0.022 | 4.791 | 0.533 | 63.679 | 3.588 |
| A_DW_O | 0.806 | 0.024 | 0.047 | 0.022 | 4.738 | 0.569 | 63.564 | 3.679 |

Table 2.4: Results obtained in the Small House environment using the omnidirectional configuration. We report the average and standard deviation over 20 successful trials for the balanced accuracy (BAC), the absolute trajectory error (ATE, in m), the number of loop closures per meter travelled, and the amount of rotation of each wheel (in $\frac{\text{rad}}{\text{s}}$).

### Omnidirectional Robots — Baselines Comparisons

We first compare our full method A with A_1, i.e. only pre-plan the waypoints' view directions once, OL_0, and INTER_0 (baselines). Results, reported in Tables 2.3 and 2.4 and Figs. 2.6 and 2.7, are equivalent for both environments. Notable differences can be observed in the trajectory length: our active approaches (A and A_1) successfully explore the environment with much shorter trajectories. Indeed, OL_0 and INTER_0 ones are $\sim 33\%$ and $\sim 39\%$ longer than A, and $\sim 19\%$ and $\sim 24\%$ longer than A_1, respectively. Even considering the wheels' rotation per meter traveled by the robot, which is higher for the active methods, with the highest difference of $\sim 11\%$ between A and INTER_0, the advantage is clear. A higher amount of wheels movement for the active methods is expected since we enforce an (almost) continuous rotational movement with our approach. We can then notice that A, A_1, and OL_0 show comparable results on ATE. However, INTER_0, thanks to the smooth movement of the robot associated with the interpolated trajectory, exhibits a lower ATE.



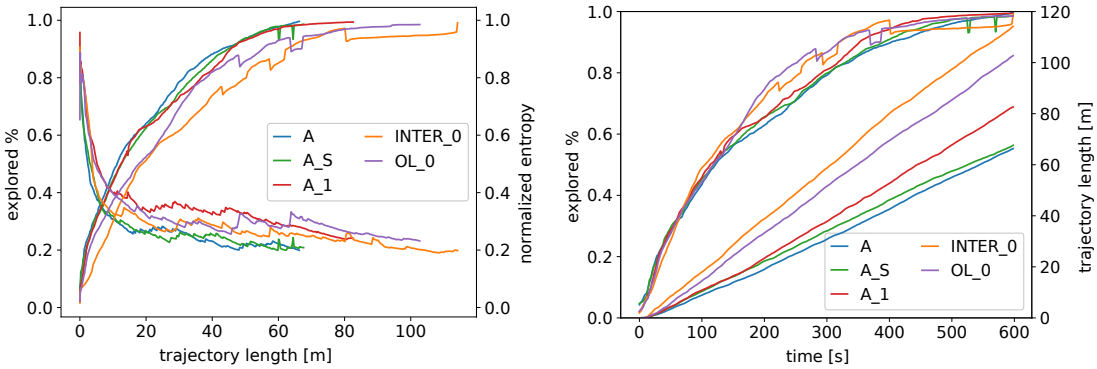

(a) Path length evolution

(b) Time evolution

Figure 2.6: Average time and path length evolution of our experiments in the Café environment using the omnidirectional configuration. Comparison of the baseline methods, i.e. INTER_0 and OL_0, with the proposed approach, iRotate (A), and its variations, A_S using a weighted sum, and A_1 enabling just the first active level.

Nonetheless, this does not translate into any clear benefit over the final entropy, exploration amount, or exploration speed. INTER_0 and OL_0 yield similar BAC to our active algorithms A and A_1. However, both show higher normalized entropy than A, despite completing the exploration earlier, thus having time to refine the result. This indicates how iRotate can better map the environment already on the first pass. Moreover, INTER_0 fails to fully explore the environment and generally shows the lowest exploration profile, despite the longest trajectories.

Regarding loop closures, we can see how A generates more of them, compared to INTER_0 and OL_0, despite a much lower path length. On the other hand, A_1 shows worse results in this metric. However, one must notice that A_1, in contrast to INTER_0 and




(a) Path length evolution

(b) Time evolution

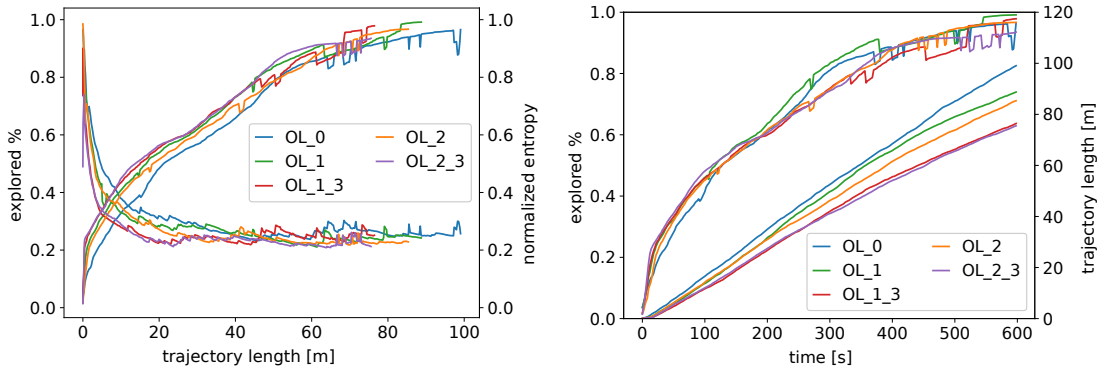
Figure 2.7: Average time and path length evolution of our experiments in the Small House environment using the omnidirectional configuration. Comparison of the baseline methods, i.e. INTER_0 and OL_0, with the proposed approach, iRotate (A), and its variations, A_S using a weighted sum, and A_1 enabling just the first active level.

OL_0, cannot navigate in an (almost) fully explored environment toward the end of the experiment. This could be one of the reasons behind this result, alongside the continuous rotational movement. Anyway, given the re-observation possibilities and the movement smoothness of both INTER_0 and OL_0, one would expect them to have higher loop closures. The fact that this is not the case further proves the benefits brought by our approach. Moreover, we will show in the ablation studies how the third level of activeness especially influences this metric, which will justify the difference of A_1 with respect to A, and how a similar insight can be obtained comparing OL_0 and OL_1. Finally, in Figure 2.6 and Figure 2.7, we can notice how, despite having a much higher exploration area per meter traveled and a continuous rotational movement, our active method A continuously keeps a much lower normalized entropy. By design, the second and (especially) the third level of activeness cause the robot to make more rotational movements. Such movements are known to increase the uncertainty in pose estimates of the robot. However, as is evident in our active approach, they still do not adversely affect robot localization.

### Omnidirectional Robots — Comparing Weighted Average and Plain Sum

Now we want to compare the effect that the weighted average has on our full approach. To do so, we consider A, which uses the proposed weighted average utility, and A_S, which applies the classic weighted sum. In the Café world, performances are quite similar, with a slightly lower ATE for the weighted average approach as can be observed in Table 2.3. Instead, when tested in the AWS environment using the weighted average scheme, there are significant gains throughout all the metrics, except for the loop closures' one (see Table 2.4). The two approaches in both the considered worlds have similar exploration profiles and trajectory lengths (see Figure 2.6 and Figure 2.7). How-

ever, the normalized entropy resulting from our approach is lower during the majority of the duration of the experiment. This result could be attributed to the ability of the weighted average to better capture the utility of the whole movement that will bring the robot to the frontier point, rather than favoring single high-utility spikes.




(a) Path length evolution

(b) Time evolution

Figure 2.8: Average time and path length evolution of our experiments in the Café environment using the omnidirectional configuration. Comparison of only OL-type methods using none (OL_0), one (OL_1, OL_2) or two (OL_1_3, OL_2_3) active levels.

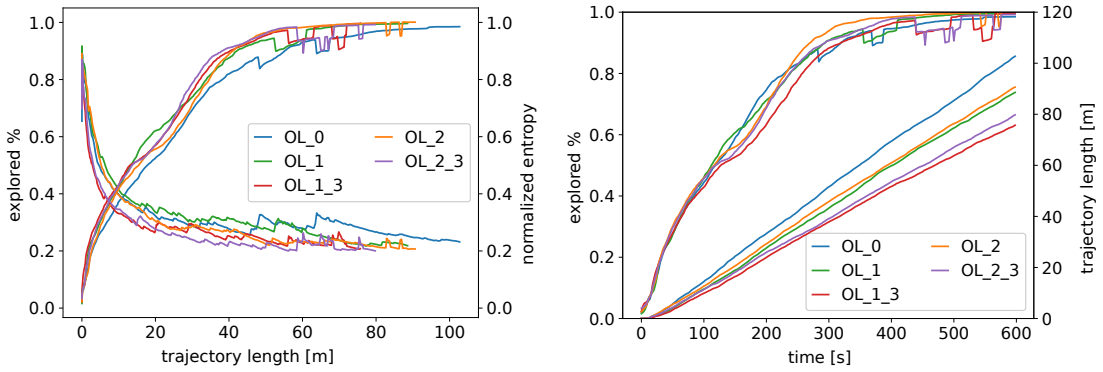
### Omnidirectional Robots — Ablation Studies

Since, as shown previously, OL_0 outperforms INTER_0, we check how our active components influence this method by comparing OL_1, OL_1_3, OL_2, and OL_2_3. In this, we apply each active component (**1**st, **2**nd, and **3**rd level of activeness) separately, leveraging the modularity of our algorithm, on top of OL_0.

We see from Figure 2.8 and Figure 2.9 that all methods outperform OL_0. Path lengths are majorly affected when the third level of activeness is enabled, with a steep reduction within the same time window. Tracking features cause a slight delay in reaching the desired orientation at every waypoint. This could be solved with more careful tuning of the weights of the third level of activeness or the NMPC.

As expected, pre-fixing the orientation along the selected path (OL_1) or doing so just for the subsequent waypoint (OL_2) has little influence over the final trajectory length since the difference in the computational effort is minimal. However, doing either greatly lowers the number of loop closures by around 15%. Overall, OL_1_3 and OL_2_3 show better performance in the majority of the metric comparisons with respect to OL_1 and OL_2. Notably, we can see how, as expected, the loop closure amounts increase when the third level of activeness is enabled. Moreover, since the two are designed to perform similar actions, OL_1_3 and OL_2_3 exhibit similar performances. However, it seems that OL_2_3 has a slight advantage by looking at BAC, ATE, and loop closure amounts.

Comparing OL_[1,2]_3 with OL_0 further proves that the continuous rotational movement introduced through the active levels does not harm the map quality or the trajectory reconstruction error.




(a) Path length evolution

(b) Time evolution

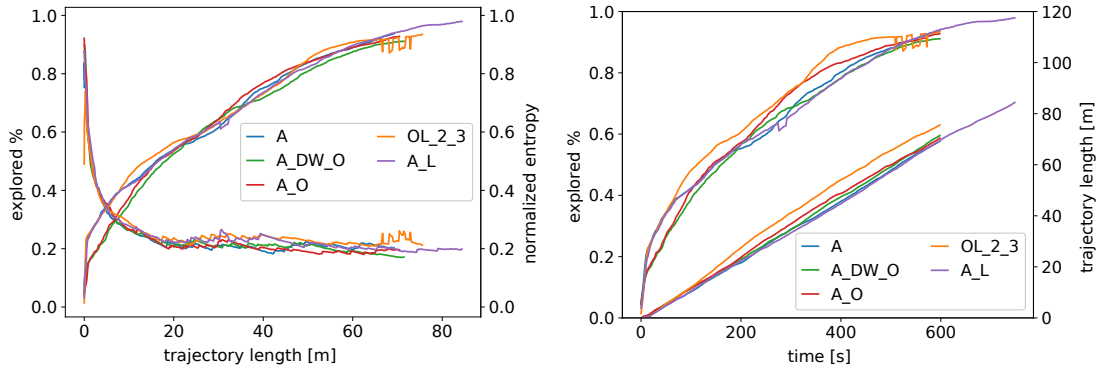
Figure 2.9: Average time and path length evolution of our experiments in the Small House environment using the omnidirectional configuration. Comparison of only OL-type methods using none (OL_0), one (OL_1, OL_2), or two (OL_1_3, OL_2_3) active levels.

### Omnidirectional Robots — Utilities Formulation Comparison

Finally, in our last comparison, whose plots are shown in Figure 2.10 and Figure 2.11, we compare OL_2_3 against our full approach A, the proposed utilities (A_O and A_DW_0), and A_L. Recall that A_L is an extended (12.5 minutes) version of our full approach A (10 minutes). We observe that all our approaches, when compared to OL_2_3, can explore the environment and do so with shorter paths ($\sim 5-12\%$). Moreover, even considering the wheels' rotation, we notice similar or better performance. This excludes a higher energy consumption due to the continuous rotation.

Our full approaches keep the normalized entropy lower both *during* and at the *end* of the experiments. The exploration speed and amount are overall comparable, but if we look at the time evolution, OL_2_3 seems to flatten earlier than expected. The ATE RMSE is, in general, better for A(_L), A_O, and A_DW_0, as compared to OL_2_3. Without loss of generalization, similar considerations can be made with respect to OL_1_3 given the similarities pointed out before. Notice also how all the OL methods are more prone to failures and the need for recovery, further showing the value of considering whole paths.

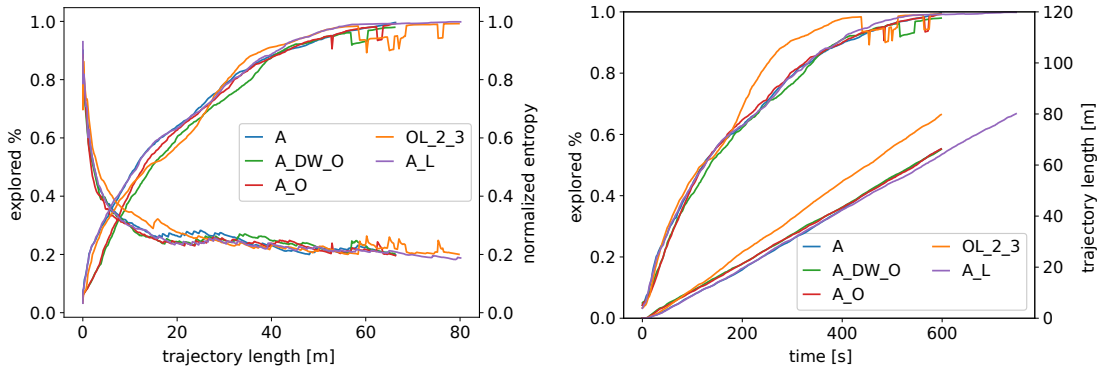
We can observe that both methods A_O and A_DW_0, as expected, are linked to lower normalized entropy and lower ATE. The presence of obstacles in the utility function seems to improve the overall system. This justifies the introduction of such a term in the utility function for an active V-SLAM approach. Since the obstacle cells are the




(a) Path length evolution

(b) Time evolution

Figure 2.10: Average time and path length evolution of our experiments in the Café environment using the omnidirectional configuration. Comparison of OL_2_3, A_L, and A using different utility formulations, i.e. an obstacle (A_O) and a dynamic weighting approaches (A_DW_O).




(a) Path length evolution

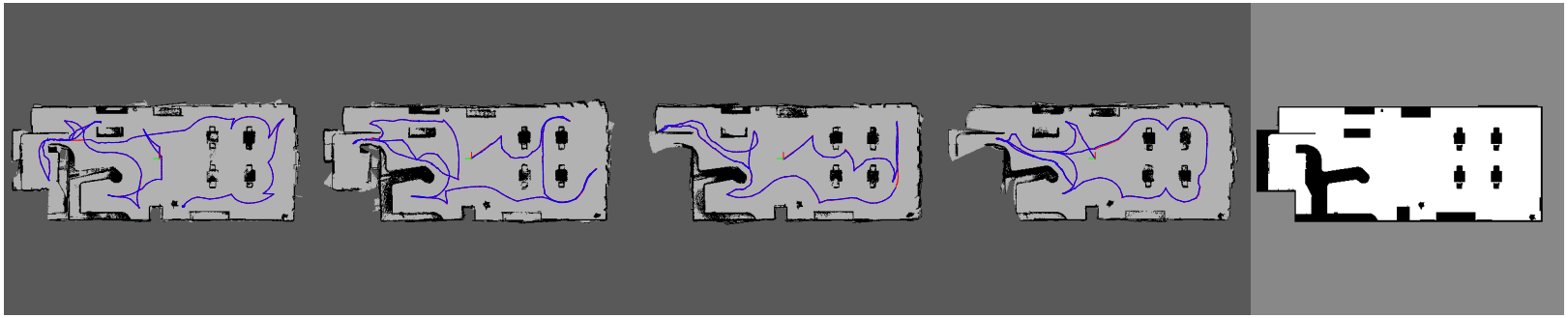
(b) Time evolution

Figure 2.11: Average time and path length evolution of our experiments in the Small House environment using the omnidirectional configuration. Comparison of OL_2_3, A_L, and A using different utility formulations, i.e. an obstacle (A_O) and a dynamic weighting approaches (A_DW_O).

ones harder to refine, and for which the system must be more vigilant, exploiting them directly in the utility computation is beneficial. Using the second formulation of the utility function ($\mu_2$, A_O) performs marginally better on both BAC and ATE against the third formulation ($\mu_3$, A_DW_O). However, A_DW_O shows a slightly lower entropy in the Café environment.

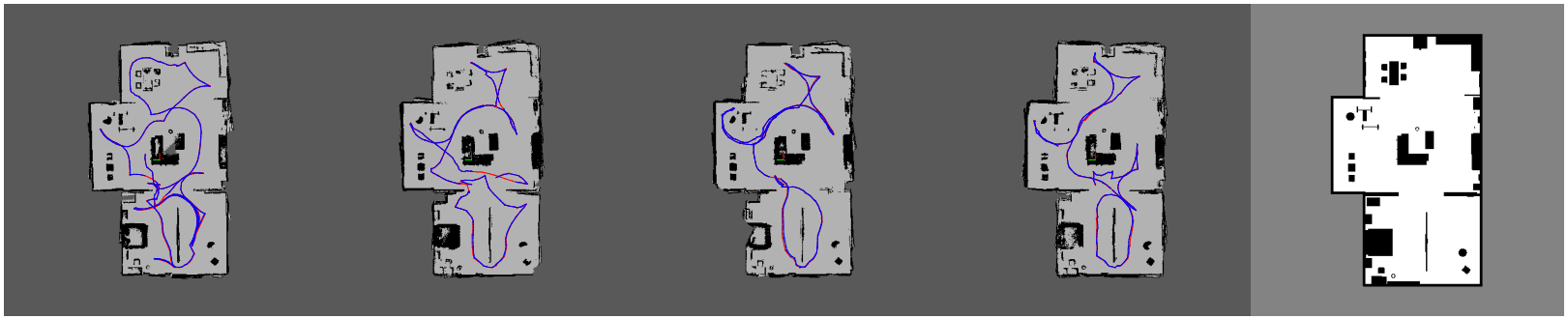
With this, we can conclude that having a dynamic weight between re-observation and exploration based on the distance from the frontier point helps in keeping a better map but slightly affects the exploration speed.

Café Environment Examples



(a) OL_0 (b) OL_2_3 (c) A (d) A_DW_O (e) Groundtruth

Small House Environment Examples



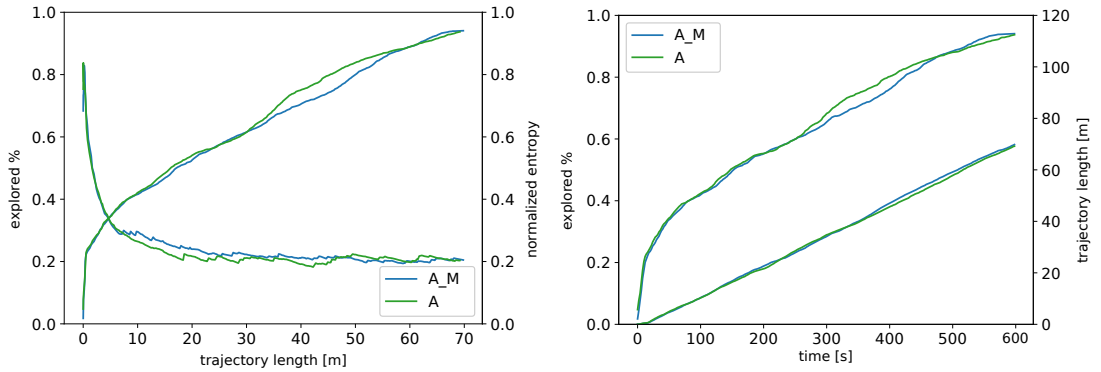
(f) OL_0 (g) OL_2_3 (h) A (i) A_DW_O (j) Groundtruth

Figure 2.12: Examples of grid maps with overlaid pose graphs for OL_0, OL_2_3, A, A_DW_O and ground truth in Small House and Café environments. In blue the connections between nodes, in red the ground truth trajectory. The images were generated with the `rtabmap-databaseViewer` tool and chosen to be representative. Loop closures are hidden for visibility's sake.

In general, while both approaches seem beneficial, using $\mu_2$ (A_O) seems to give them more advantages. Note also that, in the current implementation, the visibility range is computed on the 2D domain to keep the computation feasible. Therefore, we are 'discarding' a lot of information since many flat surfaces or cells behind low objects can potentially be seen from any given point of view.

The objective of A_L is to show that our approach has no issues in completely exploring the environment while keeping the path shorter and the same final normalized entropy. Moreover, we also find that if run for a longer time, our approach further refines its results and reaches better metrics overall. Finally, the slightly lower final explored area of our method with respect to OL_2_3 is explained by two main factors. Foremost, the computations of the first level of activeness cause a natural overhead that, despite being reduced by the frustum overlap, is still significant. An improvement in this would be considering an approximated raycasting technique rather than a complete one.

Secondly, both the third level of activeness and the computation of the next waypoint optimal heading cause a slight start and stop behavior. This can be clearly seen also in both Figure 2.8 and Figure 2.9 when comparing OL_[1,2] with OL_[1,2]_3. Anyway, a finer tuning of the parameters should solve this problem.




(a) Path length evolution

(b) Time evolution

Figure 2.13: Average time and path length evolution of our experiments in the Café environment using iRotate on our omnidirectional configuration, A, and the same method combined with the merged odometry strategy, A_M.

### Non-Omnidirectional Robots

Considering A_M and A, it is clear from the experiments performed on both environments that our proposed way of merging state estimates does not alter the results with respect to the considered baseline. There are minimal differences in the mean and standard deviation values (see Table 2.5 and Table 2.6), but nothing that indicates an overall change in performance. Thus, even though our merged odometry has a higher uncer-

tainty in all the estimates, especially in the yaw velocity, the SLAM framework remains unaffected. As expected, the final entropy, total path length, and exploration speed are similar among the two considered trials as depicted in Figure 2.13 and Figure 2.14.

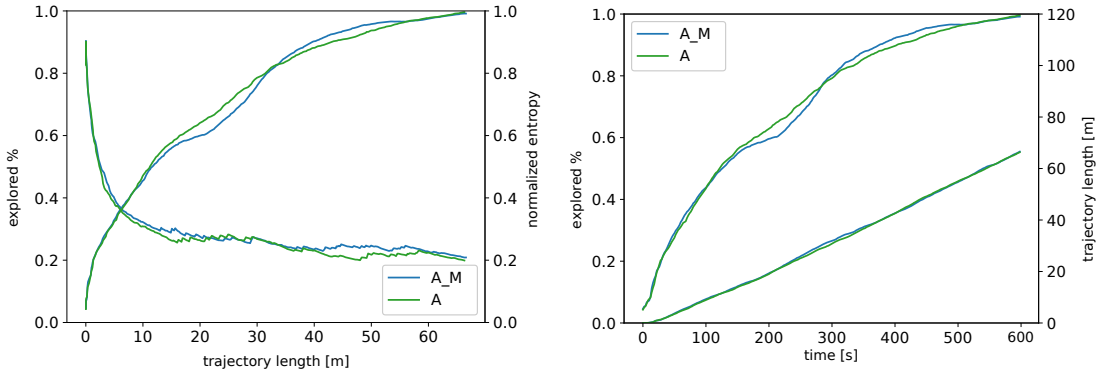
| | BAC | | Per meter wheels' rotation | | ATE RMSE | | Per meter loops | |
|---|---|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std | mean | std |
| A_M | 0.774 | 0.033 | 66.423 | 5.934 | 0.050 | 0.036 | 4.868 | 0.499 |
| A | 0.771 | 0.022 | 65.781 | 4.532 | 0.059 | 0.028 | 4.965 | 0.596 |
| HH_M | 0.793 | 0.016 | 57.927 | 5.198 | 0.025 | 0.013 | 5.180 | 0.503 |
| HH | 0.778 | 0.027 | 61.441 | 8.460 | 0.042 | 0.021 | 4.367 | 0.508 |
| OC_M | 0.784 | 0.024 | 55.494 | 5.007 | 0.027 | 0.034 | 7.608 | 0.699 |
| OC | 0.755 | 0.040 | 56.064 | 6.023 | 0.038 | 0.035 | 7.081 | 0.607 |
| Y0_M | 0.728 | 0.033 | 55.770 | 2.941 | 0.022 | 0.006 | 7.452 | 0.577 |
| Y0 | 0.705 | 0.058 | 57.774 | 6.904 | 0.031 | 0.018 | 9.486 | 1.119 |

Table 2.5: Results obtained in the Café environment comparing omnidirectional and non-omnidirectional configurations. We report the average and standard deviation over 20 successful trials for the balanced accuracy (BAC), the absolute trajectory error (ATE, in m), the number of loop closures per meter traveled, and the amount of rotation of each wheel per meter of the trajectory (in $\frac{\text{rad}}{\text{s}}$). The results are shown using the standard and the proposed merged (_M) state estimates.

| | BAC | | Per meter wheels' rotation | | ATE RMSE | | Per meter loops | |
|---|---|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std | mean | std |
| A_M | 0.810 | 0.019 | 64.514 | 5.144 | 0.056 | 0.031 | 4.634 | 0.525 |
| A | 0.818 | 0.015 | 63.043 | 2.540 | 0.051 | 0.020 | 4.692 | 0.382 |
| HH_M | 0.825 | 0.017 | 53.635 | 3.016 | 0.027 | 0.013 | 4.788 | 0.398 |
| HH | 0.808 | 0.014 | 54.618 | 4.872 | 0.038 | 0.016 | 3.659 | 0.492 |
| OC_M | 0.832 | 0.012 | 50.232 | 2.242 | 0.026 | 0.017 | 7.447 | 0.742 |
| OC | 0.811 | 0.018 | 51.257 | 3.002 | 0.030 | 0.014 | 6.597 | 0.716 |
| Y0_M | 0.785 | 0.029 | 56.059 | 1.861 | 0.032 | 0.009 | 7.334 | 0.532 |
| Y0 | 0.742 | 0.047 | 61.938 | 12.097 | 0.035 | 0.022 | 9.642 | 1.422 |

Table 2.6: Results obtained in the Small House environment comparing omnidirectional and non-omnidirectional configurations. We report the average and standard deviation over 20 successful trials for the balanced accuracy (BAC), the absolute trajectory error (ATE, in m), the number of loop closures per meter traveled, and the amount of rotation of each wheel per meter of the trajectory (in $\frac{\text{rad}}{\text{s}}$). The results are shown using the standard and the proposed merged (_M) state estimates.

When the rotation of the camera is introduced, i.e. in HH, OC, and Y0, there are some interesting results. First of all, if we compare the two state estimate definitions among all experiments, we can see how the classical way of considering the camera rotation as perfectly known is worse. This can be seen in both the environments considered, looking either at Tables 2.5 and 2.6 or Figures 2.15 and 2.16. In general, there is an improvement of the BAC between $\sim 1.5\%$ and $\sim 5\%$, comparing the merged state estimate experiments against the vanilla counterparts. ATE is also reduced (between $\sim 10\%$ and $\sim 40\%$) and loop closures are more frequent (between $\sim 10\%$ and $\sim 30\%$) except in the non-holonomic case. The most significant difference can be seen once we compare the entropy evolution. In this, when we compare the merged state estimate approach with respect to the vanilla variations for all the platforms (HH, OC, and Y0) we can see how the initial evolution is comparable in the first $\sim 20$ m of the trajectory and, after this point, they diverge. The bigger difference can be observed in both the couples HH_M-HH and OC_M-OC. Finally, Y0_M shows the smallest difference in terms of normalized entropy but also has the shortest traveled path. The Active SLAM method indeed has influence with respect to entropy evolution, but, since the exploration profiles are equivalent between all the alternatives with respect to their vanilla counterparts, we can safely infer that the difference in the performance is not due to this building block. Most likely, the main problem here is the robot registering an orientation for the camera and, when loop closures happen, the drift and the uncertainty of that are not considered and have a clear effect on the overall final estimation, i.e. the exact issue we sought to tackle.




(a) Path length evolution

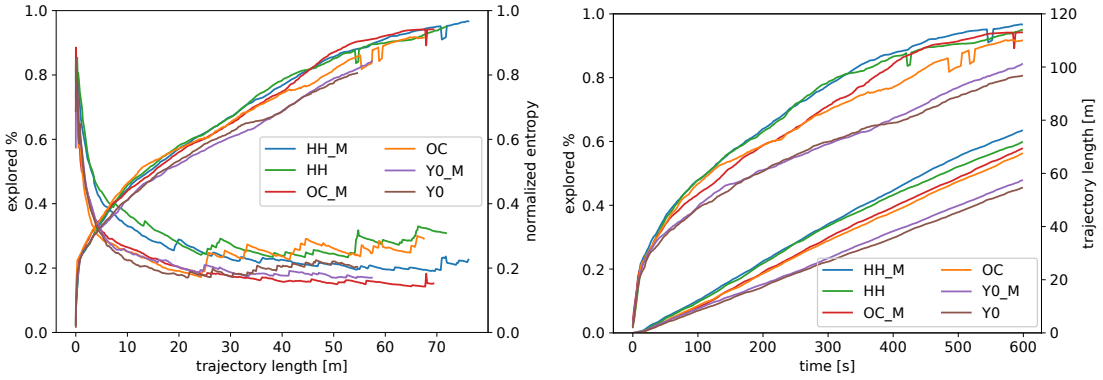
(b) Time evolution

Figure 2.14: Average time and path length evolution of our experiments in the Small House environment using iRotate on our omnidirectional configuration, A, and the same method combined with the merged odometry strategy, A_M.

Comparing the results obtained with these three platforms with respect to the omnidirectional baseline, we notice how our iRotate approach is indeed usable and can generalize well to non-omnidirectional robots. We first highlight the overall energy reduction that the approach proposed in this chapter brings. When we rotate both the robot and

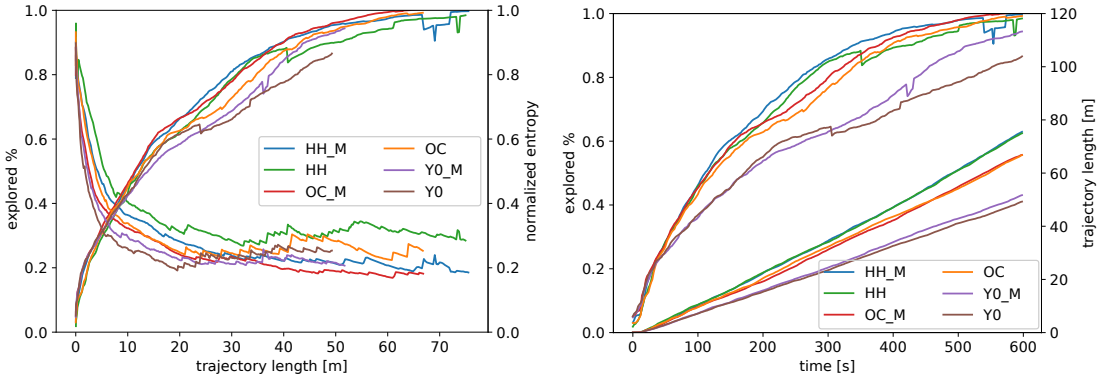
camera, even if we maintain the coupled maximum rotational speed, the system achieves $\sim 20\%$ less total wheel rotation per meter traveled. This implies that the rotation of the camera is capable of reducing the rotation of the robot, further optimizing its movements.

This is indeed reflected in every considered metric and in the fact that we perform $\sim 10\%$ longer paths within the same time window, suggesting a better optimization and faster reaching of the desired waypoints' orientations. Anyway, this does not result in longer paths since the exploration profiles are comparable.




(a) Path length evolution (b) Time evolution

Figure 2.15: Average time and path length evolution of our experiments in the Café environment using different non-omnidirectional configurations (HH, OC, Y0) and the merged odometry strategy (_M).




(a) Path length evolution (b) Time evolution

Figure 2.16: Average time and path length evolution of our experiments in the Small House environment using different non-omnidirectional configurations (HH, OC, Y0) and the merged odometry strategy (_M).

Moreover, we can see how the semi-holonomic robot (OC) is capable of obtaining even better performance with a lower normalized entropy while keeping a low wheel rotation amount. In terms of ATE and BAC, it is quite similar to the HH method, underperforming slightly in the Cafè environment while getting better results in the Small House. In this case, avoiding the rotation of the base of the robot brings more loop closures, probably linked to the fact that it is easier for the system to match 2D laser scans.

Altogether, these factors indeed help the system in achieving these results. One should also note that, in this scenario, the RTAB-Map system actually 'thinks' that the robot *is* rotating during the experiment, further proving the goodness of both camera rotation (in terms of efficiency) and our merged estimation strategy. It is worth noticing that without the camera rotation, a holonomic robot would have a very inefficient start and stop behavior to follow the desired heading that continuously changes along the trajectory.

Finally, the non-holonomic robot (Y0_M and Y0) is the worst in terms of BAC. This is probably linked to the fact that it is capable of exploring a much lower amount of area, which lowers the balanced accuracy performance. In this case, we should also note that we did not optimize the trajectory or waypoints for a non-holonomic robot to keep the comparison as fair as possible. Performing such optimization would be beneficial, especially for the total explored area. The higher number of loop closures of Y0 can be linked to the slow movement and the lower amount of area seen despite the comparable path length.

### 2.6.5 Real Robot Experiments

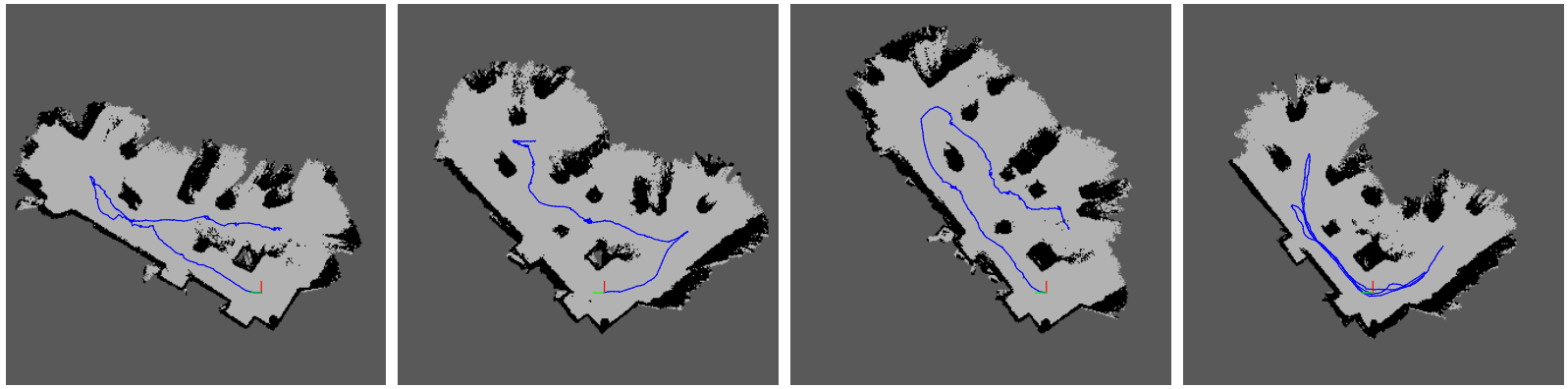
The experiments with our real robot consist of five runs of five minutes each in a custom office-like environment that is depicted in Figure 2.1. For safety reasons, the speed of the robot has been limited to 0.25 $\frac{\text{m}}{\text{s}}$. For this set of experiments, the ground truth of both the robot position and the map was not available and, therefore, the analysis can only be done with respect to the area explored, the distance traveled, the final map entropy, and the number of loop closures.

#### Omnidirectional Robots

We report the results obtained in the real environment using an omnidirectional configuration in Table 2.7 and Figure 2.18, while a representation of the constructed maps is shown in Figure 2.17.

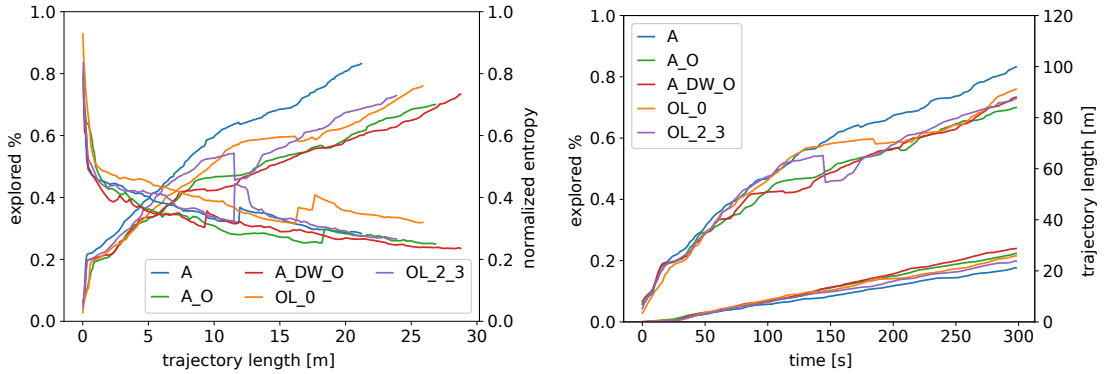
The results obtained in the real world, in general, have the same trend as our simulation experiments, further validating our approach. Most importantly, as seen in Figure 2.18, our complete approach A achieves the shortest path length and explores the maximum area while keeping a fairly low entropy. Both A_O and A_DW_O, i.e. the methods using our new proposed utilities, are the best ones considering the final normalized entropy and the average loop closures.

Real world Environment Examples



(a) OL_0 (b) OL_2_3 (c) A (d) A_DW_O

Figure 2.17: Examples of grid maps with overlaid pose graphs for OL_0, OL_2_3, A, A_DW_O in real world settings. The images, generated with the `rtabmap-databaseViewer` tool, are chosen to be representative. In blue are the connections between nodes (ground truth not available). Loop closures are hidden for visibility.




(a) Path length evolution

(b) Time evolution

Figure 2.18: Average time and path length evolution of our experiments in a real world environment using a real omnidirectional robot with our complete approach (A), its variations using different utility formulations (A_DW_O and A_O), and the best OL variations (OL_0 and OL_2_3).

| | Area | | Normalized Entropy | | Per meter Loops | |
|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std |
| A | 83.257 | 13.239 | 0.283 | 0.054 | 2.228 | 1.488 |
| A_O | 69.993 | 5.205 | 0.249 | 0.042 | 3.852 | 1.434 |
| A_DW_O | 73.344 | 1.371 | 0.235 | 0.015 | 3.051 | 1.611 |
| OL_0 | 76.037 | 5.774 | 0.319 | 0.101 | 2.397 | 1.057 |
| OL_2_3 | 72.924 | 18.419 | 0.266 | 0.041 | 1.016 | 0.384 |

Table 2.7: Results were obtained in a real world environment using the omnidirectional configuration. We report the average and standard deviation over 5 successful trials for the area explored ($m^2$), the entropy normalized on the explored area at the end of the experiment, and the number of loop closures per meter traveled.

Furthermore, we can see in Table 2.7 that OL_0 achieves more loop closures than OL_2_3, despite having a much higher normalized entropy. This is slightly in contrast with the simulation results. Finally, even if the real robot experiments are less statistically significant than the simulation ones, they indicate that our three levels of activeness improve the final results in all the compared metrics. Lastly, it must be noted that we get longer paths with A_O and A_DW_O. This is most likely linked to the robot's very low velocity. While these two methods facilitate longer and more informative paths, the OL-approaches usually focus on the nearest frontier. In the A_O and A_DW_O methods, the robot suffers from several start-and-stop instances and the continuous necessity to recompute goals.

### Non-Omnidirectional Robots

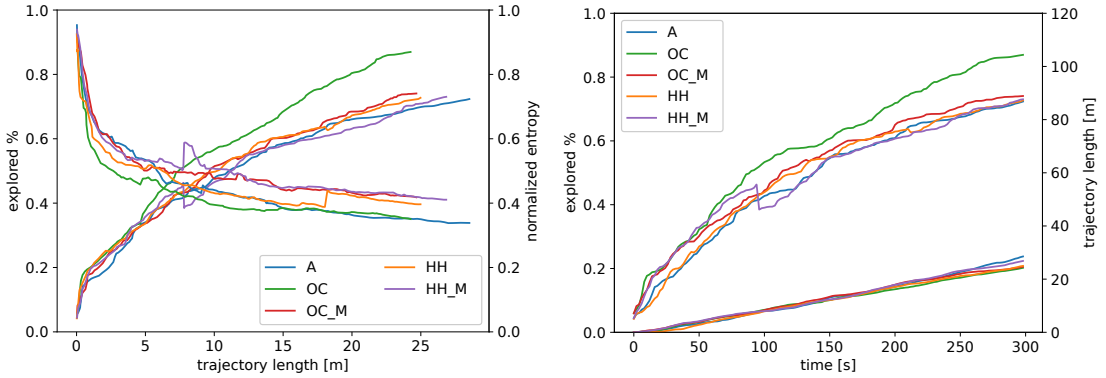
The real robot environment used for these experiments is not the same as the one used in the previous section but has been reconstructed similarly. The experiments' results are reported in Table 2.8 and Figure 2.19. Notably, the encoder necessary to read and refine the camera rotational movement through the EKF was not available. Moreover, the PID controller used caused a considerable delay any time it had to initiate the rotation of the camera due to inertia and friction forces. Moreover, since the wheels' encoder readings were unavailable, we cannot directly distinguish between the camera and the robot's rotations. To approximate that, we computed the total turn of the robot base and the camera using the SLAM graph nodes' information. Note that this differentiation is there only in the case where the merged estimate is not used. Also, unlike the simulation, this measurement is not fully indicative of the total rotation during the experiment. Therefore, in Table 2.8, the estimated rotation amounts are expressed separately for all methods in which the merged state estimate is not used. In the other cases, the *merged* one is being reported in the 'Robot's rotation' row. Finally, while the linear speed is the same as the previously run experiments, i.e. 0.3 $\frac{\text{m}}{\text{s}}$, the angular speed has been increased to 1 $\frac{\text{rad}}{\text{s}}$ (from 0.5) which caused the robotic platform to perform more harsh and inaccurate rotational movements.

| | | A | OC | OC_M | HH | HH_M |
|---|---|---|---|---|---|---|
| Area | mean | 72.348 | 86.973 | 74.072 | 72.729 | 73.033 |
| | std | 8.710 | 7.397 | 12.473 | 7.903 | 12.496 |
| Robot's rotation | mean | 1.675 | 0.228 | 3.070 | 1.684 | 2.804 |
| | std | 0.250 | 0.032 | 0.313 | 0.362 | 0.960 |
| Camera's rotation | mean | 0.227 | 2.580 | — | 1.419 | — |
| | std | 0.023 | 0.456 | — | 0.839 | — |
| Path Length | mean | 28.526 | 24.255 | 24.677 | 24.978 | 26.846 |
| | std | 2.516 | 3.093 | 2.231 | 6.794 | 4.128 |
| Loops per m | mean | 0.370 | 0.920 | 2.200 | 0.820 | 1.570 |
| | std | 0.470 | 0.900 | 1.530 | 0.650 | 2.300 |

Table 2.8: Results obtained in a real world environment comparing omnidirectional and non-omnidirectional configurations. We report the average and standard deviation over 5 successful trials for the area explored ($\text{m}^2$), the total robot's and camera's rotations per meter of the trajectory ($\frac{\text{rad}}{\text{s}}$), the path length (m), and the number of loop closures per meter traveled.

The first thing we notice when analyzing the results is the minor disparities observed between the results in Table 2.8 and those obtained from the previous real-world testing (Table 2.7). Those are to be attributed to the diverse environment configuration, which resulted in different paths and observed features, and to the estimation of the camera's

rotation relative to the base of the robot. Those two main factors combined caused a lower number of loop closures and a higher map entropy. Moreover, as shown in Table 2.8, we can see that the total amount of area explored by A and the other platforms is comparable, *but* the non-omnidirectional configurations achieve it with much shorter paths. Notably, OC has a higher exploration amount, $\sim 20\%$, with respect to the baseline A. This, combined with the shorter path and the fact that the robot is *not* rotating in OC, shows the clear benefits of our approach also in the real world. OC is less affected by the PID delay because the continuous rotation input from the NMPC causes the camera to rarely stop. The ease of control and the fact that the RTAB-Map is not aware of the noise related to the camera orientation are linked to the outperforming exploration speed of OC.




(a) Path length evolution

(b) Time evolution

Figure 2.19: Average time and path length evolution of our experiments in a real world environment using our complete approach with an omnidirectional platform (A), its non-omnidirectional variations (HH and OC), using both the classical and the new merged (_M) state estimate.

Comparing the approximated total estimated rotation, even when the system is forced to not rotate one of the two components (i.e. A, OC), it estimates $\sim 0.23$ radians per meter traveled of rotation, while, ideally, this number should be $\sim 0$. Thus, it is clear how the system has, in this scenario, more difficulties in estimating the map (higher entropy) and recognizing places (lower loop closures) with respect to our previous results. HH base's rotation amount is comparable to A due to the aforementioned PID's delay that had to be compensated by the robot's rotational movement. Despite that, HH shows some benefits even in these brief experiments by having shorter paths with respect to A but with higher exploration speed. Thus, rotating both the camera and the robot, apart from having a theoretical energy-saving advantage and despite the higher control complexity, can bring a benefit to the overall system. We can also notice how the rotational movement per meter is higher in both OC and HH, showing a more effective control strategy.

Finally, it is interesting to notice that, despite the missing encoder, the method of merg-

ing the state estimates seems promising. We can see from Table 2.8 that the loop closures benefit from this kind of representation, even if the added noise degrades the estimation of the map in terms of normalized entropy by a small amount ($\sim$ 5%, see Figure 2.19). The bigger difference can be seen comparing OC_M and OC, but, as mentioned before, the performance of OC could be a bit misleading on its own. Overall, given some limitations of our hardware platform, these results show both the usability of our approach with different robotic platforms, the benefits of rotating both the camera and the robot simultaneously, and the goodness of the proposed merging strategy for state estimates.

## 2.7 Conclusions

In this chapter, we presented a novel Active V-SLAM method, iRotate. Our approach consists of the combination of three levels of activeness. In the first level, the robot selects goals and decides on the most informative paths to them. The second level re-optimizes waypoints along the path in a way such that the real time updated map information is continuously exploited. The third level of activeness ensures that the robot orientations have maximum visual feature visibility. This eventually ensures better localization accuracy, in addition to lower map entropy. One of the most significant results of our approach is that it requires up to 39% less path traversal to obtain similarly good coverage and lower map entropy compared to methods that are tuned versions of the state-of-the-art approaches [33]. We demonstrated the efficacy of our method both in simulation and in real-world scenarios.

We have also shown how we can apply iRotate to different robotic platforms. While the mapping performance is not being affected much by *which* robotic platform is being used, it has been shown how an independent camera rotation yields lower energy consumption, comparable or higher amount of loop closures, lower ATE, and shorter path lengths. Moreover, the proposed merged state estimate has shown promising results in both simulation and real world experiments, despite the limitations of the latter ones. Note that using a 'full' holonomic robot could further optimize the energy consumption in that scenario. A small in-place rotation could indeed align the robot with the movement direction, thus avoiding the rotation movement of one of the wheels. Concerning HH, a diversified control strategy might be applied by seeking energy efficiency with the base and reaching the desired orientation with the camera. This could further reduce the overall energy consumption of the system.

# Chapter 3

# Photorealistic Simulations: Bridging Virtual and Real Worlds

Photorealistic synthetic data and novel rendering techniques significantly advanced computer vision research. However, datasets focused on computer vision applications cannot be easily applied to robotics because they typically lack physics-related information. This, combined with the difficulties of realistically simulating dynamic worlds and the insufficient photorealism, flexibility, and control options of common robotics simulation frameworks, hinders progress in (visual-)perception research for autonomous robotics. For instance, most V-SLAM methods are passive, developed under a (semi-)static environment assumption (including the just presented iRotate), and evaluated on just a limited number of pre-recorded datasets. To address these challenges, we present a highly customizable framework built upon NVIDIA Isaac Sim for Generating Realistic And Dynamic Environments - GRADE. GRADE leverages Isaac's rendering capabilities, physics engine, and low-level APIs to populate and manage realistic simulations, generate synthetic data, and evaluate online and offline robotics approaches, including Active SLAM and heterogeneous multi-robot scenarios. Within GRADE, we introduce a novel experiment repetition approach (Section 3.4.3) that allows environmental, sensors, and scenario variations of previous simulations within physics-enabled environments. This enables flexible and continuous testing, development, and data generation, with fine control of the underlying conditions without necessarily changing the fundamental aspects and progress of the repeated experiment itself. We then use GRADE to collect a high-fidelity and richly annotated synthetic video dataset of indoor dynamic environments (Section 3.5). With that, we train detection and segmentation models for humans and successfully address the syn-to-real gap, paving the way for our ZebraPose work (Chapter 5). In Chapter 4, we will leverage GRADE to benchmark state-of-the-art dynamic V-SLAM algorithms, revealing their limitations in tracking times and generalization capabilities, and evidencing that top-performing deep learning models do not necessarily lead to the best SLAM performance. This will lay out the basis of our Dynamic V-SLAM approach - DynaPix - which will be introduced in Section 4.4. The rest of this chapter is organized as follows: we will first present the problem in Section 3.1. In Section 3.2, we review the related work about i) robotics simulators, ii) indoor environments datasets, and iii) simulated animated humans. In Section 3.3, we introduce and detail the main components of the proposed framework, GRADE, in four main aspects: i) asset preparation, ii) robot creation and control, iii) simulation management, and iv) post-processing tools. Section 3.4 is then dedicated to exemplifying four different case studies implemented with GRADE. In there, we also introduce our novel experiment repetition approach. Following that, in Section 3.5 we provide details of the data generation procedure and the datasets released with this work, which will be used in our syn-to-real (Section 3.6) and Dynamic V-SLAM (Section 4.3) experiments. The experiments and results reported in this chapter in Section 3.6 include an analysis of the experiment repetition tool and the syn-to-real learning performance using the generated data on the tasks of human detection and segmentation in indoor environments. Finally, we report our conclusions and final remarks in Section 3.7. GRADE's code and generated data are provided as open-source at https://grade.is.tue.mpg.de.

## 3.1 Introduction

Directly conducting robotic experiments in the real world to test and validate new approaches can pose safety risks. Unforeseen failures of methods and sensors, corner cases, or loss of control of the autonomous robot platform may easily lead to damages or injuries. This problem is further exacerbated when the robot relies on exteroceptive sensors: noise, domain shifts, and the lack of formal performance guarantees or uncertainty quantification in most deep learning (DL) models can make behaviors unpredictable. Surely, pre-recorded datasets have been widely used to develop and evaluate new approaches. Those designed for computer vision research, such as [129, 187, 229], are visually appealing due to the use of real-world images or advanced rendering engines like Blender[1] and Unreal Engine (UE[2]). However, the absence of basic sensor readings (e.g. IMU, LiDAR), sensor states (e.g. position, orientation, velocity), and (in general) temporal information restricts their applicability in robotics contexts, where physics- and time-related information is necessary.

At the same time, gathering ground-truth data for robotics poses significant challenges. In addition to the safety risks, accurately measuring physical quantities can be intricate, time-intensive, and impractical. Even when feasible, it demands costly specialized sensors requiring rigorous calibration and synchronization — both among themselves and with other hardware like cameras — making the process particularly difficult. For example, despite the centrality of the problem to higher-level tasks, there exist only a handful of real-world SLAM benchmark datasets with ground-truth information, especially for dynamic environments [26, 27, 74, 211]. Furthermore, relying solely on already available datasets for real-world robotic applications is not straightforward due to differences in robot form factors (e.g. sensor placement), sensor configurations (e.g. camera focal length, sensor publishing frequency), and different noise models. This requires researchers to rely on their own data or a limited selection of datasets, which can overfit specific scenarios and hinder reproducibility, robustness, and broader deployment. Finally, datasets are "fixed" in time, as one cannot introduce new sensors or modify recorded environmental conditions (e.g. removing dynamic elements or changing lighting conditions) after data collection. This limitation further restricts their usage, making them inadequate for evaluating methods that require real-time dynamic decision-making, such as obstacle avoidance, environment interaction, or Active SLAM.

Therefore, to overcome the static nature of pre-recorded datasets and facilitate the safe development and evaluation of robotics methods, simulation engines such as Gazebo [110] and WeBots [146] are widely used. However, with those, it is often challenging to i) obtain and finely control realistic animated rigid and non-rigid assets, ii) simulate dynamic environments, iii) customize and control the simulation engines, and thus iv) bridge the gap between simulations and the real world. Moreover, the low visual

[1] http://www.blender.org
[2] https://www.unrealengine.com

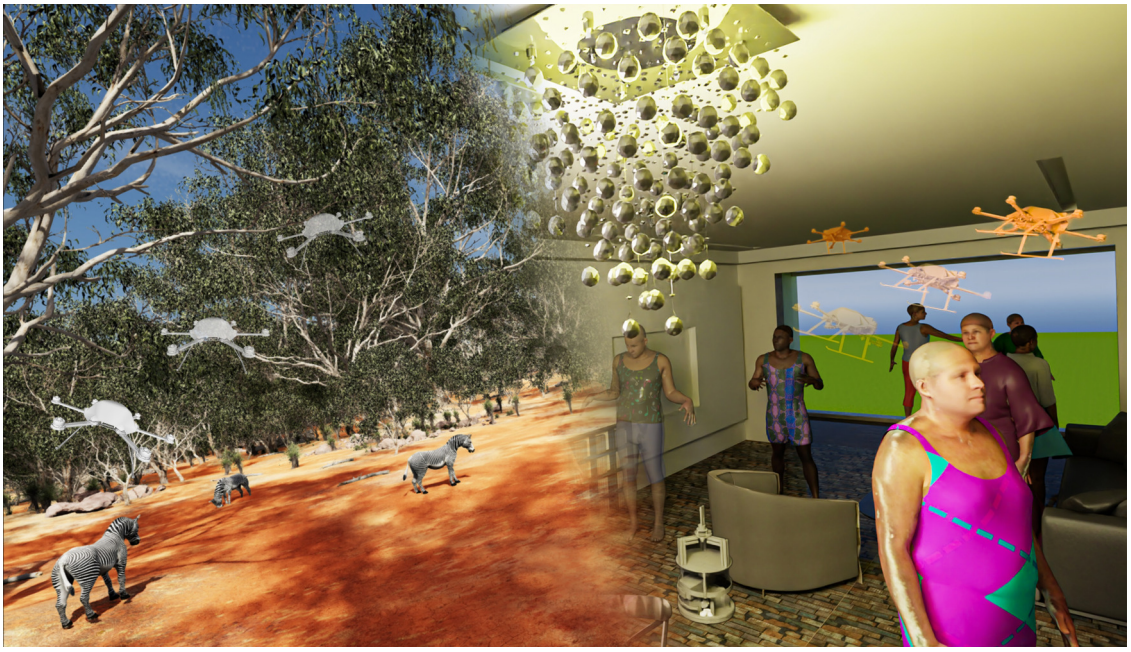


Figure 3.1: Example scenes generated by GRADE with overlaid drones. This figure shows two different scenarios simulated with GRADE. [Left] An outdoor savanna environment with manually placed animated zebras (the same scene is also present in Figure 3.3e). [Right] An indoor environment with animated humans (also present in Figures 3.3b to 3.3d, 3.8 and 3.9). The UAVs and the UGV (in the right image only) are captured in the scene itself from an external point of view and then manually overlaid to highlight their movement in the environment (similar to Figures 3.5a and 3.5b). The savanna world and the animated zebras are freely available assets we obtained from the Unreal Engine and SketchFab marketplaces. The 3D-Front indoor environment (on the right), populated with animated SMPL humans from the Cloth3D dataset, resembles a scene that we used to generate the data used by our neural networks and evaluate Dynamic V-SLAM approaches. This indoor scene has been automatically created using our data generation procedure, including the placement of the dynamic assets. The scene on the left is used only to evaluate the generalization of the method to different environments and assets. Scenes similar to the one on the right are used both for training our syn-to-real detection and segmentation models in Section 3.6.2 and to evaluate Dynamic V-SLAM approaches in Section 4.3.

fidelity of many robotics simulators exacerbates the sim-to-real transfer gap. As a result, most robotics research is conducted in highly controlled scenarios with several simplifying assumptions. Indeed, although (non-)rigid moving objects are common in real life and significantly affect vision-based localization or navigation methods, many current approaches still assume a (semi-)static world [1, 26, 191] or use simplified dynamic environments composed of basic 3D shapes [197, 241]. Overall, this is detrimental to developing and evaluating robotic systems, which depend heavily on visual perception and must operate reliably in realistic, dynamic environments. Thus, it is imperative to have a simulation framework that incorporates at least the following key characteristics: i) physical realism - to correctly simulate dynamics, ii) photorealism - to reduce the perception gap, iii) low-level software access - to allow full control, and iv) the capability to simulate dynamic entities - to enable widespread deployment. Integration with ROS, although optional, would further encourage wider adoption of such a framework, given its common use in developing higher-level software that functions simultaneously in both simulation and real robots. In short, an easily controllable simulation that closely resembles the real world with a minimal sim-to-real gap is essential to enable quick and reliable real-world deployment of robotic methods.

To address these issues, we present a solution for Generating Realistic And Dynamic Environments — GRADE. GRADE is a flexible, controllable, customizable, photorealistic, ROS-integrated pipeline that can produce visually realistic data in physically enabled environments. GRADE is built *directly* upon NVIDIA Isaac Sim[3], leveraging its rendering and PhysX engines. In contrast to existing methods, GRADE is not merely a new benchmarking approach or an application-specific platform; instead, it provides an open system that can be easily expanded towards different research goals. We make available a set of functions, tools, and case studies that serve as an entry point with low-level access to Isaac Sim's capabilities. This enables researchers to easily customize simulations to meet their needs and further bridge the gap between simulation and real-world scenarios. A sample image generated with GRADE, displaying diverse subjects, environments, and overlaid robots, is shown in Figure 3.1.

Here, we introduce different case studies highlighting GRADE's versatility, including (unconstrained) data collection (Section 3.4.1), Active V-SLAM, and heterogeneous multi-robot simulations (Section 3.4.2). Among those, we also introduce a novel experiment-repetition procedure (Section 3.4.3) that enables the exact reproduction of simulation trials under varying environmental conditions and adjustments to the robot's settings and equipment, all within a physically controlled environment. We then use GRADE to automatically generate a novel extensive dataset, which we release publicly, collected in indoor dynamic environments (Section 3.5). We employ this dataset, consisting of more than 615K frames, to assess the visual realism of the simulation through extensive experiments on human detection and segmentation with YOLOv5 [102] and Mask R-CNN [84], demonstrating strong sim-to-real performance. Indeed, our results

[3] https://developer.nvidia.com/isaac-sim

highlight that pre-training with GRADE-generated data enables models to outperform the baseline on the COCO [129] dataset. Moreover, training with synthetic images alone achieves results comparable to the baseline, even without *any* fine-tuning, on the TUM RGB-D [211] dynamic sequences.

## 3.2 Related Works

In this Section, we present the state-of-the-art of robotics simulators (Section 3.2.1) and of the main components of our data generation procedure, namely indoor environments datasets (Section 3.2.2) and simulated animated humans (Section 3.2.3).

### 3.2.1 Robotics Simulators

Gazebo [64, 190] is one of the go-to robotics simulator choices thanks to its simplicity, reliable physics engine, and ROS [171] integration. However, it lacks photorealism and full simulation control, supports only a limited range of assets and worlds, and struggles to deliver real-time performance, even for single robots in simple worlds with minimal rendering necessities [2, 155, 163]. For example, Gazebo has been used in the context of bridging the perception gap between real and simulated environments by Bayraktar et al. [12]. They introduced ADORESet, a hybrid image dataset. The combination of real and synthetic images in ADORESet aims to improve the robustness of computer vision systems by leveraging the strengths of both data types. However, contrary to what we will show in our results, their data does not generalize to the real world as shown in Table 6 of their paper. This is likely due to the low realism of Gazebo. Alternatives such as BenchBot [220], AirSim [196], Ai2Thor [111], iGibson [198], AI-Habitat [192], and Sim4CV [151] all lack essential features such as low-level simulation controllability, ROS integration, or realistic physics and visual fidelity. Additionally, some simulators model *only* rigid objects [110, 192] or do not include dynamic assets, as this would introduce challenges for their correct placement, management, or generation. Finally, computer-vision-focused simulators like Sim4CV or Kubric [78] are difficult to adapt for robot simulations, as they lack many robotics-specific sensors, accurate physics simulations, and support for robotics platforms and ROS.

Among robotic simulators, AirSim seeks to bridge the visual realism gap by building on top of Unreal Engine. However, it provides limited APIs, lacks support for custom or multiple heterogeneous robots, and does not enable direct joint control. Its native integration with ROS is also loose and incomplete[4,5,6]. Ai2Thor, designed primarily for AI and visual tasks, is not customizable for general robotics purposes, as it lacks essential

[4] https://github.com/microsoft/AirSim/discussions/3556
[5] https://github.com/mitchellspryn/UrdfSim
[6] https://github.com/microsoft/AirSim/issues/3630

sensor interfaces, such as IMUs and LiDARs[7,8], and does not offer native ROS support[9]. Similarly, AI-Habitat is mainly focused on navigation tasks. Although a community plugin exists for ROS integration [39], it is an external package with limited support rather than a core feature, e.g. currently it does not support ROS2 or Habitat2.0[10].

Notably, Habitat3.0 [167], which was released recently and concurrently to GRADE, allows the integration of **only** human animations based on the SMPL-X model [159], resulting in a much narrower scope compared to our approach. Moreover, both Habitat2.0 [218] and 3.0 support only rigid objects, limiting the realism of the simulation, a problem not present in Isaac Sim. iGibson [198] focuses instead on interactive environments with enhanced characteristics like temperature or wetness. However, its initial version was not customizable or expandable to different tasks, such as navigation and localization, and offered limited visual realism. To solve the realism issue, it was recently ported to Isaac Sim as OmniGibson [121]. Their motivations for using Isaac Sim align closely with ours, i.e. recognizing the limitations of other simulation frameworks, leveraging Isaac's physical and visual realism, and allowing for higher flexibility. However, different from GRADE, OmniGibson primarily functions as a plug-in to add characteristics such as temperature and dirtiness control, focusing on indoor activities, without any simulated dynamic humans[11], and it was published at the same time of GRADE.

Recent simulation platforms have also introduced novel approaches to robotic learning and environment modeling. Genesis[12], a newly proposed physics engine, aims to support general-purpose robotics, embodied AI, and physical AI applications. While details remain unpublished, it appears to offer diverse material simulation, a lightweight robotics environment, and high-fidelity rendering. If integrated with ROS, it could rival Isaac Sim and GRADE. Similarly, RoboGen [236] employs foundation and generative models in a propose-generate-learn cycle for autonomous skill acquisition but lacks ROS support and focuses on RL and manipulation tasks. Therefore, a more direct comparison for RoboGen would be Isaac Lab[13] (formerly Isaac Gym[14]), which is specifically designed for training RL and robotic tasks.

Finally, BenchBot [220] and its extension BEAR [81] are two solutions aimed at introducing a procedural way to benchmark (active) SLAM methods using Isaac Sim. However, they do not include any dynamic assets natively and, as a benchmark suite, are a closed system by nature. They also employ fixed control policies that could be unrealistic for most robots (e.g. 1 cm and 1° goal position accuracies). Then, due to their limited scope and the additional API layers between the user and the simulator itself, they lack

[7] https://github.com/allenai/ai2thor/issues/444
[8] https://github.com/allenai/ai2thor/issues/349
[9] https://github.com/allenai/ai2thor/issues/271
[10] https://github.com/ericchen321/ros_x_habitat/issues/25
[11] https://github.com/StanfordVL/OmniGibson/issues/116
[12] https://genesis-embodied-ai.github.io/
[13] https://isaac-sim.github.io/IsaacLab/main/index.html
[14] https://developer.nvidia.com/isaac-gym

the desirable customization possibilities, while being quite limited in their scope. For example, it is hard to integrate already-developed methods to control robots, or adapt the system to different platforms or tasks, as they only provide a limited set of predefined actions (i.e. `move_[next, angle, distance]`).

In contrast to previous approaches, GRADE supports multiple robots, Software-In-the-Loop (SIL), and ROS packages, and offers a customizable system where tools, settings, and simulation runs can be personalized to meet the specific needs of the researcher at the same time. By leveraging Isaac Sim, GRADE provides a highly flexible simulation system where various components can be adjusted, modified, or redefined. Its core strength lies in a modular architecture allowing researchers to customize the simulation pipeline to fit their specific needs. Unlike simulators focused on benchmarking particular approaches like Active SLAM, GRADE supports diverse experiments, including those that bypass the physics engine for efficient photorealistic data generation. It also introduces an *independent* automatic procedural asset placement system that can be replaced as needed and is not restricted to specific robots or perception systems. By exposing low-level functionalities, GRADE enables a degree of customization that is difficult to achieve in many existing frameworks. This ensures that researchers can modify and personalize their pipeline — from scene generation to control execution — making it a powerful tool for robotics and computer vision research.

### 3.2.2 Indoor Environments Datasets

There are two main ways in which we can represent indoor scenes within a simulation environment [180]: scans of real-world environments or posed meshed objects. Using scans of the real-world, e.g. HM3D [173], Matterport3D [35], Gibson Env [243], SceneNN [91], Replica [210], or Structured3D [265], poses several issues. First, those are non-interactive environments in which all the objects are non-movable. Second, any **new** object or asset placed within the environment will not be lit correctly and will not *realistically* affect the scene with shadows or reflections. Finally, many of these present various artifacts, e.g. unrealistic-looking objects, holes in the reconstruction due to reflective surfaces or unmapped areas, and uneven surfaces [180].

Using worlds based on meshed 3D assets addresses these problems while also allowing randomization (e.g. on textures, and object placement) and, eventually, interaction. However, datasets based on those, like ML-Hypersim [178] and InteriorNet [124], usually rely on non-freely available elements and only release rendered images, making them unusable for our purposes. These factors limit their adoption, reproduction, and expansion. Furthermore, the InteriorNet simulator has not been made available, while HyperSim's engine is not physics-based and its sequences relate only to very short trajectories (just 100 frames). ProcThor [51] is a recently developed framework to procedurally generate environments. However, it is limited in the quality of the assets and usable only within the Ai2THOR suite, which is focused on visual AI rather than robotics and offers no ROS support. OpenRooms [127] has not yet released any assets or CAD models pub-

licly. 3D-Front [69, 70] is a large publicly available dataset with meshed, professionally designed, and semantically annotated room layouts. This is, by far, the largest dataset available nowadays based on meshes that can be adopted. However, the annotations are not perfect and objects sometimes co-penetrate each other [106]. Finally, HSSD [106] is a synthetic Matterport-like dataset of indoor scenes. While this is a viable alternative to 3D-Front, it is still much smaller and does not provide light sources. The five environments released with BEAR [81] in five variations each are only slight modifications of worlds commercially available from Evermotion. Finally, we must mention that recently a mesh-based generation strategy for indoor environments, Infinigen Indoor [172], has been released. As it is already usable with Omniverse, it can be easily integrated into GRADE, allowing it to scale the automatic testing and data generation considerably by removing the necessity of limited mesh-based datasets.

Commercial solutions such as ArchVizPRO[15] and Evermotion[16] offer high-quality assets but are not freely available. In GRADE, we adopt 3D-Front for our simulations due to its accessibility, large variability, and mesh-based nature, which eliminates lighting inconsistencies. As discussed in Section 3.3.1, beyond seamlessly integrating them with Isaac Sim in our data generation procedure and in our custom automatic 3D-based asset placement strategy, we further enhance these environments with randomized textures and lighting conditions and partially refine the semantic mapping during conversion.

### 3.2.3 Simulated Animated Humans

Most dynamic content in indoor scenes comes from human movement. In V-SLAM and autonomous robotics, handling dynamic elements is crucial, as they disrupt key processes [17, 131] (e.g. loop closures, visual odometry) or necessitate the implementation of additional techinques [231] (e.g. dynamic obstacle avoidance). These challenges are often addressed using DL methods for detection, segmentation, and motion prediction [17, 231, 234], which require large-scale ground-truth datasets.

Collecting real-world GT human motion data is limited to controlled setups like Vicon Halls or motion capture (MoCap) systems [140], which use multi-camera marker-based tracking for high-precision joint estimation. The humans can then be represented virtually as parametric 3D human models that can be used for different tasks like 3D human reconstruction [188, 245] and pose estimation [123, 187], even from single images. Those models can also provide fine control over pose, shape, and motion. Commonly used human body models include SCAPE [4], GHUM [246], and the SMPL series [138, 159, 181]. SMPL is arguably the most widely adopted thanks to its flexibility, capability to model hands and facial expressions, and compatibility with various simulation engines.

Unfortunately, MoCap and Vicon systems support only a narrow range of subjects,

[15] https://oneirosvr.com/portfolio/archvizpro/
[16] https://evermotion.org/

clothing, and scenarios. To overcome these constraints, synthetic data has become increasingly popular, leveraging parametric 3D human models rendered in engines like Blender and Unreal Engine [20,187]. However, most of them have significant limitations. Many are generated by compositing human figures onto static backgrounds [20, 59] or capturing single-frame images rather than full video sequences [187, 253], often leading to artifacts such as floating humans and incoherent placements [59, 168]. Clothing representation is another challenge, as most SMPL-based datasets lack 3D clothing [229] or rely on explicit point-cloud-based models [85, 212, 233], which are difficult to edit and integrate into simulations. This is partly due to their use in rendering engines without physics simulation and the predominant research focus on human shape over clothing dynamics. Commercial solutions like RenderPeople[17] or CLO[18] offer realistic clothed human models, but freely available datasets with animated *clothed* SMPL humans remain rare. Notable exceptions include Cloth3D [15] and BEDLAM [20].

Still, as discussed in Section 3.1, existing datasets are often inadequate for robotics. They lack key information needed for autonomous systems, such as camera states, IMU readings, scene depth, and point-cloud data. Additionally, they are non-interactive, limiting their applicability in developing autonomous systems that must respond to human movement in real time. Advancing robotics research in dynamic environments requires integrating animated human assets into realistic, robot-focused simulations, particularly for tasks like obstacle avoidance and Active V-SLAM.

In GRADE, we use Cloth3D and AMASS as primary sources of animated human models for indoor dynamic environments. Cloth3D provides diverse, physically plausible clothing deformations, while AMASS offers a large corpus of high-quality human motion sequences. To integrate them into Isaac Sim, we developed a custom SMPL converter, detailed in Section 3.3.1. Note that, although Isaac Sim also supports realistic physics-based clothing simulation, we did not use this capability in GRADE.

## 3.3 GRADE Components

Following the logical structure depicted in Figure 3.2, we outline here the key components of GRADE: asset preparation and placement (Section 3.3.1), robot creation and control (Section 3.3.2), simulation management (Section 3.3.3), and post-processing tools (Section 3.3.4).

### 3.3.1 (Non-)Rigid Assets Preparation and Placement

Isaac Sim relies on the Universal Scene Description (USD) format, thus requiring the conversion of assets into USD files before their integration into the simulations. This process poses several challenges, as many objects or complex animated models cannot

[17] https://renderpeople.com/
[18] https://www.clo3d.com/

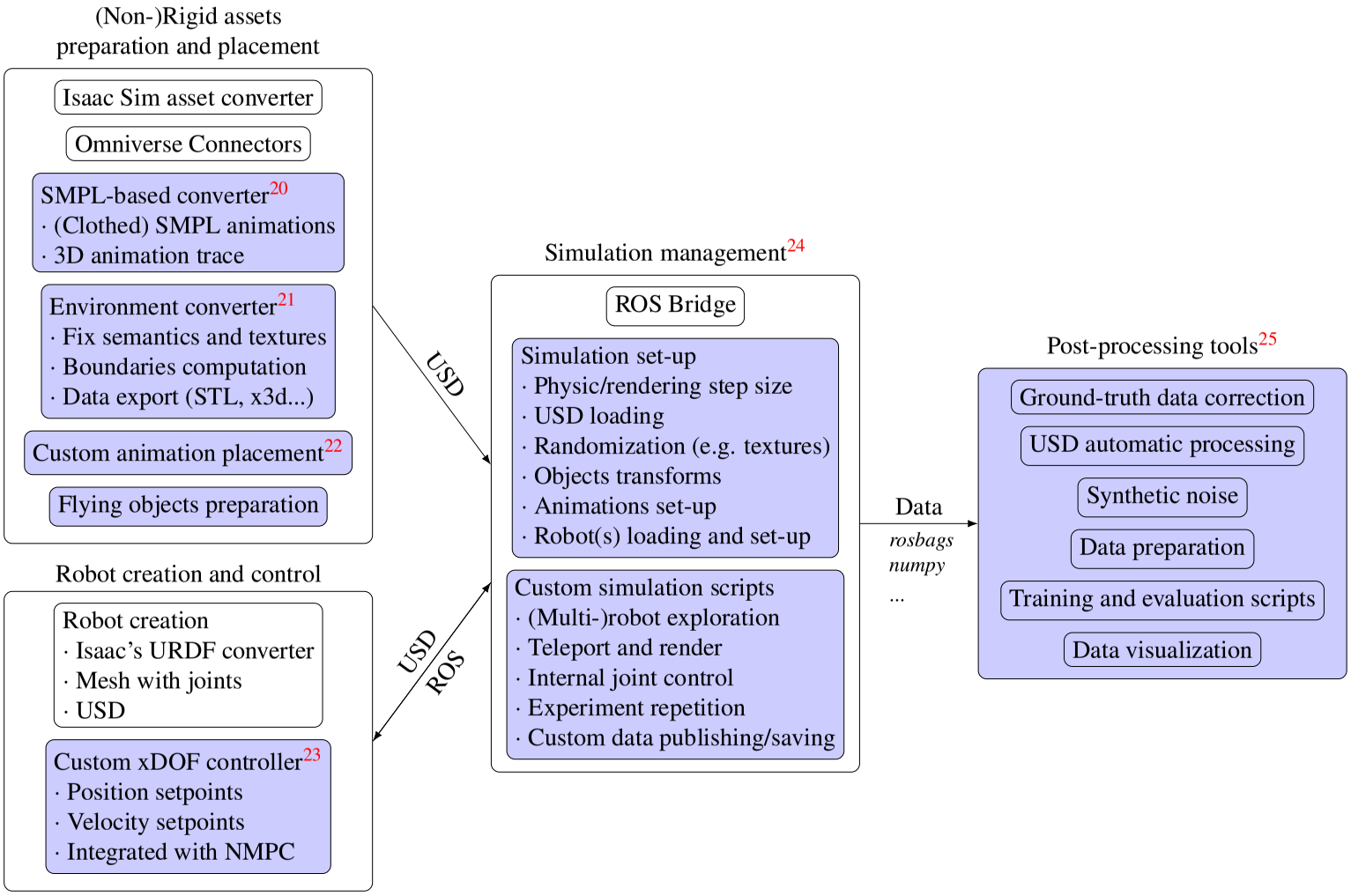



Figure 3.2: Recap of the main components of the GRADE framework. With a blue background, we highlight our custom software and reference the specific repository in the footnotes.

be easily or directly converted. Nonetheless, adopting the USD format significantly enhances the overall flexibility of the system, enabling the inclusion of assets from various sources, including UE, Blender, AutoCAD, Maya, and more, through Omniverse Connectors[19] and custom software — a common limitation for other simulation tools, such as Gazebo or Habitat.

### Environments

First, we prepare the assets that will be the robot's working environments within our simulation. Although Isaac Sim provides a converter for common OBJ and FBX files to the USD format, this process often fails with complex models and hierarchies. Moreover, additional processing, like incorporating semantic classes, exporting supplementary data, or adjusting the scale of the environments, is generally desirable. Therefore, we customize *BlenderProc* [54] to enable a reliable conversion and preparation of our environments to the USD format. Specifically, we modify its 3D-Front processor by fixing the texture generation, scaling of the assets, merging geometries, and refining the semantic mapping procedure adopted specifically for the 3D-Front dataset [69] (our main source of environments) to partially correct its wrong mappings[26]. We then export the environment in USD, STL, and X3D formats. The USD file is loaded into Isaac Sim during our simulations, while the X3D file can be easily converted into an octomap [88] for subsequent evaluations. Additionally, we also compute the enclosing rectangle and an approximated non-convex polygon that, along with the STL file, are used during the assets placement procedure.

### Objects

In addition to the environment itself, we aim to have the ability to include additional objects within the simulation to increase its diversity and complexity. To achieve this, we adapt the standard converter to be able to dynamically download and load various objects at runtime from datasets such as Google Scanned Objects [57] (GSO) and ShapeNet [36]. These objects can then be placed randomly or at predefined locations, or animated as flying entities through random, non-physics-enabled transformations.

[19] https://docs.omniverse.nvidia.com/connect/latest/connecting-to-omniverse.html
[20] https://github.com/eliabntt/animated_human_SMPL_to_USD
[21] https://github.com/eliabntt/Front3D_to_USD/
[22] https://github.com/eliabntt/moveit_based_collision_checker_and_placement/
[23] https://github.com/eliabntt/custom_6dof_joint_controller
[24] https://github.com/eliabntt/grade-rr and https://github.com/eliabntt/ros_isaac_drone
[25] https://github.com/robot-perception-group/GRADE_tools
[26] https://github.com/DLR-RM/BlenderProc/issues/430

### Pre-Animated Assets

The Omniverse connector that converts assets from Blender to the USD format works well for simple animated assets. However, similar to the challenges faced with environment conversion, it fails when handling complex objects like SMPL-based animations. As mentioned in Section 3.2.3, SMPL fittings are one of the most widespread models used to represent and control human pose and shape in simulations. Therefore, to incorporate animated humans into our experiments with GRADE, we introduce a new software tool based on Blender to automatically convert (clothed) SMPL animated sequences to USD format, e.g. from the Cloth3D dataset [15]. This tool allows us to correctly process and load various pre-animated human assets into GRADE, performing different pre-recorded motions as non-rigid entities, either through deformable meshes or subsequent skeletal transformations. Additionally, we generate the STL of the 3D trace of the animation. The STL files store the evolution of the 3D surface geometry occupied by the animated humans and their clothing throughout their *entire* animated sequences. By combining these with the STL representation of the environment (exported in the previous step), we construct a unified geometric representation that enables precise collision detection. This is thus fundamental to our automatic placement procedure, as it allows us to evaluate candidate positions and orientations of the animated assets efficiently.

### Asset Placement

Given an environment and a set of (animated) assets, we should avoid physical overlap among them. To this end, different strategies can be employed, such as hardcoding or manually setting their locations in advance (as we do in Section 3.4.1). Still, to achieve randomized data generation and seamless testing across diversified scenarios, the placement procedure must be automated. However, simple 2D occupancy projections are not a good approach in our case. As we use clothing animations and diversified actions (e.g. jumping, dancing) the projected footprints, usually approximated with rectangles, can be significantly larger than the human model itself. Thus, their overlap with other footprints would not necessarily indicate a *real* collision. For instance, an animation where the human arm passes over a table would indicate a collision when using a 2D occupancy map of the world, even though they are not actually colliding with each other.

Therefore, with GRADE we introduce a custom placement strategy specifically tailored for (clothed) animated humans, which we use in our data generation procedure (see Section 3.5). The pseudocode of our approach is provided in Algorithm 3.1. First, we load the STL files containing the 3D trace information of human and clothing animations along with the environment. The placement procedure is attempted up to 10 times per asset; if a valid position is not found, the asset is discarded (Algorithm 3.1, line 8). Each trial begins by selecting a candidate position and orientation for the asset origin. To ensure broad coverage and variation across experiments, the position is chosen randomly, uniformly distributed over the floorplan or enclosing rectangle (Algorithm 3.1,

lines 9–13). The yaw orientation is instead uniformly chosen between 0 and 360 degrees. We then check for intersections between the candidate asset, the environment, and any previously placed assets using the information we saved in the STL files. To balance realism and feasibility, we use an empirically determined collision threshold of 200 intersection points between the asset we are trying to place and any other mesh in the environment (Algorithm 3.1, line 15). Assets with fewer intersections remain in the scene to prevent unnecessary rejection of minor overlaps, such as slight penetrations with plant leaves or clothing. If the number of intersections exceeds this threshold, the placement attempt fails, and the loop repeats. Otherwise, after a successful placement, the environment updates dynamically to account for newly added assets. We implement this through a custom MoveIt interface, leveraging its integration with the Flexible Collision Library (FCL) for checking collisions between meshes. Notably, GRADE's modular and flexible design enables seamless integration of alternative placement strategies when desired.

### 3.3.2 Robot Creation and Control

#### Creation

Theoretically, custom robots can be loaded into Isaac Sim through the integrated URDF format converter. However, this does not work correctly for our robots - a three-wheeled omnidirectional robot and a flying drone - due to incorrect scaling factors, missing parts and joints, and improperly placed or absent sensors. To address these limitations, we construct our robot platforms directly within Isaac Sim by adding revolute and translational joints to mesh objects and saving them as single USD files. Joint configurations (e.g. limits, maximum speed) and sensor specifications (e.g. type, settings) can be predefined in the USD model, similarly to URDF files, or loaded and modified dynamically during the simulation. In GRADE, we implement the latter approach, enabling greater flexibility compared to URDF definitions or USD pre-configurations. We load and configure our robots **entirely** at runtime through simple Python code, including attributes such as number, type, and position of sensors, joint stiffness and physical responses, robot's weight, and desired ROS topic names. These configurations can be set at the start of a simulation or adjusted dynamically during the run itself. This workflow simplifies the setup of custom robots and experiments, increasing control and flexibility of the simulations compared to previous approaches. For instance, it enables the seamless simulation of multiple heterogeneous robotic platforms, as shown in Section 3.4.2.

#### Control

Isaac Sim provides only a few default approaches to control assets and robot movements. However, using teleportation or rigid and non-physics-based transformations is inadequate for simulating realistic robot motions and collecting useful sensor data. Nonetheless, these methods can be viable when physics information is unnecessary, such

**Algorithm 3.1** Pseudocode of our custom placement procedure.

```
Data:
WORLD: folder path of the static environment
ASSETS: list of paths of the assets to be placed
STRICT: boolean, if the random location should be inside the enclosing rectangle or the
tight boundaries
Result: List of triplets: assets, locations, and orientations
// Preparation
environment = load WORLD STL;
    boundaries = load WORLD boundaries;
    rectangle = load WORLD enclosing rectangle;
    placed = [];
    isGood = False;
    // Main Loop
foreach elements of ASSETS do
    asset = load element STL;
     for i = 1 to 10 do
        if STRICT then
            position = uniform within boundaries;
        else
            position = uniform within enclosing rectangle;
        end
        yaw = uniform between (0,360);
         isGood = CheckForCollision(environment, asset, position, yaw, limit=200);
         if isGood then
            break;
        end
    end
    if isGood then
        // Update the return vector
        placed.append((element, position, yaw));
            // Update the environment so that this asset is taken into
            account in the future
        environment.update((asset, position, yaw));
    end
end
return placed;
```

as when collecting only visual data (see Section 3.4.1). Additionally, Isaac Sim allows direct joint control through low-level APIs via position or velocity setpoints, as applied in our experiment repetition procedure (see Section 3.4.3). However, this requires pre-configured waypoints and does not support pre-developed Software-In-the-Loop (SIL) control frameworks commonly integrated through ROS and Gazebo. Alternatively, Isaac Sim includes built-in motion and control models for a few specific platforms - an ineffective approach when working with custom robots that are not natively supported. For example, BenchBot [220] relies on this feature to simulate ground robots, offering only a narrow set of predefined commands, further constraining its flexibility and generality. Moreover, Isaac Sim lacks support for fluid-dynamic physics, which is necessary for simulating UAVs and, similarly to Gazebo, it does not model frictionless perpendicular translation movements required for omnidirectional wheels. The recent PegasusSimulator [96] addresses PX4 UAV control by directly applying force to the drone mesh, thus still without simulating actual fluid dynamics. Our approach differs as we seek to allow for custom robot simulation and control by employing a PID-based joint-level controller to manage robot movements. We leverage the ROS communication system and joint definitions, receiving position or velocity setpoints from other software, such as (N)MPC or Active SLAM frameworks, and convert them into low-level commands. The simulation software then processes those and translates them into robot movements. The ROS communication system is crucial for seamless integration with the Isaac Sim framework, allowing us to assign velocity and position setpoints to each joint independently while remaining agnostic to the underlying robot architecture. As a result, GRADE can support multiple platforms, from UAVs to robotic arms, and is not limited to individual robots or simple camera setups, unlike other simulators or frameworks.

### 3.3.3 Simulation Management

The main simulation cycle, which uses Isaac Sim APIs along with custom utilities, primarily manages: i) starting and configuring the simulation environment, ii) loading, placing, and configuring assets and robots, iii) executing several randomization procedures, iv) launching complementary ROS nodes when necessary, and v) managing simulation steps (both physics and rendering) and data saving. Through this framework, we can control various options, such as the number of dynamic assets, the initial location of the robot, and the size of the physics and rendering steps. It also allows programmatic and dynamic modification of environmental conditions, physics and rendering settings, light colors and intensity, material reflection parameters, asset textures, and the time of day, among others, thereby increasing the variability of simulations. This variability, along with the ability to independently enable or disable physics and ROS, allows us to support a wide range of simulation scenarios, thus enabling broader applicability.

In GRADE, we have explicitly implemented several illustrative simulations demonstrating different desirable applications, including Active-SLAM-based exploration, ROS-free data collection in a savanna environment with animated zebras, or experiment

repetition, as described in Section 3.4. While Isaac Sim provides randomization and ground-truth data-saving methodologies, we found these functionalities limited. With GRADE, we expose and integrate the underlying methods directly into our simulation management approach for a customized experience. In particular, the default data-saving tool is restricted to camera-related information and executed each time a rendering call is made. However, as multiple rendering calls are required to generate an accurate image of complex environments due to path-tracing computations, this results in neither accurate nor comprehensive data. To address these limitations, we modify the saving process to gain finer control, fix various issues (e.g. segmentation ID overflow), and collect additional information, such as the camera's vertical field of view and IMU measurements at each timestep. Moreover, our finer control allows us to customize sensor and ROS message rates independently of rendering and to access and modify the information before publishing, for instance, to add noise or implement custom drop rates.

### 3.3.4 Post-Processing Tools

Real-world sensor data is inherently noisy, exhibiting issues like measurement drift in IMUs, motion blur in images, and depth inaccuracies from sensors that follow exponential noise models. Therefore, ground-truth data generated by the engine must be processed to closely replicate real-world conditions, which is essential for training robust DL systems and evaluating methods such as Dynamic V-SLAM. Noise could also be introduced at runtime by modifying the data from the simulator before it is saved or published by ROS nodes. However, post-processing the saved ground-truth data offers greater flexibility by allowing, for example, multiple experiments with different noise levels. Therefore, within GRADE, we develop a tool to introduce noise into the saved ground-truth data, building upon and extending methods from RotorS [71] and Zhang et al. [260]. Specifically, our approach incorporates: i) IMU noise and bias, ii) RGB motion blur and rolling shutter noise, and iii) depth filtering and noise. The noise augmentation tool is structured to allow seamless extension to additional sensor modalities. The framework processes raw data sequences or *rosbags* by applying user-defined perturbations to different data streams. Its structured pipeline separates data modalities (e.g. RGB, IMU), allowing different integrations with minimal modifications. We use this tool in experiments to prepare data for Dynamic V-SLAM and network training, adding noise to depth and RGB for one dataset while also augmenting segmentation masks for motion blur in another (see Section 3.6.2). Beyond the noise augmentation, we also automate Dynamic V-SLAM evaluation and address errors in the generated ground-truth data caused by known issues in Isaac Sim, such as inaccurate 3D bounding boxes[27], incorrect poses of some animated assets[28,29], and timing discrepancies in *rosbags*[30].

[27] https://forums.developer.nvidia.com/t/234675
[28] https://forums.developer.nvidia.com/t/251877
[29] https://forums.developer.nvidia.com/t/193900/10
[30] https://github.com/qcr/benchbot/issues/66

(a) People in a city (b) Multi-robot: ground robot (c) Multi-robot: UAV #1 (d) Multi-robot: UAV #2 (e) Zebras in a savanna

Figure 3.3: Few examples of environments that can be simulated with GRADE. The RGB images are shown in the top row, with the associated instance segmentations (randomly colored) below. For the multi-robot UAV images (Figures 3.3b to 3.3d), we highlight the other robots in the field of view with a red box. An external view of the UAV observing the city and the apartment environments can be observed in Figure 3.5. The images are best viewed in color.

## 3.4 GRADE Case Studies

We use GRADE to address three sample case studies that cover the different ways we can apply the framework. These are i) a ROS-free simulation and curated data collection in a savanna environment (Section 3.4.1), ii) online testing of Active SLAM approaches in a (multi-)robot scenario (Section 3.4.2), and iii) our newly introduced experiment repetition setup (Section 3.4.3). Throughout our work in GRADE, we use i) a UAV Firefly model from RotorS [71], and ii) the three-wheeled omnidirectional robot from iRotate. As mentioned in Section 3.3.2, each robot is equipped with a single joint for every degree of freedom (six for the UAV and three for the omnidirectional ground robot), and their sensors (camera, IMU) are loaded dynamically at runtime.

### 3.4.1 ROS-Free Simulation in a Savanna Environment

In this example, we deploy a UAV in a savanna environment[31] with several simulated animated zebras. The focus is on creating a custom simulation without using either ROS or any additional SIL method to control the robot. A schematic of this setup is provided in Figure 3.4. While the primary focus of our work in this chapter is on indoor dynamic environments, we include this experiment to demonstrate the adaptability of GRADE beyond human-centric indoor scenes. By evaluating GRADE in a completely different setting — an outdoor savanna with dynamic, non-human agents — we validate its ability to incorporate diverse environment sources, such as those available in Unreal Engine, and diverse animation sources, beyond SMPL-based humans, and extend its applicability to other domains where dynamic elements play a key role. Notably, we will leverage this case study in Chapter 5 to generate our synthetic data. The zebra and their animations are sourced from a free SketchFab[32] asset. Using Blender, we export four animation sequences, namely walking, eating, trotting, and jumping, create three different transition sets between these animations, and manually place the zebras within the main environment (Figure 3.3e, left side of Figure 3.1). Waypoints are provided to the Isaac engine directly from the main simulation loop as predefined position/orientation goals. We use either i) a scripted sequence of waypoints for each of the six joints using the physics engine, or ii) a physics-less mode where the drone acts as a floating object that 'slides' smoothly between waypoints. The drone dynamics are governed by its mass and joint characteristics (e.g. damping coefficient) in the physics-enabled mode. Otherwise, the UAV follows an interpolated trajectory between goal locations.

[31] https://fab.com/s/a6b34008be90
[32] https://skfb.ly/opCUB

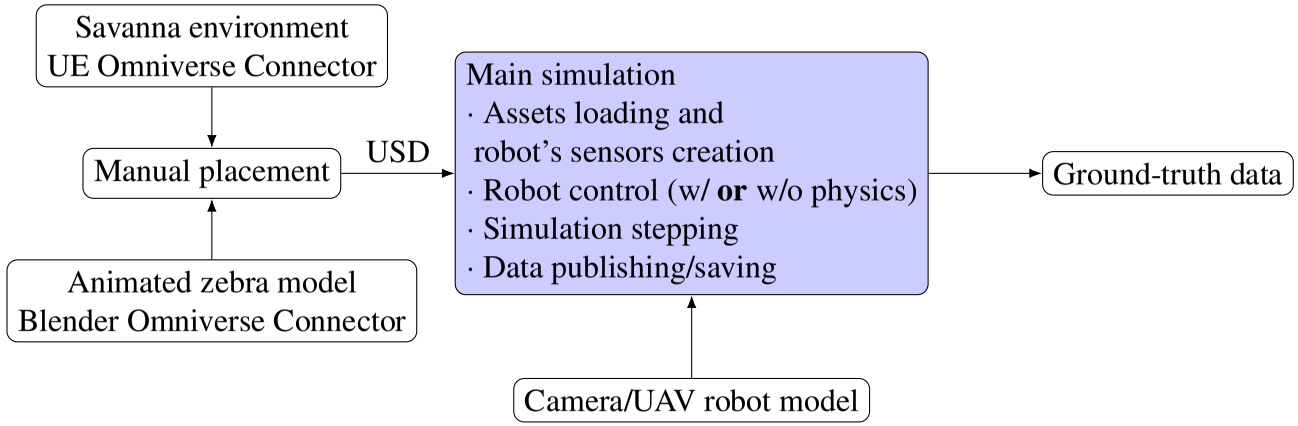



Figure 3.4: Flow diagram of the ROS-free system used in the savanna simulation presented in Section 3.4.1. In blue, we highlight our customizations. Here, we take a savanna environment from UE and an animated zebra model from SketchFab. We combine them manually in a single USD environment. This USD is then loaded with the UAV robot model in Isaac Sim. The robot is controlled by the main simulation script directly, whether with or without the physics engine, which also manages the rendering and the data-saving steps.

(a) UAV in a city (b) UAV and UGV in an apartment

Figure 3.5: We show here an external view of the UAV (similar to Figure 3.1) in a city environment and an apartment, which are the same used in Figure 3.3a and Figures 3.3b to 3.3d, respectively. In (b), it is also possible to observe the UGV.

### 3.4.2 (Multi-)Robots and Active SLAM

This scenario uses GRADE in conjunction with previously developed ROS approaches. Specifically, we aim to use Active SLAM methods to explore indoor environments (from 3D-Front) with both UAVs and UGVs. The scheme of this workflow is depicted in Figure 3.6. We adapt and interface the FUEL [268] Active SLAM framework with Isaac Sim to compute exploration goals for the UAV. FUEL uses online RGB-D and odometry data from the simulation to actively compute exploration goals. However, the original FUEL implementation leverages an integrated custom and simplified simulation to control the drone movements. Therefore, to bridge this gap and interface ourselves with Isaac Sim and GRADE, we supply the exploration goals to an additional NMPC [104] to predict a realistic trajectory for the UAV. The final predicted state of this trajectory is then sent to our custom controller, which then provides commands to the simulation itself. The omnidirectional robot is instead managed by the iRotate framework in conjunction with our custom controller without any additional layer, as it natively provides NMPC velocity setpoints for the base and the independently rotating camera. In this scenario, we also create simulations that simultaneously manage multiple and heterogeneous robots. The challenge here is that we have to load specific sensor suites and set different ROS topics for each one of the robots created dynamically. We do so by allowing dynamic reconfiguration and loading of parameters within the same simulation script. An example of the multi-robot simulation is shown in Figures 3.3b to 3.3d. In those images, two robots are UAVs, each running a different instance of FUEL, and one is the three-wheeled ground omnidirectional robot used in iRotate (Section 2.4).

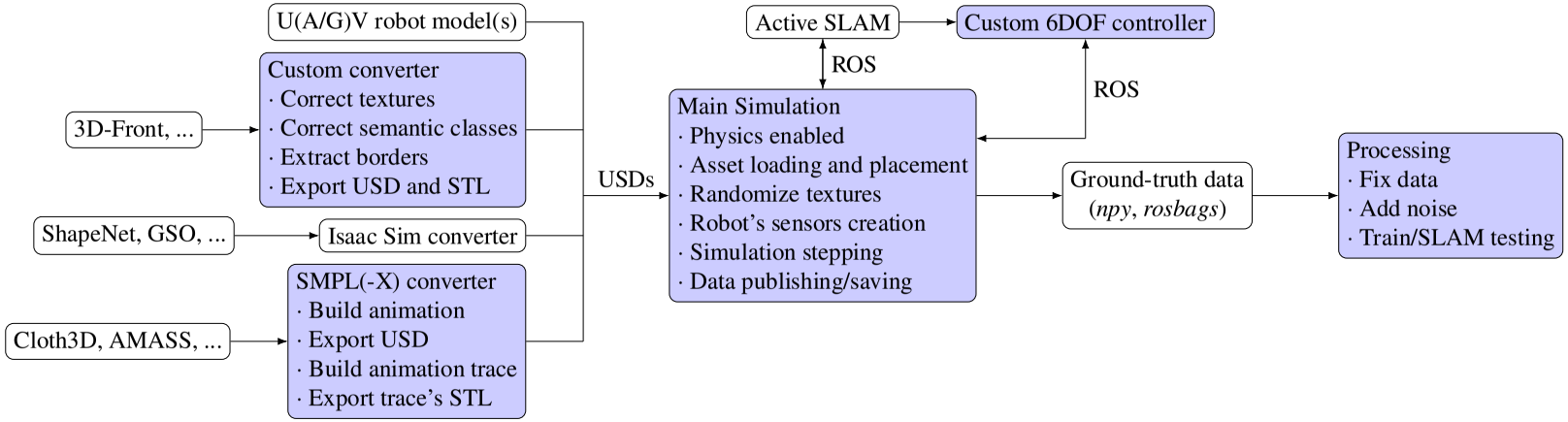



Figure 3.6: A flow diagram of the main dataset generation pipeline presented in Section 3.4.2. In blue, we highlight our customizations. First, we take environments from 3D-Front, flying objects (when desired) from ShapeNet and GSO, and animated humans from Cloth3D. We process both the environment and the animated humans using our custom converters, preparing the data for the simulation and the asset placement procedure. A single procedure then takes care of running the simulation, from the asset loading and placement to the data publishing and saving. The main simulation is in the loop with an Active SLAM method and a customized 6DOF controller that communicates with each other using ROS. The ground-truth data is then processed by our tools that apply fixes when needed, as described in the chapter, noise, and use it to train detection and segmentation approaches and evaluate Dynamic V-SLAM methods.

### 3.4.3 Experiment Repetition and Enhancement

GRADE provides a way to replay any previously recorded experiment precisely. This can be done either by keeping them as-is or by modifying them by selectively altering any number of conditions. These include, for example, attaching new sensors (e.g. cameras, LiDARs) to the original robot itself, changes in light conditions, or the inclusion of new robots, humans, animals, or other objects in the surrounding environment. This is while keeping the physics simulation enabled. Therefore, with GRADE, we introduce a new method that allows studying the robustness of different approaches by changing the robot's surrounding conditions and/or expanding previously collected datasets under the *exact* same settings, e.g. by collecting new data. To this end, we provide two possible solutions under the requirements that both i) all of the robot poses throughout the experiment and ii) the initial simulation conditions were logged. We either teleport the robot to the exact logged location and re-render the scene and the new sensors as-is, or use the previously logged joint velocities and target positions at every time step as targets of the Isaac Sim internal joint controller. While more flexible, this second approach could generate some minor deviation due to the unknown acceleration between two timesteps. However, combining the two strategies can easily mitigate this effect. Moreover, we can further interpolate the missing poses when necessary, e.g. if a newly added sensor has a different rate than the one with which the pose was saved. Notably, repeating experiments with this approach removes any variability that the developed method might have — for example, when testing Active SLAM approaches. At the same time, it ensures that the remainder of the simulation is conducted under controlled and physically realistic conditions. Note that, differently from replaying *rosbags*, using fixed seed numbers when the simulation is prepared, or deterministic Gazebo runs, our approach *allows* changing the underlying state of the simulation. This would happen for example following modifications like adding new sensors to the robot (with mass), different scene content that is impacted by physics, additional robots in the scene, or fully disabling the physics to re-render a scene without dynamic elements, as we do in our experiments in Section 4.3. All of this can be controlled programmatically through the means of a simple Python script with which we can selectively choose both what to alter and then how to control the simulation itself.

Examples of the re-rendered images and depth maps during an experiment repetition run are provided in Figure 3.8, while the workflow can be found in Figure 3.7. In Figure 3.8 we also show an example of changes in the original simulation run by changing the scene contents and lighting conditions and observing the scene from a new perspective. We will evaluate the deviation with respect to the logged poses and the rendering differences in Section 3.6.1.

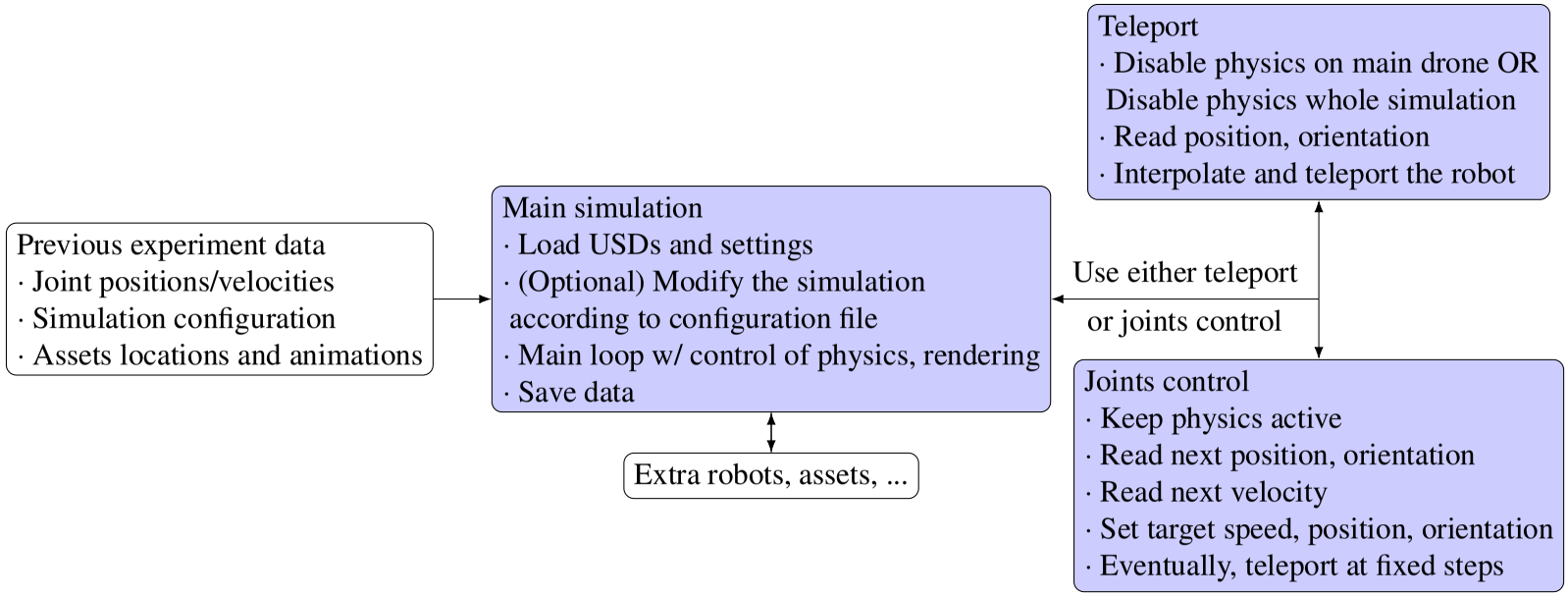



Figure 3.7: Flow diagram of the experiment repetition pipeline presented in Section 3.4.3. In blue, we highlight our customizations. Given the previously recorded experiment data, consisting of the simulation configuration, logged joint positions and velocities, and assets locations and animation sequences, we can exactly repeat the experiment while allowing a wide range of modifications. The main simulation tool loads the assets, optionally modifies the simulation content (e.g. removes objects, changes lighting conditions, adds sensors to the robot), repeats the experiment, and saves the data. Additional robots, assets, or SIL can be configured to interact with the repeated experiment by integrating them into the script that manages the simulation. The main robot can be controlled either by using teleporting or the joint control system. Both receive the logged information from the main simulation tool.

## 3.5 Data Generation

We use GRADE to generate the data for our experiments in indoor dynamic environments through the aforementioned Active V-SLAM modality. Following what we described in the previous sections we use i) the custom environment converter, to prepare the world and extract its STL and boundaries; ii) the SMPL animation converter, to prepare the animated assets and extract the STL representing their trajectories; iii) the custom placement procedure; iv) GSO and ShapeNet as flying objects; and v) our custom controller to manage the robot(s). An example of the richly annotated data generated can be seen in Figure 3.9, and the summary of the data we release with GRADE is presented in Section 3.5.1.

Our main source of environments is the 3D-Front [69] dataset, i.e. one of the largest collections of mesh-based indoor environments. We enhance them with random textures from *ambientCG*[33] and varying lighting conditions. We also collect data in one outdoor city environment from the Unreal Engine marketplace[34] (Figures 3.3a and 3.5a), and once in an indoor[35] world from SketchFab (Figures 3.3b to 3.3d and 3.5b), both used also to further validate GRADE and Isaac Sim's flexibility across different sources. The dynamic components in the scenes are *animated* humans and, in some experiments, random flying objects. The humans are taken from the Cloth3D [15] and AMASS [140] datasets. While Cloth3D provides clothed assets, AMASS CMU sequences consist of only unclothed SMPL fittings. While doing so, we randomize the appearance of the assets by using Surreal's SMPL textures [229], i.e. freely available low-resolution textures, as shown, for example, in Figure 3.1. The flying objects belong to various categories (e.g. toys, balls, tables, etc.) and serve multiple purposes: they generate occlusions between the camera and the other elements in the environment, increase the variability of the scene, and introduce dynamic elements that can challenge Dynamic V-SLAM methods. For example, they negatively impact feature rejection methods such as segmentation and detection models or optical flow approaches since they create occlusions, do not belong to common dynamic classes, and have unpredictable motions. Additionally, their presence reduces the likelihood of loop closures, as they can randomly cover the scene, further testing the robustness of the evaluated V-SLAM systems. Moreover, introducing them will allow us to automatically create images with partially covered humans, thus increasing the variability of their appearance on the images we will use to train detection and segmentation models. These objects, belonging to the GSO and ShapeNet datasets, are loaded dynamically at runtime. We rigidly "animate" them through random time-keyed transformations in scale, orientation, and position using multiple goals set within the environment limits. By design, this is done without considering any possible collision with the environment or other assets. While precise collision-avoidance strategies

[33] https://ambientcg.com/
[34] https://fab.com/s/c3c3d0cf2c4f
[35] https://skfb.ly/YZoC

can be implemented, like pre-computing safe trajectories, this allows a higher variability of their motion patterns. Overall, this diverse data generation approach, enabled by GRADE, allows us to systematically analyze the impact of dynamic elements on perception and localization tasks.

| | $x, y, z$ | roll, pitch | yaw |
|---|---|---|---|
| Position | Env. limits | $[-25, 25]$ or $[0, 0]$ deg | $[0, 360]$ deg |
| Speed | 0.5 m/s | 40 deg/s | 30 deg/s |

Table 3.1: We report the joint limits used for the UAV in our data generation procedure, as described in Section 3.5.

The main process is as follows. First, we load the environment, center it with respect to the origin, and generate the 2D occupancy map. In this, we also define the size of the physics step, i.e. how much the clock will advance during every loop of the simulation, and rendering parameters, e.g. auto-exposure or path/ray-tracing settings. In our experiments, the physics step is set to 1/240th of a second. Second, we randomize lights' colors, intensities, and surface roughness (i.e. reflection capabilities). Third, we load the robot, move it to the starting location, attach its specific set of sensors (including the cameras) and ROS publishers, and link it to the motion controller through the correct ROS topics. Based on the chosen configuration, the robot is controlled with either full 6 DOF capabilities or with a stabilized flight (without roll and pitch). The UAV's joints are subject to position and speed limitations as shown in Table 3.1. Recall that the FUEL Active SLAM framework will control the UAV online during the simulation, allowing us to explore the environment and collect data autonomously. Then, we import a random number of people and randomly place them within the environment. The number of flying objects loaded varies based on the experiment settings. The initial location of the robot is randomized using the same placement procedure as the animated assets to prevent collisions. Additionally, because our automatic data generation relies on a Visual SLAM approach, we pre-optimize the robot's initial orientation to avoid featureless areas, such as windows. In our environments, we do not include artificial panoramic backgrounds or detailed outdoor scenes typically found around homes, meaning these regions lack the necessary visual cues for robust V-SLAM initialization. By ensuring the robot does not start facing such areas, we improve both the stability of SLAM and the reliability of the automatic data generation process.

Overall, considering that all simulations have animated humans and accounting for the degrees of freedom of the robot and the presence of flying objects, we obtained six scenarios (excluding the outdoor and multi-robot setting) summarized in Table 3.2.

After loading and setting up the simulation experiment, we bootstrap the first 1 second for every experiment to randomize the initial conditions of the robot. When the bootstrap sequence ends, we publish a single message to signal the start of the experiment and record data for 60 seconds. In the main simulation loop, we i) advance the physics

| Humans | GSO | ShapeNet | Horizontal | Sequences |
|---|---|---|---|---|
| 7-40 | 0 | 0 | | 63 |
| | | | ✓ | 77 |
| | 5 | 5 | | 44 |
| | | | ✓ | 63 |
| | 10 | 10 | | 33 |
| | | | ✓ | 62 |

Table 3.2: Summary of our generated data, including the number of sequences released for each configuration. The number of humans is randomly selected between 7 and 40 before placement. However, the final number of humans in the scene may be lower due to space constraints that prevented their successful placement, as explained in Section 3.3.1. The number of flying objects taken from the GSO and ShapeNet datasets is fixed. A tick in the *Horizontal* column indicates whether the UAV is free to move or constrained to a horizontal orientation (i.e. can only rotate in yaw).

one step at a time, ii) automatically control the animation timeline and the rendering steps, iii) publish the ROS information at the desired rates (using the physics step as a reference), and iv) write data to the disk. The control of the animation timeline and the rendering steps is necessary because, as mentioned previously, multiple calls to the path-tracing function are necessary to render complex scenes correctly. Each one of these calls, however, will "advance" the animations in the scene (by advancing the time on the timeline). Therefore, in GRADE, we implement a procedure to ensure a correct alignment of the physics, rendering, and animations towards a precise data generation procedure.

The scene is rendered with path tracing and auto-exposure. The rendering speed greatly depends on the number of lights, reflections, assets in the scene, and cameras. We use two cameras with the same horizontal and vertical FOV, one low resolution ($640 \times 480$) and one high resolution ($1920 \times 1080$). Across different architectures, rendering each **couple** of views took an average of 12 seconds, including the time necessary to get the remaining ground truth information, such as instance segmentation. However, by tuning the simulation parameters, one can improve that to multiple images per second, up to processing times faster than 15 FPS. In this, the physics simulation step causes a noticeable delay whenever high-frequency messages related to physics need to be published (e.g. the IMU) due to its tiding with the USD files and the necessity to constantly read and write information to them[36]. The low-resolution RGB and depth are published with ROS and saved with the *rosbag* tool. The high-resolution camera is used to save ground truth data with *numpy* arrays such as 2D and 3D bounding boxes, instance segmentation masks, camera pose, as well as RGB and depth. A full description

[36] https://forums.developer.nvidia.com/t/226738/6

of the data we save with the corresponding rates is presented in Table 3.3. The animations are reversed based on the average movement duration to increase variability and avoid too many static figures. Note that, due to flying objects and the unpredictability of moving subjects, sometimes the drone goes 'through' dynamic assets. Although a way to avoid this would be to pre-sample valid trajectories, we see this as an opportunity to develop more reliable systems. Indeed, when these events occur, they cause situations that are usually untested with currently available real-world datasets, like losing track of the features due to completely black (e.g. corrupted) images or sudden changes between frames (e.g. due to a new flying object). However, those are possible scenarios that *can* occur if one considers real hardware or communication links that might fail or degrade or real-world environments in which animals or anything else could cause occlusions of the camera lens even for a brief moment. Nonetheless, we are not aware of any data currently available that poses such a challenge. Importantly, our experiment repetition tool (Section 3.4.3) provides a means to systematically address occlusions without compromising experimental consistency. The tool enables the regeneration of frames affected by occlusions while maintaining the original experiment setup, including camera poses, scene dynamics, and lighting conditions. This is without introducing errors, as shown in Section 3.6.1, preserving both realism and repeatability. Additionally, this capability will allow us to evaluate how different algorithms respond to different scenarios. We will bring this concept to an extreme during our evaluations of Dynamic V-SLAM methods by re-rendering the tested dynamic sequences without *any* moving object (Section 4.3). We provide an example of a re-rendered scene without the dynamic elements in Figure 3.8.

| **Sensor** | Clock | IMU | TF | Joint state | Camera pose | Odometry | RGB, depth semantics... | Starting experiment |
|---|---|---|---|---|---|---|---|---|
| **Hz** | 240 | 240 | 120 | 120 | 60 | 60 | 30 | once |

Table 3.3: For each sensor (first row), we report the frequency in Hz used during our data generation procedure (Section 3.5).

### 3.5.1 Summary of Released Data and Code

We release 342 sequences of 1800 frames each that, at 30 fps, correspond to 342 minutes of video, i.e. 615K frames. Those are summarized in Table 3.2. For each of these 342 experiments, we release depth data, instance segmentation (including clothing segmentation), 2D tight and loose bounding boxes[37], 3D bounding boxes, and the corresponding camera information and poses. Additionally, we release the processed animated human data with 3D per-vertex locations and skeletal information. All of the aforementioned data is saved via *numpy* arrays. For each sequence, we also release the recorded *rosbags*

[37] https://forums.developer.nvidia.com/t/222506/5

(a) Original frame (b) Re-generated frame (c) Difference between *(a)* and *(b)* (d) *(a)* w/o dynamic objects and after a light color change (e) *(a)* observed from a novel viewpoint

Figure 3.8: Examples of RGB and depth maps generated using the experiment repetition functionality of GRADE. In the first column, we show the original RGB and depth frames. The second column depicts the regenerated sensor readings captured at the *same* location. In column *c* we show the difference between the two corresponding frames in the RGB and depth domains, i.e. the difference between columns (a) and (b). Columns *d* and *e* show the same scene with a different lighting condition, first using the same viewpoint but hiding all the dynamic objects (d) and then changing the camera viewpoint to a different orientation (e). In column *e*, the depth map is inverted for clarity. The images are best viewed in color.

Figure 3.9: An example of the data generated using our simulation framework GRADE. Top row, left to right: rendered RGB image, corresponding depth map, optical flow, and surface normals. Bottom row, left to right: 2D bounding boxes, semantic instances, semantic segmentation, and SMPL shapes. The images are best viewed in color.

with IMU readings, TF tree, joint states, low-resolution RGBD images, and the robot's state. For convenience, IMU readings, camera pose, and robot pose, which we originally stored in the *rosbags*, are also provided independently from them as *numpy* arrays. For each experiment, we provide the initial configuration of each asset, the state of the random number generator used, the USD file of the simulation, and other accompanying information necessary to replay the experiment. Other data, such as normal vectors and optical flow, can be generated using the experiment repetition tool.

All the USD files of the animated sequences and environments can be freely downloaded upon acceptance of the necessary licenses, or generated from scratch when necessary (ShapeNet objects). The source code is fully open source. For convenience, we also release the pre-processed data to test Dynamic V-SLAM methods and the data relative to each experimented approach reported in this thesis in Section 4.3. Finally, we release the checkpoints of the trained networks, the data used to train them, as well as the labels for the TUM RGB-D dataset. A summary of the datasets used for the deep learning experiments, their training and validation splits, and the notation used for these can be seen in Table 3.5. Some examples of the generated data, as well as external views of the drone in the environment, can be observed in Figure 3.1[right], Figures 3.3a to 3.3d, 3.5 and 3.9.

## 3.6 Experiments

In this section, we first analyze the experiment repetition module in Section 3.6.1 to verify that the re-generated poses, images, and depth maps correspond to those of the original experiment. We then evaluate how well the generated data can address the syn-to-real gap in Section 3.6.2. In the next chapter, we will also benchmark Dynamic Visual SLAM methods in Section 4.3 to verify the usability of the simulated data to evaluate such approaches. There, we will study their limitations and the relation between their performance and the underlying deep detection and segmentation models.

| | $x$ | $y$ | $z$ | roll | pitch | yaw |
|---|---|---|---|---|---|---|
| Unit | m | m | m | rad | rad | rad |
| Mean | −3.50e−6 | −9.00e−7 | −1.94e−7 | −4.61e−7 | −8.70e−7 | −4.48e−7 |
| Std. Dev. | 2.75e−5 | 3.06e−5 | 2.71e−5 | 2.93e−5 | 2.90e−5 | 2.90e−5 |

Table 3.4: Evaluation of the precision of the robot poses obtained using the experiment repetition tool. For each component, we report the mean and the standard deviation of the difference between the repeated and the original values computed over 3601 instances.

### 3.6.1 Experiment Repetition Tool

To test the exactness of the experiment repetition procedure, we employ a 60-second experiment and compare the precision of: i) position and orientation values over 3601 poses, and of the re-rendered ii) RGB, and iii) depth maps over 1800 samples. Table 3.4 reports the mean and standard deviation of the difference between the originally recorded and re-executed robot poses for each component. The results show that the average deviation is on the order of $1e{-}7$ meters, demonstrating the high precision of our approach. To verify the rendering consistency, we then compute the average of the RGB image structural similarity (SSIM) [238] between the re-rendered and the original frames. The resulting average SSIM is 99.6% with a 0.15% standard deviation, which indicates near-perfect similarity. Finally, we evaluate the precision of the re-computed depth maps using the mean average difference and SSIM index with respect to the original ones. The mean of the average difference between the original and repeated depth maps is 0.0015 meters with a standard deviation of 0.0019 meters. Note that this is mostly due to aliasing effects and pixel values that lie alongside the borders of the objects. Indeed, the corresponding depth images have a structural similarity attested at 99.8% with a 0.23% standard deviation.

### 3.6.2 Syn-to-Real Transfer Learning

Our objective is to demonstrate that synthetic data generated with GRADE successfully captures real-world features and enables the training of models that generalize well over real images. To this end, we evaluate GRADE's syn-to-real transfer capabilities using two popular neural networks: YOLOv5 [102] and Mask R-CNN [84]. Our objective is the detection and segmentation of humans. We train the networks in three modalities: 1) from scratch with both synthetic and real-world data, 2) fine-tuning with real-world images the networks pre-trained on synthetic data, and 3) using datasets of mixed synthetic and real-world data, indicated with a '+' sign between the datasets' acronyms.

To train YOLO and Mask R-CNN, we use both i) a subset of the generated dataset, which we will refer to as **S-GRADE**, and ii) the complete dataset, i.e. **A-GRADE**. Images with a high probability of being occluded by flying objects due to peculiar depth and/or color information are automatically discarded. S-GRADE consists of 18K frames, with dynamic humans and without flying objects. Of those, 16.2K have humans in them and 1.8K are only background. A-GRADE contains all available data, i.e. 591K images, of which 362K have humans. We augment the images of S-GRADE using a random rolling shutter noise model ($\mu = 0.015$, $\sigma = 0.006$) and a fixed exposure time of 0.01 following [260]. To augment A-GRADE, we use a random exposure time between 0 and 0.1 seconds for each sequence and update the segmentation masks and bounding boxes to account for the additional motion blur. This is unnecessary for S-GRADE noisy images as the noise is much lower due to the shorter (and fixed) exposure time.

The real-world data is from the COCO dataset and the *fr3/walking* sequences of TUM

| | Description | Train images | Val. images |
|---|---|---|---|
| CH | COCO's images containing humans | 64115 | 2693 |
| S-CH | COCO random subset containing humans | 1256 | 120 |
| TH | TUM RGB-D *fr3* walking sequences | — | 3580 |
| S-GRADE | Subset of our synthetic data | 16K | 2K |
| A-GRADE | Our full synthetic data | 473K | 118K |
| $*$ -EX | Model trained on $*$ and saved at epoch X | | |
| BASELINE | Official model weights trained on COCO | | |

Table 3.5: Legend of the nomenclature used in the training and evaluation of the syn-to-real testing of the humans' detection and segmentation models presented in Section 3.6.2. In the first column, we indicate the abbreviation used. In the second column, we report its brief description. When needed, the last two columns contain the number of samples in the training and validation sets, respectively.

RGB-D [211]. From COCO, we utilize only the subset of data containing humans in the frame and will call this dataset **CH** (**C**OCO-**H**umans) from now on. CH contains 64115 training and 2693 validation images. From those, we randomly sample 1256 training and 120 validation images, totaling $\sim 2\%$ and $\sim 4\%$ of CH training and validation sets. Here, we call this subset **S-CH** and use it to understand how the networks perform with limited real-world data. The *fr3/walking* sequences consist of 3579 images with people, 5362 instances, and 130 background samples. We manually label those with precise bounding boxes and segmentation masks using the free version of Roboflow[38] and release this data publicly. We will use **TH** to indicate this data in our tests. CH exhibits high variability in human representation, including outdoor scenes, crowds, and diverse clothing. In contrast, the TH dataset aligns with our synthetic data, focusing on indoor dynamic sequences. We evaluate the performance with the COCO standard metric (mAP@[.5, .95], or mAP) and the PASCAL VOC's metric (mAP@.5, or mAP50). Note that, differently from PeopleSansPeople [59], we save the best checkpoint based on the training dataset's validation set, i.e. not using CH or its subset. We do not perform any hyperparameter tuning and use the default network settings. Our baseline models (called BASELINE henceforth) are the networks trained on the full COCO dataset. A recap of these datasets and the notation used in the next sections is in Table 3.5.

### Human Detection with YOLOv5

We trained YOLOv5s using its default parameters and for the standard 300 epochs. The results are reported in Table 3.6.

We first analyze the models trained only on single datasets. When evaluated on CH validation data, the model trained from scratch with the S-GRADE dataset exhibits lower

[38] https://roboflow.com/

| | | CH | | TH | |
|---|---|---|---|---|---|
| (Pre-)Training set | Fine-tuning set | mAP50 | mAP | mAP50 | mAP |
| BASELINE | — | 0.753 | 0.492 | 0.916 | 0.722 |
| S-CH | — | 0.492 | 0.242 | 0.661 | 0.365 |
| S-GRADE | — | 0.206 | 0.109 | 0.616 | 0.425 |
| S-GRADE-E50 | — | 0.234 | 0.116 | 0.683 | 0.431 |
| A-GRADE | — | 0.176 | 0.093 | 0.637 | 0.459 |
| A-GRADE-E50 | — | 0.282 | 0.154 | 0.798 | 0.613 |
| S-GRADE | S-CH | 0.561 | 0.302 | 0.744 | 0.488 |
| A-GRADE | S-CH | 0.540 | 0.299 | 0.762 | 0.514 |
| S-GRADE | CH | **0.801** | *0.544* | 0.931 | *0.778* |
| A-GRADE | CH | *0.797* | 0.542 | 0.932 | **0.786** |
| S-GRADE + S-CH | — | 0.590 | 0.334 | 0.855 | 0.648 |
| A-GRADE + S-CH | — | 0.527 | 0.289 | 0.801 | 0.597 |
| S-GRADE + CH | — | **0.801** | **0.547** | **0.938** | **0.786** |
| A-GRADE + CH | — | 0.764 | 0.503 | *0.936* | *0.778* |

Table 3.6: YOLOv5s bounding box evaluation results. We report the mAP50 and mAP over the specified validation set. We put in bold the best result and in italics the second best. The BASELINE is obtained using the officially released model of YOLOv5s trained on the full COCO dataset.

precision than the one trained solely with S-CH. However, on TH data, these models show comparable performance, with the network trained solely on S-GRADE achieving approximately $\sim 5\%$ lower mAP50 but 6% higher mAP. The network pre-trained with S-GRADE and then fine-tuned on S-CH shows a significant performance improvement with respect to the model trained using only S-CH. Specifically, this increase is around 6% on the CH validation (both metrics), and of $+8\%$ and $+12\%$ on the TH dataset mAP and mAP50, respectively. Similarly, fine-tuning S-GRADE with the full CH dataset surpasses the baseline results by $\sim 5\%$ in both metrics on CH, and $\sim +2\%$ mAP50 and $\sim +5\%$ mAP on the TH dataset. Models that use the A-GRADE dataset during training exhibit comparable or slightly worse performance on the CH validation set than the ones that use S-GRADE data (less than 3%). At the same time, using A-GRADE in place of S-GRADE data yields better performance on the TH dataset ($\sim 3\%$). We can therefore say that the models that use A-GRADE or S-GRADE during training tend to perform well on indoor human detection and do not generalize well to CH. This is likely due to the lack of examples in the synthetic dataset that can correctly represent the data distribution of CH. Interestingly, the models evaluated using the checkpoint saved at the 50th training epoch, identified as E50 in Table 3.6, consistently outperform the corresponding models trained only with A-GRADE or S-GRADE in all metrics and datasets. Moreover, A-GRADE-E50, when tested on TH data, outperforms models trained from scratch on

both synthetic and S-CH datasets or fine-tuned on S-CH, achieving a remarkable 79.8% mAP50. This indicates that there would be an advantage of using the real-world data during validation, as done in PeopleSansPeople [59]. However, this would prevent us from correctly evaluating the performance of using solely the synthetically generated images by introducing a bias when using *only* synthetic data.

Training using mixed synthetic and real-world datasets generally outperforms the corresponding pre-training and fine-tuning strategy using the same datasets. For example, the results on the TH dataset increased by up to $\sim 11\%$ mAP50 and $\sim 16\%$ mAP with the model trained on S-GRADE+S-CH. The only significant improvement on the CH validation set is observed with the S-GRADE+S-CH data, likely due to the difference in cardinality of the combined datasets and the data distribution of the CH data.

Overall, the best-performing model we obtained is S-GRADE+CH with an improvement over the baseline of $\sim 5\%$ on both metrics on the CH dataset, and 2% and 6% on the TH dataset. These results indicate how the data generated with GRADE can be used both to address the syn-to-real gap and improve the results of the models trained only on real-world data.

### Human Detection and Segmentation with Mask R-CNN

We use the *detectron2* [242] implementation of Mask R-CNN, using a 3x training schedule[39] and a ResNet50 backbone. We use the default steps (210K and 250K) and maximum iterations (270K) parameters with four images per batch when training A-GRADE and CH. We reduced those to 60K, 80K, and 90K when training S-GRADE and to 80K, 108K, and 120K and 2 images per batch for S-CH due to their relatively small size. We evaluate the models every 2K iterations and save the best one by comparing the mAP50 metric on each of the two tasks, detection and segmentation. Due to the size of the A-GRADE dataset, we evaluate the model trained from scratch on this data every 3K iterations. We save the best model separately for each task and evaluate its accuracy using 0.05 and 0.70 confidence thresholds. Since the training and evaluation schedules greatly impact the performance of this network[40], we also train from scratch the same network with the CH data using our configuration.

The results presented in Table 3.7 are similar to those obtained with YOLO in the previous section. However, unlike the previous case, fine-tuning with the entire CH dataset does not improve upon the official baseline. In contrast, combining A-GRADE or S-GRADE with CH generally results in higher mAP and mAP50 than the model trained solely on CH data, emphasizing *both* the impact of training procedures and our data.

These tests also show that, when using the A-GRADE dataset, we can consistently outperform the corresponding model that uses S-GRADE in both datasets and tasks. At the same time, the model trained only with A-GRADE synthetic data performs worse

[39] https://github.com/facebookresearch/detectron2/blob/main/configs/Misc/scratch_mask_rcnn_R_50_FPN_3x_gn.yaml

[40] https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md

| | | Threshold 0.7 | | | | | | | | Threshold 0.05 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Detection | | | | Segmentation | | | | Detection | | | | Segmentation | | | |
| | | CH | | TH | | CH | | TH | | CH | | TH | | CH | | TH | |
| (Pre-)Training set | Fine-tuning set | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP |
| BASELINE | — | 0.727 | 0.504 | 0.860 | 0.709 | 0.723 | 0.445 | 0.870 | 0.652 | 0.852 | 0.567 | 0.910 | 0.738 | 0.828 | 0.493 | 0.913 | 0.675 |
| CH | — | *0.693* | 0.471 | 0.829 | 0.653 | 0.681 | 0.410 | 0.838 | 0.584 | 0.829 | 0.537 | 0.898 | 0.692 | 0.801 | 0.461 | 0.890 | 0.611 |
| S-CH | — | 0.340 | 0.161 | 0.526 | 0.250 | 0.351 | 0.155 | 0.543 | 0.231 | 0.439 | 0.195 | 0.610 | 0.282 | 0.392 | 0.168 | 0.568 | 0.241 |
| S-GRADE | — | 0.128 | 0.064 | 0.563 | 0.312 | 0.100 | 0.043 | 0.509 | 0.264 | 0.167 | 0.077 | 0.637 | 0.343 | 0.117 | 0.048 | 0.561 | 0.283 |
| A-GRADE | — | 0.202 | 0.115 | 0.727 | 0.502 | 0.178 | 0.088 | 0.709 | 0.408 | 0.269 | 0.140 | 0.784 | 0.531 | 0.214 | 0.100 | 0.749 | 0.425 |
| S-GRADE | S-CH | 0.428 | 0.232 | 0.708 | 0.412 | 0.401 | 0.195 | 0.665 | 0.374 | 0.518 | 0.265 | 0.748 | 0.432 | 0.465 | 0.216 | 0.694 | 0.387 |
| A-GRADE | S-CH | 0.450 | 0.262 | 0.736 | 0.489 | 0.460 | 0.231 | 0.758 | 0.449 | 0.560 | 0.303 | 0.788 | 0.515 | 0.515 | 0.247 | 0.780 | 0.458 |
| S-GRADE | CH | *0.693* | *0.474* | 0.858 | *0.679* | *0.682* | *0.415* | *0.858* | *0.611* | *0.833* | *0.539* | *0.916* | *0.713* | *0.805* | *0.467* | *0.905* | *0.633* |
| A-GRADE | CH | **0.714** | **0.489** | **0.869** | **0.696** | **0.710** | **0.430** | **0.869** | **0.638** | **0.843** | **0.550** | *0.916* | **0.728** | **0.813** | **0.476** | **0.908** | **0.660** |
| S-GRADE + S-CH | — | 0.297 | 0.154 | 0.650 | 0.381 | 0.268 | 0.126 | 0.608 | 0.321 | 0.408 | 0.194 | 0.724 | 0.417 | 0.344 | 0.149 | 0.661 | 0.346 |
| A-GRADE + S-CH | — | 0.300 | 0.168 | 0.791 | 0.561 | 0.283 | 0.138 | 0.746 | 0.467 | 0.384 | 0.200 | 0.842 | 0.588 | 0.335 | 0.155 | 0.779 | 0.483 |
| S-GRADE + CH | — | 0.683 | 0.463 | *0.859* | 0.676 | 0.671 | 0.401 | 0.849 | 0.603 | 0.821 | 0.528 | **0.917** | 0.709 | 0.790 | 0.452 | 0.896 | 0.626 |
| A-GRADE + CH | — | 0.563 | 0.356 | 0.846 | 0.659 | 0.540 | 0.306 | 0.846 | 0.587 | 0.713 | 0.422 | 0.903 | 0.689 | 0.669 | 0.355 | 0.888 | 0.608 |

Table 3.7: Mask R-CNN detection and segmentation results using both thresholds, i.e. 0.7 and 0.05. For both tasks, we report the mAP50 and mAP over the specified validation set. We put in bold the best results and in italics the second best. The BASELINE is obtained using the officially released model of Mask R-CNN trained on the full COCO dataset. We also report the model trained on CH using our training and validation schedules for a fairer comparison.

than the one trained only with S-CH real-world data when evaluated on CH. Still, the result is reversed on the models validated on the TH dataset, with one trained only with A-GRADE outperforming the one trained only with S-CH data of up to $\sim 25\%$. This is true considering both tasks and both confidence thresholds.

Overall, the best model is obtained by pre-training on A-GRADE and fine-tuning on CH. This training strategy yields an improvement of $\sim 2\%$ and $3-5\%$ on CH and TUM datasets, respectively. The second-best model is the one trained on S-GRADE and fine-tuned on CH. The models trained on mixed data, in general, perform worse than the corresponding fine-tuned counterparts on the CH dataset while achieving similar results on the TUM dataset. These differences, also with respect to YOLO findings, are likely due to imbalanced synthetic and real-world data, and the scheduling, which greatly affects the number of epochs. Indeed, the model trained on the mixed S-GRADE+CH data performs similarly to the model pre-trained on S-GRADE and fine-tuned on CH.

Notably, unlike prior work such as [12], our approach achieves strong (although not perfect) generalization to real-world images using only synthetic data, even without incorporating real-world data in the validation set. Similar to their findings, mixing datasets leads to improved performance over the baseline. However, an equally crucial factor is the visual realism of the simulation itself, as one aims to avoid retraining perception models for every new task or system verification. Moreover, for better generalization, a system trained solely on synthetic data can be applied to out-of-distribution tasks where annotated datasets or large volumes of real images are unavailable [20, 187, 271]. Our results highlight the effectiveness of our approach in leveraging synthetic data for robust generalization. This reinforces the potential of our method for applications where real annotated data is scarce or unavailable.

## 3.7 Conclusions

GRADE is a novel, flexible solution for simulating robots in photorealistic dynamic environments, enabling efficient research, development, and benchmarking of autonomous robotic systems. GRADE addresses the limitations of previous robotics and vision-focused simulation frameworks by providing a streamlined system for simulation set-up and management, ground truth data generation, offline and online robot testing, as well as benchmarking of robotics and visual-based (learning) methods in physical and photo-realistic environments. This is achieved through the exploitation, integration, and expansion of Isaac Sim's capabilities via customizable asset preparation and placement, data saving and processing procedures, and robot preparation, set-up, and control.

We demonstrate GRADE flexibility by employing it in different case studies, ranging from simple visual data generation in physics-less simulations to heterogeneous multi-robot experiments managed by Active SLAM frameworks. Unlike previous systems that leveraged Isaac Sim, GRADE is not focused only on providing a closed framework for specific robots or applications, e.g. benchmarking V-SLAM systems. Instead, it is built

as close as possible to the low-level APIs of Isaac Sim, allowing for finer control and customization over the experiments. All the code and the data generated through GRADE for our experiments are provided as open-source for the benefit of the community.

With GRADE, we provide the first method allowing for precise programmatic experiment repetition with adaptable surroundings in physics-enabled simulations. Data can now be modified or expanded in simple and effective ways after the simulation happens by, for example, changing surroundings conditions (e.g. removing or adding dynamic objects) or adding new sensors (e.g. a stereo camera). Unlike previous systems, our approach is not limited by fixing seed numbers or rigid simulation conditions. Instead, it extends beyond simple changes (e.g. lighting adjustments), enabling substantial modifications to the simulation environment. This is an important step towards flexible testing, higher robustness, and thus better generalization, helping reduce the sim-to-real gap.

The strong syn-to-real performance of the learned human detection and segmentation tasks demonstrates the effectiveness of our simulation and its visual realism. Notably, even without incorporating highly detailed human models with features like hair, shoes, or high-resolution textures, the generated data proves sufficiently realistic to significantly enhance network performance when combined with real-world images. Furthermore, training exclusively on GRADE synthetic data achieves results that closely match the baseline in indoor environments, greatly reducing the need for extensive data collection and manual annotation and clearly addressing the sim-to-real gap. While commercial synthetic clothed human models, such as RenderPeople or CLO, could further enhance realism and potentially improve training outcomes, we deliberately avoid their use to ensure the reproducibility and open redistribution of our generated data, which is essential for open research. Moreover, we want to emphasize the fact that we obtain these results *without* introducing or leveraging any domain-adaptation technique, as opposed to previous approaches. Using GRADE, we can address the syn-to-real gap by generating a high amount of realistic and diverse data.

# Chapter 4

# Dynamic Visual SLAM: Benchmarking, Challenges, and Pixels

In this chapter, we investigate the role of perception in dynamic environments and its relation to V-SLAM. Our goal is to move beyond static scene assumptions and towards robust Dynamic V-SLAM by leveraging GRADE, our synthetic data generation framework.

We begin by outlining the problem in Section 4.1 and reviewing the state of the art in Dynamic V-SLAM (Section 4.2). Most Dynamic V-SLAM approaches rely on semantic information, geometric constraints, or optical flow to filter out dynamic elements. However, a significant limitation in this field is the scarcity of datasets designed for dynamic environments. Many existing methods are evaluated only on a handful of available datasets, which often contain a limited range of motion patterns and object interactions. This poses challenges for assessing generalization and robustness. Moreover, collecting diverse real-world sequences with controlled, repeatable dynamic interactions is inherently difficult, costly, and sometimes unsafe — especially in cluttered or highly dynamic environments. Synthetic datasets and simulations, therefore, play a crucial role in filling this gap by providing controlled, diverse, and challenging scenarios for evaluation.

In this chapter, we seek to use GRADE to solve this issue. Hence, as a first step, we benchmark V-SLAM methods in static and dynamic environments (Section 4.3.1). This is critical for establishing a baseline and ensuring that data simulated through GRADE can reproduce expected results in well-understood scenarios. Once this validation is complete, we conduct a thorough evaluation of representative state-of-the-art Dynamic V-SLAM methods using simulated runs in dynamic indoor settings. Our results reveal that these methods frequently suffer from noise, trajectory drift, and tracking failures, underscoring the challenges posed by dynamic environments. We further analyze the impact of detection and segmentation performance on two Dynamic V-SLAM methods, using both synthetic and real-world sequences (Section 4.3.2). Contrary to common belief, we find that the best-performing deep learning models do not always yield the best results in Dynamic V-SLAM scenarios. Through this analysis, we build intuition regarding the limitations of existing approaches and identify some key failure points: noisy motion estimations, hallucinations, and the misinterpretation of object motion. Additionally, we highlight the limitations of existing methods, including reliance on predefined static/dynamic thresholds, preselected dynamic object classes, and an inability to detect unknown moving objects, which restricts their generalization capabilities.

To address these challenges, we introduce DynaPix (Section 4.4), a semantic-free Dynamic V-SLAM system that estimates per-pixel motion probabilities and integrates them into an improved pose optimization process. Our method computes motion probabilities using static background differencing and optical flow on splatted frames. These probabilities guide map point selection and are incorporated through weighted bundle adjustment within the tracking and optimization modules of ORB-SLAM2. We rigorously evaluate DynaPix on both the GRADE and TUM RGB-D datasets in Section 4.5, demonstrating significantly lower trajectory errors and extended tracking durations in both static and dynamic environments, as highlighted in our conclusions (Section 4.6). The source code, datasets, and results of DynaPix are available at https://dynapix.is.tue.mpg.de/.

## 4.1 Introduction

Visual SLAM algorithms have undergone significant development [28] and found wide-ranging applications in various scenarios, including service robots [200], autonomous vehicles [24], and augmented reality devices [101]. However, recall that most V-SLAM frameworks are developed under a *static-world* assumption [26, 28, 162, 261] and use visual features as a means to estimate the robot's (camera) movements (Section 1.3). The presence of dynamic objects violates such an assumption and causes degradation in both **estimation accuracy and system robustness** [26, 200], which limits their widespread deployment in real-world scenarios. Indeed, addressing this problem is necessary to develop robots that can safely act in dynamic environments.

To resolve this issue, different visual feature removal procedures have been introduced. Many are based on the detection or segmentation of dynamic object classes [17,255,266]. Several works instead use optical flow [135,262], depth clustering [194,258], or learning-based visual odometry methods [21, 199]. To increase robustness, other approaches combine geometric and semantic information [17, 42, 90], thus still heavily relying on semantic cues. Nonetheless, research in SLAM for dynamic worlds has still limited (although increasing) traction, mainly due to difficulties in simulating data and the inherent danger of directly testing an autonomous method in the real world. Thus, given the limited availability of testing sequences and environments, those methods often fail when applied to different situations or environments [234], as we will show in Section 4.3. However, thanks to GRADE's advancements, we can now benchmark current state-of-the-art Dynamic SLAM methods safely in realistically looking simulations. To do so, we select some representative dynamic sequences from the dataset we introduced in Section 3.5. We then use these simulation experiments to benchmark popular indoor Dynamic V-SLAM algorithms. First, we evaluate such methods on static synthetic sequences to demonstrate that the data is usable by said frameworks, that visual features can be correctly extracted, and that the robot trajectory is tracked. To do so, we re-render the selected sequences while disabling the dynamic content using our experiment repetition tool (Section 3.4.3). Then, we use the selected dynamic sequences to evaluate such methods and show how these would fail if the data differs from the few currently used real world benchmarking datasets. Moreover, since those approaches heavily rely on detection and segmentation networks, we also experiment with the different trained models of YOLOv5 [102] and Mask R-CNN [84] presented in Section 3.6.2. With these, we show that the model with the highest accuracy does not always attain the best ATE result when applied to the Dynamic V-SLAM method.

By analyzing these results, we notice that, in general, those methods often suffer from failures related to prediction accuracy, noise, and/or imprecise estimations. Detection-based methods remove features belonging to the static environment within the detected bounding box. Segmentation is often approximate, especially around object borders or thin regions. Both detection- and segmentation-based methods have limited generalization capabilities, e.g. to different moving objects, as they rely on an a-priori choice of

dynamic categories and on the networks' performance. Furthermore, optical flow suffers from exploding magnitudes, especially in scenes with repeating patterns or featureless areas. Finally, learning-based methods often cannot generalize well to different cameras or scenarios [235]. Moreover, most methods mask the *entirety* of the objects without considering whether all or just part of them are moving — a common scenario for animals or humans that can move also just a limb or a hand. They also adopt simple binary moving/static masks instead of more informative blended probability distributions. This implies that *all* features belonging to the identified moving region are rejected, independently of whether those are effectively moving or not. This can lead to frequent tracking losses and instability due to the exclusion of valuable information or the excessive filtering that can reject the majority of the features extracted from the frame. This can happen especially when objects that might be moving occupy most of the image, or when there are many static features extracted from these objects that could be leveraged [199]. Notably, we also realized that all these methods fail not only to correctly but also to *fully* track our experiments, without being able to recover from failures. This is a metric often overlooked by the community. However, the amount of time the SLAM framework is capable of estimating a given trajectory is an essential indication of the robustness that helps to contextualize the results.

These considerations are at the base of DynaPix, our novel Dynamic V-SLAM approach, that we present in Section 4.4. Building on top of ORB-SLAM2 [152], DynaPix is based on a pixel-wise motion probability estimation technique for dynamic indoor environments. The intuition is that the SLAM backend depends on a limited number of image features and keypoints associated with *single* pixels. Thus, every extracted feature can contribute to the process through a weighting factor based on its motion state, regardless of whether it belongs to a *supposedly* dynamic object or not. In the first stage, we introduce a novel static/dynamic differencing method on both neighboring frames and estimated optical flows. We use these to compute per-pixel *movable* and *moving* probability estimations, respectively. Those estimations are then combined to derive a per-pixel *motion* probability that we use as a weighing factor in the SLAM backend, both within the map points selection and the weighted bundle adjustment (BA) procedures. To validate our method, we perform extensive evaluations and ablation studies on both the TUM RGB-D [211] and the GRADE datasets, computing the absolute trajectory error (ATE) metric and tracking rate for each experiment.

To summarize, our novel contributions are: i) the benchmarking and validation of GRADE in the context of V-SLAM; ii) the introduction of a novel per-pixel motion probability computation, that does not rely on semantic information, thresholds, or appearance-based detectors; iii) the integration of these probabilities in the tracking and optimization modules of a visual SLAM system; iv) the introduction of an extensive (and extendable) dataset of dynamic sequences following what we introduced in Chapter 3. Overall, our rigorous experiments show that our approach achieves lower trajectory errors and higher tracking times in both synthetic and real world sequences, effectively reducing the influence of dynamic factors.

## 4.2 Related Works

### Moving Object Segmentation

Typical approaches apply geometric constraints to eliminate dominant motion in images caused by camera movement, and then cluster regions that follow different rigid transformations [145]. These methods often involve motion detection with depth clusters [95, 258], multi-view epipolar geometry [55], or optical-flow-based methods [18, 135]. Learning-based flow estimation networks have also been investigated [56, 215]. However, these approaches often rely on predefined thresholds as detection criteria [250], struggle to adapt to diverse environments [26], and suffer from inaccurate flow estimation. Deep Neural Networks (DNNs) can also be employed to detect or segment moving objects [17, 131, 255, 266]. However, those approaches require a set of *fixed and predefined* classes. Therefore, they may mask completely stationary objects (e.g. parked cars), as well as the entirety of objects that are only partially moving [90] (e.g. humans sitting). Moreover, they heavily rely on the chosen network's performance and can suffer from imprecise segmentation or incorrect object classifications.

Methods combining geometric constraints, semantic information, and/or flow estimation apply a trade-off in identifying moving regions and are still typically based on learning-based frameworks. Therefore, they depend on their training data [134, 175, 250], aim to segment the scene into binary categories (moving/static), and mask entire object instances even if they are (partially) static. Moreover, these methods are often validated in outdoor scenarios, where most observations are static, thus limiting their applicability to dynamic indoor environments. In DynaPix, we avoid considering only full objects and any predefined threshold or segmentation class to mark pixels as static/dynamic. Instead, we generate a moving probability value for each pixel by combining background differencing with optical flow information from splatted views.

### Dynamic Visual SLAM

Most Dynamic V-SLAM methods are *de-facto* enhanced versions of classic SLAM frameworks [152, 169] combined with motion segmentation techniques. Dynamic regions are either separately tracked [8, 16, 230] or discarded as outliers [17, 41, 117, 237, 255, 266] to reduce the negative effects on pose estimation. DynaSLAM [17] combines Mask R-CNN [84] and multi-view geometry to process moving objects, DS-SLAM [255] applies a lightweight SegNet [6] to obtain segmented masks, and Dynamic-VINS [131] uses YOLO-based detections. However, these methods suffer from wrongful detections, retain few features when there are many dynamic objects, and fail to identify moving objects outside the selected classes, thus leading to frequent tracking failures [26].

Methods that integrate motion probability, like Detect-SLAM [266] which derives it from object detections, also fuse semantic attributes with geometric ones. DP-SLAM [117] and Cheng et al. [41] put forward dynamic region removal techniques

within the Bayesian framework to enhance motion probability updates. DE-SLAM [244] instead restores more static features for pose estimation of adjacent frames, even within selected dynamic categories. All of these, however, discard the dynamic features directly without retaining motion probabilities in the SLAM optimization process. Close to DynaPix, [83, 90, 267] use motion probability as weights in the BA procedure (introduced in Section 1.3.3). However, these are used *only* within the tracking module of ORB-SLAM2 and are continuously discounted during the optimization process, making their effect negligible over subsequent iterations. Moreover, they often require additional optimization steps to attain acceptable results. In contrast, our method updates the motion probabilities following subsequent observations and propagates them throughout the entire SLAM framework. In concurrent work, Liu et al. [130] proposed a method based on intensity change, sparse optical flow, and clustering to compute motion probabilities. However, their method heavily relies on thresholds due to the noise from those components, limits consideration to the current and the last frames, and uses intensity errors rather than our background reconstruction. Furthermore, similar to previous studies, these approaches often neglect to report the effective tracking time in their experiments. Indeed, the vast majority of evaluations focus only on trajectory errors (e.g. ATE and RPE). The total time a method tracks a trajectory is often overlooked, despite it being necessary for a thorough assessment of SLAM performance [26].

## 4.3 Benchmarking Dynamic V-SLAM

To perform these evaluations, we select eight different runs among all the sequences generated in Section 3.5. Each is 60 seconds long and comprises RGB-D (30 fps), IMU (240 Hz), and ground truth pose (60 Hz) data. The eight runs are divided and labeled as follows: two are recorded in static environments (**S**), two contain dynamic people and no flying objects (**D**), two have both people and flying objects (**F**), and two present an occlusion of the camera (**WO**). The occlusion creates some challenging, completely black frames during the experiments. For each one of these kinds (S, D, F, WO), the UAV is either i) kept horizontal, in which case it cannot perform roll and pitch movements, or ii) is free to move, to increase variability in our evaluations. If the robot is not free to move, we post-fix the label of the run with the letter **H**, i.e. SH, DH, FH, and WOH. To complete our evaluations, we also use our experiment repetition procedure to re-render all the dynamic sequences into their static counterparts by disabling all the dynamic assets and re-rendering RGB and depth data. We indicate those by post-fixing **-static**. This allows us to study the effects of the dynamic entities in our experiments. Statistics of the average camera (absolute) speed, average (absolute) acceleration, number of dynamic frames, and average portion of the dynamic frame belonging to dynamic people of the S, SH, D, DH, WO, WOH, F, and FH sequences are reported in Table 4.1. The average speed and acceleration are obtained from the ground truth values of the odometry recorded at 60 Hz. We note that Table 4.1 shows small velocity and accelera-

tion components in the roll and pitch axes for the DH experiment. However, in DH, the robot is constrained to remain horizontal, meaning that roll and pitch velocities should theoretically be zero. Upon inspection of the data, we confirmed that these deviations resulted from a minor collision between the robot and the environment, which introduced unintended but minimal rotational effects.

The ground-truth data generated by the simulator is processed before the evaluations to make it closer to real-world conditions. Depth data is first limited to 3.5 meters - a reasonable value when using, for example, a RealSense D435i. Then we limit it to 5 meters, to study the effect of the depth range on the SLAM methods. We evaluate the SLAM frameworks using the same data but enhanced with additional noise. The noise applied to the depth values is based on the RealSense noise model [79]. We apply random rolling shutter noise ($\mu = 0.015$, $\sigma = 0.006$) and blur (following [260]) to the RGB data. The IMU drift and noise parameters are taken from [71].

| | Average speed<br>$[x, y, z]$, m/s<br>[roll, pitch, yaw], rad/s | Average acceleration<br>$[x, y, z]$, m/s$^2$<br>[roll, pitch, yaw], rad/s$^2$ | Dynamic frames | Dynamic frames coverage (%) |
|---|---|---|---|---|
| FH | $[0.084, 0.091, 0.014]$<br>$[0.000, 0.000, 0.444]$ | $[0.425, 0.551, 0.123]$<br>$[0.000, 0.000, 0.698]$ | 1800 | 34.06 |
| F | $[0.244, 0.239, 0.139]$<br>$[0.189, 0.184, 0.436]$ | $[0.891, 0.882, 0.740]$<br>$[1.625, 1.564, 1.242]$ | 1247 | 11.55 |
| DH | $[0.255, 0.185, 0.046]$<br>$[0.029, 0.014, 0.476]$ | $[1.011, 0.938, 0.416]$<br>$[0.291, 0.158, 1.659]$ | 959 | 8.06 |
| D | $[0.293, 0.264, 0.127]$<br>$[0.288, 0.269, 0.422]$ | $[1.206, 1.248, 0.895]$<br>$[1.895, 1.797, 1.070]$ | 1196 | 10.01 |
| WOH | $[0.275, 0.255, 0.086]$<br>$[0.000, 0.000, 0.433]$ | $[0.949, 1.025, 0.553]$<br>$[0.000, 0.000, 0.725]$ | 1181 | 11.33 |
| WO | $[0.224, 0.249, 0.103]$<br>$[0.214, 0.220, 0.416]$ | $[1.011, 1.059, 0.634]$<br>$[1.892, 1.564, 1.278]$ | 1763 | 17.94 |
| SH | $[0.304, 0.254, 0.047]$<br>$[0.000, 0.000, 0.426]$ | $[0.965, 0.908, 0.273]$<br>$[0.000, 0.000, 1.018]$ | 0 | — |
| S | $[0.259, 0.236, 0.114]$<br>$[0.213, 0.228, 0.435]$ | $[0.915, 0.822, 0.538]$<br>$[1.592, 1.571, 1.013]$ | 0 | — |

Table 4.1: Motion and dynamic frames analysis for each one of the dynamic sequences used in our Dynamic V-SLAM benchmarking.

We will evaluate two static V-SLAM methods and four Dynamic V-SLAM approaches. Those are: i) RTAB-Map [114] and ii) ORB-SLAM2 [152], which do not explicitly address dynamic entities; iii) DynaSLAM [17], which uses Mask R-CNN to segment dynamic content; iii-iv) DynamicVINS [131] (in both its VO and VIO variations, abbreviated to DynaVINS here), which uses YOLO to detect it; v) StaticFusion [194], a non-learning method that performs RGB-D based clustering; and vi) TartanVO [234], i.e. a learned visual odometry system developed specifically for challenging scenarios.

For fairness, we choose to *not* modify the parameters of any of the SLAM approaches we benchmark. Despite that, we had to increase the number of extracted features to 3000 in both DynaSLAM and ORB-SLAM2 to allow for an easier and repeatable initialization in some sequences, and edit DynaVINS, taking inspiration from VINSFusion [170], to keep the system running in case of a tracking failure.

We report the absolute trajectory error (ATE) [26] and the total time a considered V-SLAM framework can successfully track the trajectory. The latter, expressed as tracking rate (TR), is a critical quantity to be considered. It helps the reader put ATE values in perspective whenever the tested method fails due to some featureless frames or occlusions, as the ATE alone cannot completely quantify the robustness [26]. The ATE has been computed using the standard TUM RGB-D evaluation tool. We perform ten different trials and report the mean and standard deviation of both metrics for each test.

### 4.3.1 Visual SLAM Performance

The results of testing the selected methods on the re-rendered static sequences are presented in Table 4.2. In Tables 4.3 and 4.4, instead, we report the dynamic experiments alongside the runs performed on the static S and SH sequences, whose depth was limited to 3.5 and 5 meters, respectively. The first and most noticeable result we can observe is that TartanVO and StaticFusion consistently exhibit high ATE and perfect TR. TartanVO's performance can be attributed to poor generalization capabilities due to the domain gap between its training data and GRADE scenes, along with its reliance on RGB information alone. In contrast, StaticFusion is sensitive to parameter tuning, as observed also in other studies, e.g. [186]. We can also observe how, generally, all methods perform well in SH and S, with low ATE and high tracking rates. Meanwhile, in the other sequences, the results vary a lot, though at least one method consistently achieves good performance. These results show that the data generated by GRADE can be used effectively to perform visual odometry and demonstrate, at the same time, the low adaptation capabilities of some of these algorithms. The low tracking rates of RTAB-Map on D-static, DH-static, and F-static are to be associated with events in which the system loses track of the odometry and resets, without recovering. Notably, while both ATE and TR vary across methods, the standard deviations are generally low.

For dynamic sequences, the good ATE results of certain methods can be misleading. For example, in four out of eight sequences without added noise where the depth data is limited to 3.5 meters (Table 4.3) DynaSLAM loses track of the trajectory for at least $\sim 27$ seconds, and up to 54 seconds. This highlights the importance of reporting *both* the tracking rate and the ATE errors when evaluating Dynamic SLAM methods, as we will also see in the next section. Although all the evaluated methods show compelling results when tested with common datasets like TUM RGB-D or EuRoC, they exhibit several limitations when tested on our data.

| | | DynaVINS — VO | | DynaVINS — VIO | | StaticFusion | | TartanVO | | DynaSLAM | | ORB-SLAM2 | | RTAB-Map | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| D-static | mean | 0.366 | 0.992 | 0.158 | 0.989 | 2.878 | 0.999 | 1.732 | 1.000 | 0.047 | 0.985 | 0.020 | 0.984 | 0.090 | 0.198 |
| | std | 0.330 | 0.002 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.051 | 0.021 | 0.003 | 0.020 | 0.000 | 0.000 |
| DH-static | mean | 1.548 | 0.737 | 6.768 | 0.993 | 2.745 | 0.999 | 1.173 | 1.000 | 0.008 | 0.107 | 0.005 | 0.189 | 0.086 | 0.655 |
| | std | 0.192 | 0.220 | 0.202 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.006 | 0.038 | 0.000 | 0.007 | 0.000 | 0.000 |
| F-static | mean | 1.176 | 0.825 | 1.344 | 0.982 | 2.823 | 0.999 | 3.765 | 1.000 | 0.621 | 0.841 | 0.327 | 0.904 | 0.049 | 0.317 |
| | std | 0.404 | 0.262 | 0.549 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.498 | 0.035 | 0.481 | 0.006 | 0.000 | 0.000 |
| FH-static | mean | 0.020 | 0.993 | 0.038 | 0.990 | 0.100 | 0.999 | 0.531 | 1.000 | 0.011 | 1.000 | 0.010 | 1.000 | 0.093 | 1.000 |
| | std | 0.002 | 0.000 | 0.002 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.004 | 0.000 | 0.000 | 0.000 |
| WO-static | mean | 0.806 | 0.990 | 0.201 | 0.976 | 0.062 | 0.999 | 2.437 | 1.000 | 0.036 | 0.793 | 0.040 | 0.752 | 0.120 | 1.000 |
| | std | 0.442 | 0.003 | 0.015 | 0.007 | 0.000 | 0.000 | 0.000 | 0.000 | 0.029 | 0.288 | 0.025 | 0.342 | 0.000 | 0.000 |
| WOH-static | mean | 1.473 | 0.984 | 0.139 | 0.985 | 1.469 | 0.999 | 2.610 | 1.000 | 0.012 | 0.538 | 0.015 | 0.538 | 0.208 | 1.000 |
| | std | 0.264 | 0.008 | 0.014 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.004 | 0.000 | 0.009 | 0.000 | 0.000 | 0.000 |

Table 4.2: ATE [m] and tracking rate (TR) obtained on the selected re-rendered static GRADE sequences (rows) grouped in their ground-truth and noisy versions. We report the mean and standard deviation over ten trials. The depth data is limited to 3.5 m.

| | | | DynaVINS — VO | | DynaVINS — VIO | | StaticFusion | | TartanVO | | DynaSLAM | | ORB-SLAM2 | | RTAB-Map | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| Ground-truth data | D | mean | 1.450 | 0.888 | 0.192 | 0.989 | 1.212 | 0.999 | 1.264 | 1.000 | 0.042 | 0.830 | 0.283 | 0.981 | 0.417 | 0.891 |
| | | std | 0.441 | 0.096 | 0.010 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.009 | 0.111 | 0.036 | 0.021 | 0.000 | 0.000 |
| | DH | mean | 1.582 | 0.644 | 8.020 | 0.993 | 1.664 | 0.999 | 1.259 | 1.000 | 0.011 | 0.097 | 0.005 | 0.179 | 0.091 | 0.654 |
| | | std | 0.468 | 0.301 | 0.282 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.007 | 0.028 | 0.001 | 0.007 | 0.000 | 0.000 |
| | F | mean | 1.532 | 0.841 | 2.057 | 0.980 | 2.866 | 0.999 | 4.132 | 1.000 | 0.858 | 0.440 | 0.294 | 0.565 | 0.086 | 0.219 |
| | | std | 0.504 | 0.230 | 0.478 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.184 | 0.115 | 0.151 | 0.228 | 0.000 | 0.000 |
| | FH | mean | 0.220 | 0.993 | 0.075 | 0.989 | 0.085 | 0.999 | 0.551 | 1.000 | 0.258 | 1.000 | 0.295 | 1.000 | 0.324 | 1.000 |
| | | std | 0.058 | 0.000 | 0.006 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.054 | 0.000 | 0.067 | 0.000 | 0.000 | 0.000 |
| | WO | mean | 1.219 | 0.910 | 0.582 | 0.957 | 2.807 | 0.999 | 2.473 | 1.000 | 0.090 | 0.079 | 0.157 | 0.197 | 0.275 | 0.197 |
| | | std | 0.291 | 0.092 | 0.072 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.011 | 0.000 | 0.007 | 0.000 | 0.000 | 0.000 |
| | WOH | mean | 1.474 | 0.808 | 0.223 | 0.981 | 1.980 | 0.999 | 2.361 | 1.000 | 0.013 | 0.538 | 0.011 | 0.538 | 0.088 | 0.569 |
| | | std | 0.548 | 0.120 | 0.027 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| | S | mean | 0.036 | 0.993 | 0.222 | 0.991 | 7.919 | 0.999 | 1.205 | 1.000 | 0.011 | 1.000 | 0.011 | 1.000 | 0.084 | 1.000 |
| | | std | 0.003 | 0.000 | 0.010 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 |
| | SH | mean | 0.029 | 0.993 | 0.119 | 0.991 | 0.594 | 0.999 | 2.395 | 1.000 | 0.011 | 1.000 | 0.012 | 1.000 | 0.089 | 1.000 |
| | | std | 0.005 | 0.000 | 0.007 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| Noisy data | D | mean | 1.362 | 0.899 | 0.693 | 0.989 | 2.278 | 0.999 | 1.356 | 1.000 | 0.061 | 0.554 | 0.725 | 0.877 | 0.401 | 0.606 |
| | | std | 0.355 | 0.090 | 0.086 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.033 | 0.002 | 0.052 | 0.035 | 0.000 | 0.000 |
| | DH | mean | 1.628 | 0.609 | 1.982 | 0.993 | 1.091 | 0.999 | 1.234 | 1.000 | 0.003 | 0.051 | 0.004 | 0.054 | 0.052 | 0.175 |
| | | std | 0.642 | 0.338 | 0.268 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| | F | mean | 2.039 | 0.855 | 2.431 | 0.975 | 3.992 | 0.999 | 4.223 | 1.000 | 0.212 | 0.255 | 0.142 | 0.258 | 0.047 | 0.191 |
| | | std | 0.575 | 0.193 | 1.616 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.037 | 0.037 | 0.017 | 0.028 | 0.000 | 0.000 |
| | FH | mean | 0.455 | 0.991 | 0.207 | 0.988 | 0.854 | 0.999 | 0.582 | 1.000 | 0.244 | 0.986 | 0.240 | 1.000 | 0.184 | 1.000 |
| | | std | 0.199 | 0.003 | 0.104 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.062 | 0.042 | 0.081 | 0.000 | 0.000 | 0.000 |
| | WO | mean | 1.219 | 0.844 | 1.085 | 0.955 | 2.213 | 0.999 | 2.380 | 1.000 | 0.099 | 0.079 | 0.181 | 0.197 | 0.329 | 0.197 |
| | | std | 0.116 | 0.207 | 0.275 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.009 | 0.000 | 0.006 | 0.000 | 0.000 | 0.000 |
| | WOH | mean | 1.364 | 0.910 | 0.560 | 0.981 | 1.826 | 0.999 | 2.399 | 1.000 | 0.032 | 0.536 | 0.021 | 0.536 | 0.118 | 0.569 |
| | | std | 0.541 | 0.115 | 0.086 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.011 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| | S | mean | 0.067 | 0.993 | 0.200 | 0.991 | 3.538 | 0.999 | 1.306 | 1.000 | 0.022 | 1.000 | 0.024 | 1.000 | 0.212 | 1.000 |
| | | std | 0.007 | 0.000 | 0.010 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 |
| | SH | mean | 0.073 | 0.993 | 0.693 | 0.991 | 4.184 | 0.999 | 2.517 | 1.000 | 0.017 | 1.000 | 0.018 | 1.000 | 0.092 | 1.000 |
| | | std | 0.008 | 0.000 | 0.408 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |

Table 4.3: ATE [m] and tracking rate (TR) obtained on the selected GRADE sequences (rows) grouped in their ground-truth and noisy versions. We report the mean and standard deviation over ten trials. The depth data is limited to 3.5 m.

This is irrespective of noise presence or differences in the depth limits. We can see that most of the experiments performed on noisy data are, as expected, slightly worse than those performed using ground-truth images and depth values. ORB-SLAM and DynaSLAM yield, for the most part, comparable results on both metrics. DynaVINS VO generally performs worse than its VIO counterpart, showing its reliance on the IMU sensor and the low robustness of the VO approach.

StaticFusion, counterintuitively, shows degrading performance when tested with the depth data limited to 5 meters, with generally not compelling results. DynaVINS appears, on average, to be the most stable method across different testing methodologies, remaining unaffected by changes in depth ranges or data noise. However, its results are not easily repeatable, as indicated by the standard deviation values. DynaVINS VIO seems to be the best-performing method if we consider ATE and TR jointly. However, despite the use of the IMU, the ATE obtained with DynaVINS VIO on the DH non-noisy sequence is over 8 meters for just a 60 s sequence, and just 1.582 meters for the VO counterpart, as reported in Table 4.3. Similar results can be observed in the S and SH sequences, where DynaVINS VO shows 3 to 10 times lower ATE than the one obtained with DynaVINS VIO, indicating how the method does not always benefit from using the IMU sensor. With certain combinations, SLAM methods perform worse on statically rendered sequences than on dynamic ones, like RTAB-Map when tested on the D static and dynamic sequences. This may be because, in some cases, the methods use features from undetected dynamic content to track camera movement better when facing featureless areas like plain walls.

### 4.3.2 Dynamic V-SLAM and Deep Learning relation

Here, we evaluate DynaVINS (VO) and DynaSLAM jointly with several models we trained in Section 3.6.2 using YOLOv5 and Mask R-CNN. We assess their performance on both synthetic sequences with the depth limited to 3.5 meters of GRADE (without noise) and the TUM *fr3/walking* sequences [211]. The results are presented in Table 4.5. For Mask R-CNN, we use the best-performing model on the *segmentation* task. The baseline results on the TUM RGB-D sequences are obtained using the baseline models. Note that we were unable to reproduce the published results for DynaVINS, especially the *rpy* and *static* sequences. This discrepancy can be attributed to several factors. First, there can be non-deterministic behavior due to variations in feature extraction, loop closure detection, and optimization. As a result, exact reproducibility across different runs is inherently challenging [17]. Additionally, differences in parameter tuning and the number of runs performed may have contributed to the observed discrepancies. Furthermore, variations in computing environments (e.g. CUDA versions, CPU, hardware configurations), compilation methods, and versions of installed libraries (e.g. OpenCV, YOLO) may influence the results. Note that for all sequences, we can observe specific runs that have performance that is similar to the published results and within the computed standard deviation over ten trials.

| | | | DynaVINS — VO | | DynaVINS — VIO | | StaticFusion | | TartanVO | | DynaSLAM | | ORB-SLAM2 | | RTAB-Map | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| Ground-truth data | D | mean | 1.429 | 0.713 | 0.195 | 0.989 | 22.374 | 0.999 | 1.264 | 1.000 | 0.046 | 0.805 | 0.251 | 0.995 | 0.547 | 0.995 |
| | | std | 0.275 | 0.265 | 0.009 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.009 | 0.070 | 0.053 | 0.000 | 0.000 | 0.000 |
| | DH | mean | 1.692 | 0.655 | 7.926 | 0.993 | 14.938 | 0.999 | 1.259 | 1.000 | 0.014 | 0.087 | 0.006 | 0.176 | 0.097 | 0.646 |
| | | std | 0.568 | 0.307 | 0.290 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.010 | 0.026 | 0.002 | 0.005 | 0.000 | 0.000 |
| | F | mean | 1.733 | 0.789 | 2.262 | 0.981 | 2.781 | 0.999 | 4.132 | 1.000 | 0.708 | 0.420 | 0.408 | 0.487 | 0.091 | 0.219 |
| | | std | 0.370 | 0.302 | 0.750 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.311 | 0.127 | 0.245 | 0.151 | 0.000 | 0.000 |
| | FH | mean | 0.424 | 0.991 | 0.073 | 0.989 | 0.059 | 0.999 | 0.551 | 1.000 | 0.274 | 0.987 | 0.223 | 1.000 | 0.200 | 1.000 |
| | | std | 0.290 | 0.003 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.072 | 0.038 | 0.083 | 0.000 | 0.000 | 0.000 |
| | WO | mean | 1.315 | 0.772 | 0.585 | 0.963 | 1.418 | 0.999 | 2.473 | 1.000 | 0.094 | 0.079 | 0.134 | 0.197 | 0.210 | 0.197 |
| | | std | 0.361 | 0.259 | 0.068 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.013 | 0.000 | 0.007 | 0.000 | 0.000 | 0.000 |
| | WOH | mean | 1.683 | 0.900 | 0.234 | 0.982 | 4.926 | 0.999 | 2.361 | 1.000 | 0.012 | 0.538 | 0.011 | 0.538 | 0.087 | 0.569 |
| | | std | 0.452 | 0.092 | 0.027 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 |
| | S | mean | 0.044 | 0.993 | 0.226 | 0.991 | 22.282 | 0.999 | 1.205 | 1.000 | 0.015 | 1.000 | 0.013 | 1.000 | 0.081 | 1.000 |
| | | std | 0.002 | 0.000 | 0.007 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| | SH | mean | 0.031 | 0.993 | 0.116 | 0.991 | 2.721 | 0.999 | 2.395 | 1.000 | 0.013 | 1.000 | 0.012 | 1.000 | 0.087 | 1.000 |
| | | std | 0.006 | 0.000 | 0.005 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| Noisy data | D | mean | 1.438 | 0.826 | 0.601 | 0.989 | 10.123 | 0.999 | 1.350 | 1.000 | 0.057 | 0.686 | 0.683 | 0.835 | 0.529 | 0.902 |
| | | std | 0.298 | 0.225 | 0.059 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.009 | 0.160 | 0.067 | 0.075 | 0.000 | 0.000 |
| | DH | mean | 1.779 | 0.846 | 2.672 | 0.993 | 23.295 | 0.999 | 1.214 | 1.000 | 0.003 | 0.051 | 0.005 | 0.054 | 0.056 | 0.183 |
| | | std | 0.407 | 0.118 | 0.646 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| | F | mean | 1.910 | 0.888 | 5.411 | 0.973 | 2.661 | 0.999 | 4.192 | 1.000 | 0.220 | 0.274 | 0.128 | 0.268 | 0.123 | 0.218 |
| | | std | 0.405 | 0.199 | 3.811 | 0.002 | 0.000 | 0.000 | 0.000 | 0.000 | 0.034 | 0.025 | 0.023 | 0.030 | 0.000 | 0.000 |
| | FH | mean | 0.411 | 0.993 | 0.150 | 0.988 | 2.379 | 0.999 | 0.568 | 1.000 | 0.273 | 1.000 | 0.279 | 1.000 | 0.158 | 1.000 |
| | | std | 0.149 | 0.000 | 0.017 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.051 | 0.000 | 0.082 | 0.000 | 0.000 | 0.000 |
| | WO | mean | 1.180 | 0.890 | 1.117 | 0.960 | 1.724 | 0.999 | 2.399 | 1.000 | 0.113 | 0.080 | 0.135 | 0.197 | 0.265 | 0.197 |
| | | std | 0.409 | 0.202 | 0.120 | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.014 | 0.002 | 0.007 | 0.000 | 0.000 | 0.000 |
| | WOH | mean | 1.351 | 0.869 | 0.534 | 0.980 | 2.691 | 0.999 | 2.389 | 1.000 | 0.037 | 0.536 | 0.019 | 0.536 | 0.081 | 0.539 |
| | | std | 0.230 | 0.131 | 0.085 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.006 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| | S | mean | 0.052 | 0.993 | 0.202 | 0.991 | 21.558 | 0.999 | 1.259 | 1.000 | 0.026 | 1.000 | 0.027 | 1.000 | 0.088 | 1.000 |
| | | std | 0.003 | 0.000 | 0.011 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |
| | SH | mean | 0.057 | 0.993 | 0.496 | 0.991 | 5.602 | 0.999 | 2.537 | 1.000 | 0.018 | 1.000 | 0.018 | 1.000 | 0.082 | 1.000 |
| | | std | 0.006 | 0.000 | 0.298 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.002 | 0.000 | 0.002 | 0.000 | 0.000 | 0.000 |

Table 4.4: ATE [m] and tracking rate (TR) obtained on the selected GRADE sequences (rows) grouped in their ground-truth and noisy versions. We report the mean and standard deviation over ten trials. The depth data is limited to 5 m.

| | | Training Fine-tuning | S-GRADE+S-CH | | A-GRADE | | S-GRADE | | S-GRADE+CH | | S-GRADE CH | | A-GRADE S-CH | | A-GRADE+CH | | A-GRADE CH | | A-GRADE+S-CH | | S-GRADE S-CH | | S-CH | | BASELINE | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| TUM RGB-D | DynaVINS — VO | halfsphere | mean | 0.074 | 0.968 | 0.062 | 0.968 | 0.051 | 0.968 | 0.063 | 0.968 | 0.064 | 0.968 | 0.100 | 0.967 | 0.060 | 0.968 | 0.069 | 0.968 | 0.064 | 0.968 | 0.072 | 0.967 | 0.148 | 0.966 | 0.097 | 0.967 |
| | | | std | 0.022 | 0.000 | 0.026 | 0.000 | 0.008 | 0.000 | 0.007 | 0.000 | 0.013 | 0.000 | 0.074 | 0.004 | 0.006 | 0.000 | 0.020 | 0.000 | 0.023 | 0.000 | 0.034 | 0.003 | 0.178 | 0.004 | 0.080 | 0.004 |
| | | rpy | mean | 0.166 | 0.975 | 0.155 | 0.975 | 0.230 | 0.975 | 0.129 | 0.974 | 0.169 | 0.967 | 0.108 | 0.975 | 0.142 | 0.972 | 0.115 | 0.972 | 0.143 | 0.975 | 0.131 | 0.975 | 0.136 | 0.975 | 0.130 | 0.972 |
| | | | std | 0.028 | 0.000 | 0.018 | 0.000 | 0.023 | 0.000 | 0.018 | 0.004 | 0.061 | 0.008 | 0.015 | 0.000 | 0.048 | 0.006 | 0.016 | 0.005 | 0.020 | 0.000 | 0.012 | 0.000 | 0.030 | 0.000 | 0.027 | 0.005 |
| | | static | mean | 0.357 | 0.981 | 0.135 | 0.972 | 0.279 | 0.978 | 0.050 | 0.970 | 0.141 | 0.972 | 0.091 | 0.972 | 0.123 | 0.855 | 0.071 | 0.970 | 0.212 | 0.980 | 0.325 | 0.976 | 0.291 | 0.919 | 0.542 | 0.968 |
| | | | std | 0.373 | 0.005 | 0.133 | 0.007 | 0.188 | 0.007 | 0.021 | 0.006 | 0.182 | 0.007 | 0.071 | 0.007 | 0.072 | 0.189 | 0.054 | 0.006 | 0.065 | 0.007 | 0.268 | 0.008 | 0.142 | 0.115 | 0.757 | 0.025 |
| | | xyz | mean | 0.049 | 0.970 | 0.039 | 0.970 | 0.050 | 0.970 | 0.044 | 0.970 | 0.045 | 0.970 | 0.041 | 0.970 | 0.043 | 0.970 | 0.046 | 0.970 | 0.047 | 0.970 | 0.044 | 0.970 | 0.040 | 0.849 | 0.055 | 0.970 |
| | | | std | 0.013 | 0.000 | 0.006 | 0.000 | 0.010 | 0.000 | 0.010 | 0.000 | 0.010 | 0.000 | 0.005 | 0.000 | 0.005 | 0.000 | 0.011 | 0.000 | 0.008 | 0.000 | 0.008 | 0.000 | 0.007 | 0.241 | 0.014 | 0.000 |
| | DynaSLAM | halfsphere | mean | 0.031 | 1.000 | 0.029 | 1.000 | 0.029 | 0.999 | 0.029 | 0.999 | 0.030 | 0.999 | 0.029 | 1.000 | 0.029 | 0.999 | 0.028 | 1.000 | 0.029 | 1.000 | 0.031 | 0.971 | 0.031 | 0.999 | 0.028 | 1.000 |
| | | | std | 0.002 | 0.000 | 0.002 | 0.000 | 0.001 | 0.003 | 0.002 | 0.002 | 0.001 | 0.003 | 0.001 | 0.001 | 0.001 | 0.003 | 0.001 | 0.000 | 0.001 | 0.001 | 0.002 | 0.049 | 0.001 | 0.002 | 0.002 | 0.001 |
| | | rpy | mean | 0.063 | 0.939 | 0.048 | 0.974 | 0.100 | 0.972 | 0.037 | 0.865 | 0.037 | 0.855 | 0.035 | 0.874 | 0.036 | 0.897 | 0.041 | 0.860 | 0.042 | 0.896 | 0.038 | 0.831 | 0.039 | 0.833 | 0.035 | 0.846 |
| | | | std | 0.047 | 0.029 | 0.015 | 0.023 | 0.021 | 0.003 | 0.004 | 0.039 | 0.004 | 0.021 | 0.003 | 0.019 | 0.006 | 0.034 | 0.011 | 0.019 | 0.005 | 0.042 | 0.003 | 0.025 | 0.005 | 0.043 | 0.006 | 0.022 |
| | | static | mean | 0.007 | 1.000 | 0.006 | 1.000 | 0.007 | 0.997 | 0.007 | 0.850 | 0.007 | 0.839 | 0.007 | 0.974 | 0.008 | 0.971 | 0.007 | 0.839 | 0.007 | 0.998 | 0.007 | 1.000 | 0.011 | 1.000 | 0.007 | 0.957 |
| | | | std | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.036 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.002 | 0.000 | 0.000 | 0.002 |
| | | xyz | mean | 0.016 | 0.979 | 0.016 | 0.956 | 0.016 | 0.980 | 0.016 | 0.915 | 0.016 | 0.915 | 0.016 | 0.922 | 0.016 | 0.915 | 0.015 | 0.915 | 0.016 | 0.915 | 0.016 | 1.000 | 0.016 | 0.979 | 0.015 | 0.915 |
| | | | std | 0.001 | 0.000 | 0.001 | 0.000 | 0.000 | 0.015 | 0.000 | 0.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.001 | 0.000 | 0.001 | 0.000 | 0.001 | 0.000 | 0.001 | 0.000 | 0.000 | 0.024 | 0.000 | 0.001 |
| GRADE | DynaVINS — VO | D | mean | 1.363 | 0.657 | 1.497 | 0.917 | 1.410 | 0.808 | 1.186 | 0.893 | 1.382 | 0.926 | 1.504 | 0.818 | 1.488 | 0.927 | 1.488 | 0.803 | 1.345 | 0.865 | 1.403 | 0.945 | 1.348 | 0.793 | 1.450 | 0.888 |
| | | | std | 0.331 | 0.277 | 0.315 | 0.085 | 0.259 | 0.220 | 0.448 | 0.097 | 0.273 | 0.090 | 0.280 | 0.226 | 0.391 | 0.089 | 0.250 | 0.303 | 0.323 | 0.232 | 0.260 | 0.075 | 0.346 | 0.297 | 0.441 | 0.096 |
| | | DH | mean | 1.621 | 0.712 | 1.368 | 0.691 | 1.501 | 0.770 | 2.330 | 0.416 | 1.348 | 0.590 | 1.412 | 0.685 | 1.380 | 0.659 | 2.109 | 0.577 | 2.296 | 0.676 | 1.949 | 0.598 | 1.345 | 0.640 | 1.582 | 0.644 |
| | | | std | 0.228 | 0.247 | 0.271 | 0.174 | 0.501 | 0.201 | 2.139 | 0.287 | 0.267 | 0.265 | 0.343 | 0.280 | 0.405 | 0.256 | 2.523 | 0.208 | 2.143 | 0.252 | 1.690 | 0.281 | 0.222 | 0.218 | 0.468 | 0.301 |
| | | F | mean | 1.490 | 0.746 | 1.701 | 0.820 | 1.628 | 0.755 | 1.743 | 0.741 | 1.688 | 0.721 | 1.571 | 0.946 | 1.723 | 0.819 | 1.511 | 0.743 | 1.794 | 0.807 | 1.747 | 0.825 | 1.604 | 0.880 | 1.532 | 0.841 |
| | | | std | 0.200 | 0.324 | 0.480 | 0.266 | 0.524 | 0.322 | 0.366 | 0.310 | 0.439 | 0.325 | 0.607 | 0.063 | 0.376 | 0.283 | 0.310 | 0.332 | 0.319 | 0.276 | 0.351 | 0.254 | 0.311 | 0.230 | 0.504 | 0.230 |
| | | FH | mean | 0.382 | 0.991 | 0.355 | 0.991 | 0.541 | 0.991 | 0.236 | 0.993 | 0.287 | 0.992 | 0.233 | 0.992 | 0.404 | 0.991 | 0.414 | 0.963 | 0.221 | 0.948 | 0.263 | 0.966 | 0.258 | 0.992 | 0.220 | 0.993 |
| | | | std | 0.259 | 0.003 | 0.209 | 0.003 | 0.374 | 0.003 | 0.073 | 0.000 | 0.218 | 0.002 | 0.154 | 0.002 | 0.295 | 0.003 | 0.214 | 0.081 | 0.057 | 0.091 | 0.107 | 0.079 | 0.219 | 0.002 | 0.058 | 0.000 |
| | | WO | mean | 1.130 | 0.814 | 1.186 | 0.659 | 1.344 | 0.839 | 1.334 | 0.864 | 1.199 | 0.755 | 1.184 | 0.766 | 1.331 | 0.855 | 1.279 | 0.790 | 1.195 | 0.857 | 1.469 | 0.772 | 1.473 | 0.887 | 1.219 | 0.910 |
| | | | std | 0.242 | 0.287 | 0.294 | 0.278 | 0.303 | 0.208 | 0.433 | 0.207 | 0.198 | 0.304 | 0.236 | 0.289 | 0.506 | 0.229 | 0.445 | 0.202 | 0.225 | 0.203 | 0.399 | 0.264 | 0.289 | 0.102 | 0.291 | 0.092 |
| | | WOH | mean | 1.389 | 0.847 | 1.186 | 0.878 | 1.324 | 0.904 | 1.492 | 0.874 | 1.935 | 0.937 | 1.710 | 0.928 | 1.575 | 0.807 | 1.283 | 0.838 | 1.195 | 0.867 | 1.458 | 0.862 | 1.529 | 0.897 | 1.474 | 0.808 |
| | | | std | 0.522 | 0.146 | 0.506 | 0.104 | 0.340 | 0.143 | 0.572 | 0.139 | 1.117 | 0.075 | 0.509 | 0.071 | 0.725 | 0.158 | 0.479 | 0.156 | 0.415 | 0.177 | 0.429 | 0.162 | 0.418 | 0.137 | 0.548 | 0.120 |
| | DynaSLAM | D | mean | 0.070 | 0.837 | 0.040 | 0.884 | 0.044 | 0.816 | 0.042 | 0.870 | 0.056 | 0.672 | 0.037 | 0.833 | 0.074 | 0.867 | 0.046 | 0.738 | 0.066 | 0.856 | 0.026 | 0.462 | 0.114 | 0.434 | 0.042 | 0.830 |
| | | | std | 0.049 | 0.085 | 0.004 | 0.037 | 0.008 | 0.058 | 0.007 | 0.026 | 0.044 | 0.098 | 0.006 | 0.100 | 0.077 | 0.083 | 0.011 | 0.054 | 0.071 | 0.110 | 0.006 | 0.064 | 0.105 | 0.113 | 0.009 | 0.111 |
| | | DH | mean | 0.011 | 0.091 | 0.013 | 0.097 | 0.012 | 0.094 | 0.007 | 0.086 | 0.012 | 0.096 | 0.008 | 0.090 | 0.016 | 0.099 | 0.015 | 0.090 | 0.015 | 0.099 | 0.020 | 0.079 | 0.014 | 0.079 | 0.011 | 0.097 |
| | | | std | 0.011 | 0.025 | 0.008 | 0.026 | 0.009 | 0.023 | 0.005 | 0.030 | 0.008 | 0.032 | 0.006 | 0.023 | 0.012 | 0.026 | 0.014 | 0.033 | 0.009 | 0.020 | 0.020 | 0.023 | 0.010 | 0.023 | 0.007 | 0.028 |
| | | F | mean | 0.789 | 0.523 | 0.676 | 0.408 | 0.592 | 0.426 | 0.587 | 0.371 | 0.843 | 0.593 | 0.581 | 0.450 | 0.692 | 0.447 | 0.851 | 0.590 | 0.803 | 0.400 | 0.858 | 0.484 | 0.486 | 0.260 | 0.858 | 0.440 |
| | | | std | 0.271 | 0.165 | 0.305 | 0.160 | 0.317 | 0.230 | 0.325 | 0.122 | 0.013 | 0.141 | 0.347 | 0.136 | 0.298 | 0.193 | 0.036 | 0.142 | 0.232 | 0.156 | 0.053 | 0.140 | 0.232 | 0.153 | 0.184 | 0.115 |
| | | FH | mean | 0.193 | 0.987 | 0.250 | 0.999 | 0.177 | 1.000 | 0.274 | 0.995 | 0.211 | 1.000 | 0.219 | 1.000 | 0.304 | 1.000 | 0.209 | 0.999 | 0.242 | 1.000 | 0.054 | 0.455 | 0.215 | 0.946 | 0.258 | 1.000 |
| | | | std | 0.063 | 0.025 | 0.061 | 0.001 | 0.039 | 0.000 | 0.066 | 0.008 | 0.034 | 0.000 | 0.029 | 0.000 | 0.228 | 0.000 | 0.030 | 0.002 | 0.067 | 0.001 | 0.023 | 0.266 | 0.071 | 0.085 | 0.054 | 0.000 |
| | | WO | mean | 0.071 | 0.079 | 0.090 | 0.079 | 0.087 | 0.079 | 0.083 | 0.079 | 0.080 | 0.079 | 0.087 | 0.079 | 0.091 | 0.079 | 0.083 | 0.079 | 0.086 | 0.079 | 0.069 | 0.079 | 0.087 | 0.097 | 0.090 | 0.079 |
| | | | std | 0.005 | 0.000 | 0.006 | 0.000 | 0.010 | 0.000 | 0.012 | 0.000 | 0.009 | 0.000 | 0.008 | 0.000 | 0.012 | 0.000 | 0.013 | 0.000 | 0.014 | 0.000 | 0.011 | 0.000 | 0.020 | 0.000 | 0.011 | 0.000 |
| | | WOH | mean | 0.014 | 0.538 | 0.012 | 0.538 | 0.011 | 0.535 | 0.013 | 0.538 | 0.014 | 0.537 | 0.013 | 0.526 | 0.012 | 0.538 | 0.014 | 0.538 | 0.012 | 0.538 | 0.043 | 0.104 | 0.017 | 0.520 | 0.013 | 0.538 |
| | | | std | 0.001 | 0.000 | 0.002 | 0.000 | 0.002 | 0.000 | 0.001 | 0.000 | 0.003 | 0.000 | 0.003 | 0.003 | 0.002 | 0.000 | 0.002 | 0.000 | 0.001 | 0.000 | 0.015 | 0.086 | 0.007 | 0.003 | 0.002 | 0.000 |

Table 4.5: ATE [m] and tracking rate (TR) obtained by evaluating DynaVINS and DynaSLAM while varying the model checkpoint used by the underlying YOLO and Mask R-CNN networks (columns). The evaluations are performed using both the TUM RGB-D (top half of the table) and selected GRADE sequences (bottom half of the table). GRADE's data has no additional noise, and its depth is limited to 3.5 m.

However, the original paper [131], does not report either standard deviation or number of trials, making this comparison difficult. Despite these variations, all other experiments in our study were conducted within the same system, ensuring consistency in our evaluations. Results on the TUM dataset indicate that changing the model for DynaVINS has no impact on TR. DynaSLAM instead is highly influenced by the segmentation quality, with many trials showing TR similar to or better than the baseline. Surprisingly, this happens even when using low-performing models. For example, with the model pre-trained on S-GRADE and fine-tuned on S-CH we can observe a tracking time 8.5% higher on the *xyz* sequence. For both SLAM frameworks, the ATE varies based on the model used. However, the best-performing networks are often not associated with the best ATE and TR couple. For example, using the model trained only with S-GRADE yields already compelling results. When using it with DynaVINS, we can obtain the best performance on *halfsphere* sequence, with half of the ATE with respect to to the baseline. The results obtained using DynaSLAM with Mask R-CNN trained on S-GRADE show ATEs comparable to the baseline results but with higher tracking times, thus with better overall performance. At the same time, DynaSLAM on the *rpy* sequence with the model pre-trained on A-GRADE and fine-tuned on CH, i.e. the best performing one according to the results shown in Table 3.7, significantly degrades the tracking rate of approximately 5%. Finally, when we evaluate the same SLAM methods using the different trained models on the different GRADE sequences, we find that it influences *both* the TR and ATE. However, as with the TUM RGB-D sequences, these results show no clear advantage in using the models with the highest performance, as, unintuitively, even the ones performing poorly in the detection and segmentation tasks can attain higher TR and ATE with respect to the other ones. Notably, we can see from these results that even while using the baseline network, both methods suffer from imperfect tracking rates, which can go as low as 84.6% for DynaSLAM on the *rpy* sequence.


Figure 4.1: DynaPix's motion probabilities on GRADE (left) and TUM RGB-D (right) frames. On the left side of each image, we colored the estimated *moving* regions. On the right side, ORB features are colored based on the motion probabilities, from static (green) to dynamic (red).

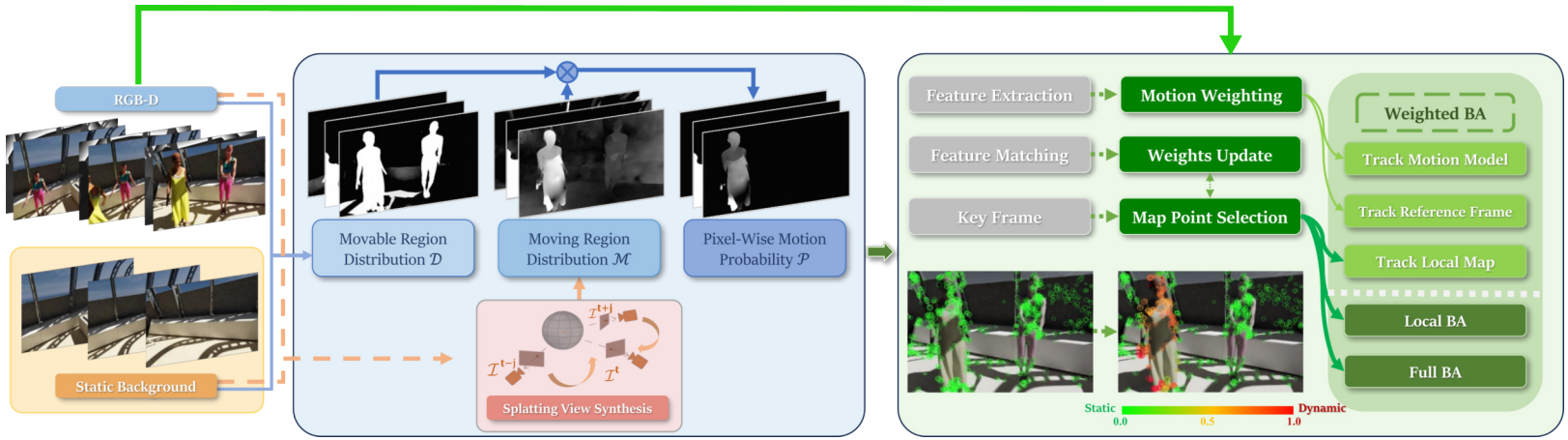



Figure 4.2: The DynaPix architecture consists of two main blocks, the *motion* probability estimation (blue box), and the modified ORB-SLAM2 backend (green box). We use RGB-D and corresponding background images to extract *movable* and *moving* regions on the current frame. The computed per-pixel *moving* probabilities (Section 4.4.1) are then integrated into the green-colored SLAM backend modules (Section 4.4.2).

## 4.4 DynaPix

Our thorough testing on several state-of-the-art Dynamic V-SLAM methods using synthetic sequences obtained with GRADE shows how most methods fail to track sequences that are out of distribution compared to common datasets. They also highlight the necessity of reporting the average sequence tracking rate to evaluate overall SLAM performance correctly, as the ATE alone may mislead evaluations - especially in dynamic environments. Our results show how state-of-the-art methods fail either in *correctly* (i.e. high ATE) or *completely* (i.e. low ATE but low tracking rate) estimating the trajectories, despite their good performance when tested on common datasets. Moreover, our evaluations performed on TUM RGB-D and synthetic sequences using our trained detection and segmentation models exemplify the demand for thorough evaluations and studies in Dynamic V-SLAM. Indeed, using the best-performing trained networks does not always yield the best result, suggesting that more reliable feature rejection procedures and robust methods are needed. Notably, the enhanced realism and flexibility of GRADE have enabled rigorous and diverse testing of SLAM approaches in simulation. By closely mirroring real-world conditions, our framework allows for widespread and stress testing of state-of-the-art methods, exposing them to a broader range of scenarios and edge cases. As a result, even without explicitly introducing dedicated sim-to-real adaptation techniques, the improved evaluation process can inherently facilitate the smooth and effective transfer of these methods to real-world conditions by allowing researchers to evaluate adaptability and robustness beforehand.

As we just saw, removing dynamic observations is a de facto standard in Dynamic V-SLAM, where many frameworks use a limited number of keypoints associated with specific image features. Then, while the at-unison movement of pixels can be captured as part of a supposedly dynamic object, e.g. with segmentation models, the information is still carried by the single feature located on a precise pixel of the image. Moreover, not always the entirety of an object is moving simultaneously, e.g. when a person walks or waves a hand. Thus, rather than grouping all pixels belonging to the same object equally, and inspired by optical flow-based methods, we approach the problem from a pixel-wise perspective. Furthermore, we note that directly removing the features from the system using a binary static/moving classification without considering their current motion states may produce a significant loss of valuable and usable information.

Therefore, our goal is to estimate a movement probability associated with each pixel and dynamically integrate this notion into the SLAM backend. This allows us to selectively weigh singular feature points rather than binary classify whole groups of pixels equally. With this estimator, we manage to overcome the general shortcomings of classic semantic-based detectors, such as imprecise detection/segmentation or limitation to predefined categories, while successfully reducing the estimated errors due to the direct usage of optical flow methods. As discussed in Sections 4.1 and 4.2, learning-based flow estimation networks [56, 215] are commonly used to identify moving regions. However, their deployments in indoor dynamic environments pose specific challenges, including

increased false correspondences in textureless areas (e.g. empty walls, floors), noisy estimation due to incomplete elimination of camera motion, and misclassification.

Based on the aforementioned insights, our robust visual SLAM system for indoor dynamic environments consists of two novel modules: a pixel-wise motion probability estimator and an enhanced pose optimization process. The architecture of the proposed DynaPix SLAM system can be seen in Figure 4.2. The system inputs are RGB-D and corresponding static background images, which show the same scene in its version without any dynamic entity. Those can be generated either in simulation through synthetic generation or, for real-world sequences, by estimated background images, e.g. via video inpainting, view synthesis, or background filling techniques [17, 126, 147]. Then, the motion estimator generates a **probabilistic representation** which identifies moving elements within the current image frame (Section 4.4.1). This module enables the detection of **specific moving parts** within objects, as well as shadow and reflections, overcoming the limitations of traditional semantic or rigid motion segmentation methods. Our proposed probabilistic motion estimation is composed of two submodules: *movable region* and *moving region* estimation. Movable regions are computed through background differencing and generate distributions covering potential moving objects with the corresponding shadows/reflections. Moving regions are then estimated through a rectified flow differencing mechanism, which uses a novel combination of splatting view synthesis and static background with dynamic flow subtraction. The two are then fused to obtain the final pixel-wise motion probability. Then, we incorporate these motion probabilities into the SLAM back-end through a **dynamic weighting** of the keypoints (Section 4.4.2). This refined optimization module ensures the preservation of more keypoints during the tracking phase, thereby enhancing the system's robustness with extended tracking duration.

### 4.4.1 Pixel-wise Motion Probability

To eliminate the influence of dynamic objects, we estimate what is moving within the current scene. We introduce a two-staged method to identify i) potentially dynamic (*movable*) and ii) currently *moving* regions. The *movable* regions are the area of the image where motion **can** occur. This serves as prior confidence to refine *moving* region estimation into pixel-wise motion probabilities.

#### Preliminary: Splatting-based View Synthesis

Homography transformation is commonly used to reproject images from other viewpoints to the current location [216]. However, this method assumes that all observed points lie on the same plane, regardless of their depth information. This results in mismatches between the current frame $\mathcal{I}^t$ and the reprojection of the previous frame in the current viewpoint, $\widetilde{\mathcal{I}}^{t-1}$, as shown in Figure 4.3. To address this, we follow the idea of

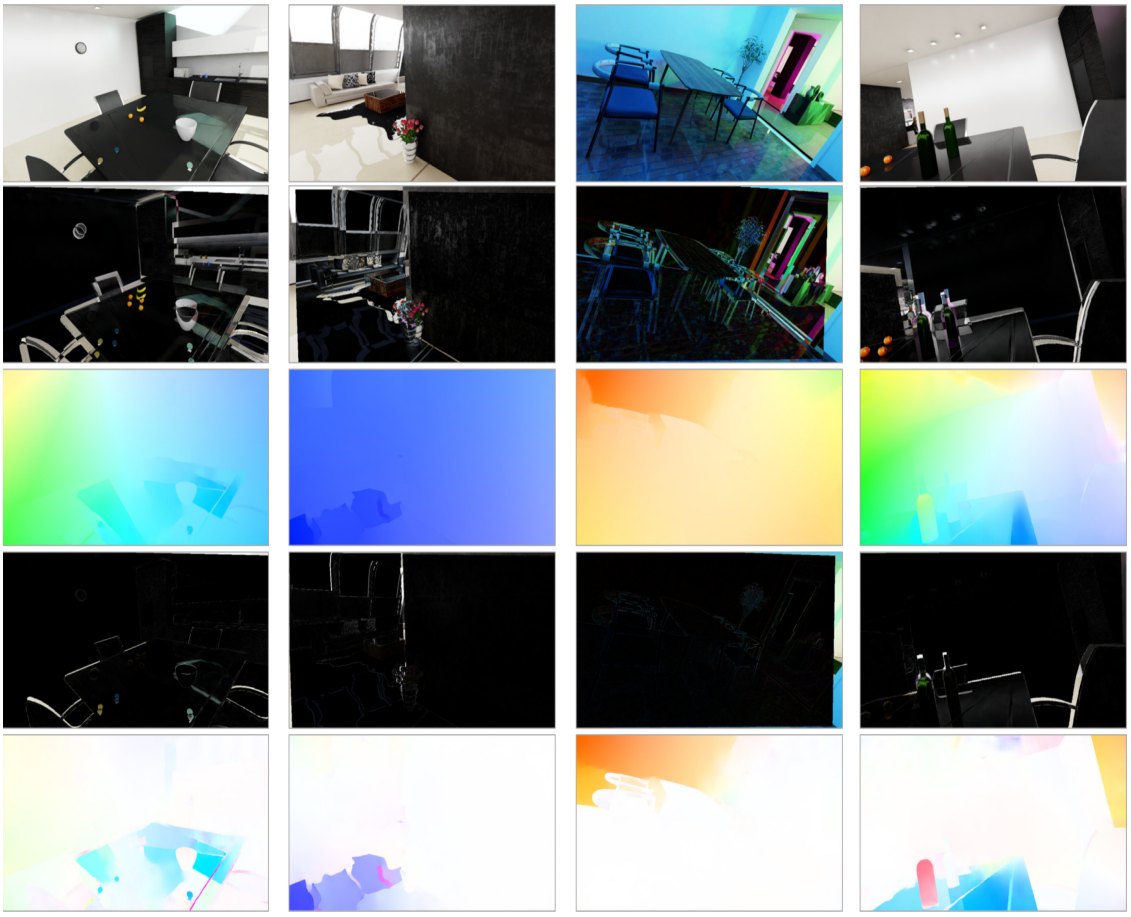


Figure 4.3: Projecting frames through **Soft Splatting** (fourth line, our method) and **Homography Transformation** (second line, standard approach) techniques using frames for static scenes. The second and fourth lines show the difference between frames $\widetilde{I}^{t+1}$ and $I^t$ to illustrate the alignment performance of these two methods. The third and fifth lines demonstrate their effects on flow estimation. These images also highlight the errors of the optical-flow estimation approach in featureless areas.

*softmax splatting* [154] for more accurate view synthesis. Given the camera intrinsic matrix and the depth map of the previous frame, we can project the pixels of $\mathcal{I}^{t-1}$ into the 3D space to recover their 3D information, as introduced in Section 1.3.2. Then, we can reproject these 3D points to the current viewpoint using an (estimated) transformation between the two poses. With these correspondences, each pixel of $\widetilde{\mathcal{I}}^{t-1}$ can be finally synthesized by participation from adjacent ones:

$$\widetilde{\mathcal{I}}^{t-1}(u',v') = \frac{\sum(\exp(z)\cdot\mathcal{I}^{t-1}(u,v))}{\sum(\exp(z))} \tag{4.1}$$

where $\sum(\cdot)$ denotes the sum of all contributing pixels from the original frame $\mathcal{I}^{t-1}$, $z$ is

the pixel's depth, and $\exp(z)$ serves as the weighting factor. More details can be found in [154]. In Figure 4.3, we show the advantage of splatted views over homography transforms.

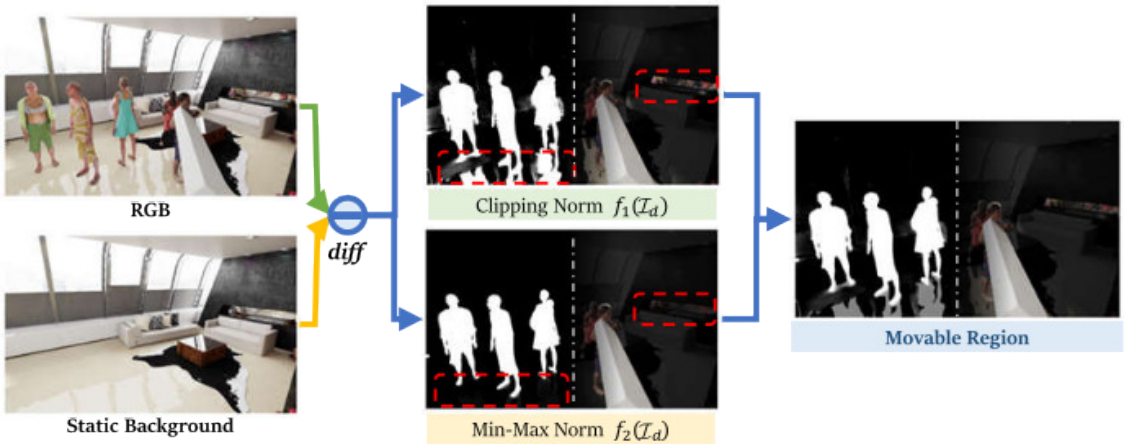



Figure 4.4: Flow diagram of the *movable* regions estimation.

### Movable Region Estimation

Although semantic cues are effective in identifying movable regions, they are incapable of capturing objects beyond predefined categories and imperceptible variations induced by dynamic objects, such as occlusions, reflections, and shadows. Thus, we use the static background image, $\mathcal{I}_{BG}$, in our system as prior information alongside the current image, $\mathcal{I}$. The difference between the current dynamic and static images in the RGB domain is captured by

$$\mathcal{I}_d = |\mathcal{I} - \mathcal{I}_{BG}| \tag{4.2}$$

Given the image difference $\mathcal{I}_d$, the per-pixel *movable* probability $\mathcal{D}$ is then computed through:

$$\mathcal{D} = \zeta \cdot f_1(\mathcal{I}_d) + (1-\zeta) \cdot f_2(\mathcal{I}_d), \tag{4.3}$$

where $f_1$ and $f_2$ indicate respectively clipping and min-max normalizations to scale $\mathcal{I}_d$ into $[0,1]$. The factor $\zeta$, given as

$$\zeta = \frac{1}{2} + \frac{1}{\exp(0.04 \cdot \max(\mathcal{I}_d)) + 1} \in [0.5, 1], \tag{4.4}$$

is used to weight these two terms. This is necessary because the direct application of $f_2$ (min-max normalization) on $\mathcal{I}_d$ could cause exploding scaling issues. Such problems typically occur when the current observation $\mathcal{I}$ closely resembles its static background $\mathcal{I}_{BG}$, potentially due to the absence of dynamic objects coupled with minor color discrepancies. This movable estimation process can provide reliable information on all potential

dynamic objects. Naturally, depending on how the background image is generated and what is considered to be movable or not, this step will have different effects as $\mathcal{I}_{BG}$ will vary accordingly. The workflow of this procedure is presented in Figure 4.4, while some qualitative results can be seen in Figure 4.5.

Figure 4.5: Step-by-step visualization of the movable region estimation on multiple synthetic frames, covering shadows, reflections, and movable objects. On the first two rows, we can see the dynamic and background frames. In the third row, we show the result of the clipping operation, while in the fourth row, we report the result of the min-max operation. Finally, on the fifth row, we show the overall result.

### Moving Region Estimation

Having identified the movable regions, we proceed to estimate the currently moving parts rather than segmenting entire objects. The moving region can be decomposed into several sets of 3D points, each characterized by continuous displacements in Euclidean space. These displacements can be further projected onto the current 2D image frame for observations, resulting in pixel-wise motion across successive frames. Figure 4.6 illustrates the pipeline of our moving region estimation, while Figure 4.7 some qualitative results. We first reproject adjacent frames into the current view to eliminate camera motion. We apply FlowFormer [93] to calculate optical flow, thereby deriving pixel-wise displacements

to identify moving regions. Ideally, when comparing the current frame with the reprojected frames, static pixels should remain at the same coordinates while moving pixels display noticeable displacements. However, this approach can be compromised by the incomplete elimination of camera motion or false pixel correspondences in texture-less areas. Therefore, we adopt *splatting-based view synthesis* for accurate projection, and *static/dynamic flow differencing* for robust moving region estimation.

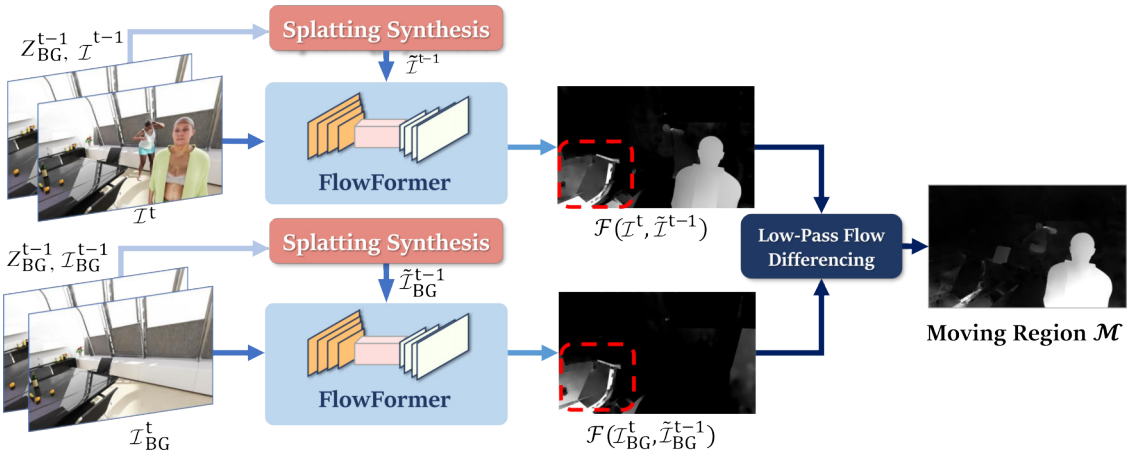



Figure 4.6: Flow diagram of the *moving* region estimation between $\mathcal{I}^t$ and $\mathcal{I}^{t-1}$.

After obtaining the reprojected view, $\widetilde{\mathcal{I}}^{t-1}$, the estimation of moving regions is formulated as the flow estimation between $\mathcal{I}^t$ and $\widetilde{\mathcal{I}}^{t-1}$, indicated with $\mathcal{F}(\mathcal{I}^t, \widetilde{\mathcal{I}}^{t-1})$. The displacement in pixel-wise correspondences is quantified by the flow magnitude. However, this may not fully eliminate either the camera motion effects or the incorrect pixel correspondences that frequently occur in texture-less and ambiguous texture-rich areas. Therefore, we also perform flow estimation on static background images, using the procedure previously described, to further compensate for possible errors in these regions through:

$$\mathcal{F}^*(\mathcal{I}^t, \widetilde{\mathcal{I}}^{t-1}) = \min(\mathcal{F}(\mathcal{I}^t, \widetilde{\mathcal{I}}^{t-1}), \mathcal{F}(\mathcal{I}^t, \widetilde{\mathcal{I}}^{t-1}) - \mathcal{F}(\mathcal{I}^t_{BG}, \widetilde{\mathcal{I}}^{t-1}_{BG})) \tag{4.5}$$

A low-pass filter is then used to moderate the flow magnitude increase, due to subtraction in noisy estimations, and a min-max normalization operation is applied to project these values into $[0,1]$. Assuming the scene dynamics are consistent over a short interval, the motion attribute of each pixel should remain similar or vary minimally in neighboring frames. Without loss of generality, this can be extended to other frames closely spaced in time. Therefore, we finalize the *moving region* estimation across multiple frames as:

$$\mathcal{M} = \frac{1}{2n} \sum_{j \in J} \left( \mathcal{F}^*(\mathcal{I}^t, \widetilde{\mathcal{I}}^{t+j}) + \mathcal{F}^*(\mathcal{I}^t, \widetilde{\mathcal{I}}^{t-j}) \right) \tag{4.6}$$

where $J$ is a set of time offsets with $n$ as its cardinality. In our implementation, we adopt

$J = [2]$ (i.e. $n = 1$). As each index of $\mathcal{M}$ corresponds to the pixels of $\mathcal{I}^t$ and their values are between 0 and 1, $\mathcal{M}$ is also bounded between 0 and 1. A flow magnitude different from zero, in principle, indicates that a pixel has effectively moved between two frames. However, noise and incorrect matching may also result in such outcomes. Moreover, in general, the bigger the movement, the higher the magnitude and the detrimental effect on the SLAM backend. Thus, we chose to treat this quantity as our per-pixel *moving* probability factor of frame $\mathcal{I}^t$.

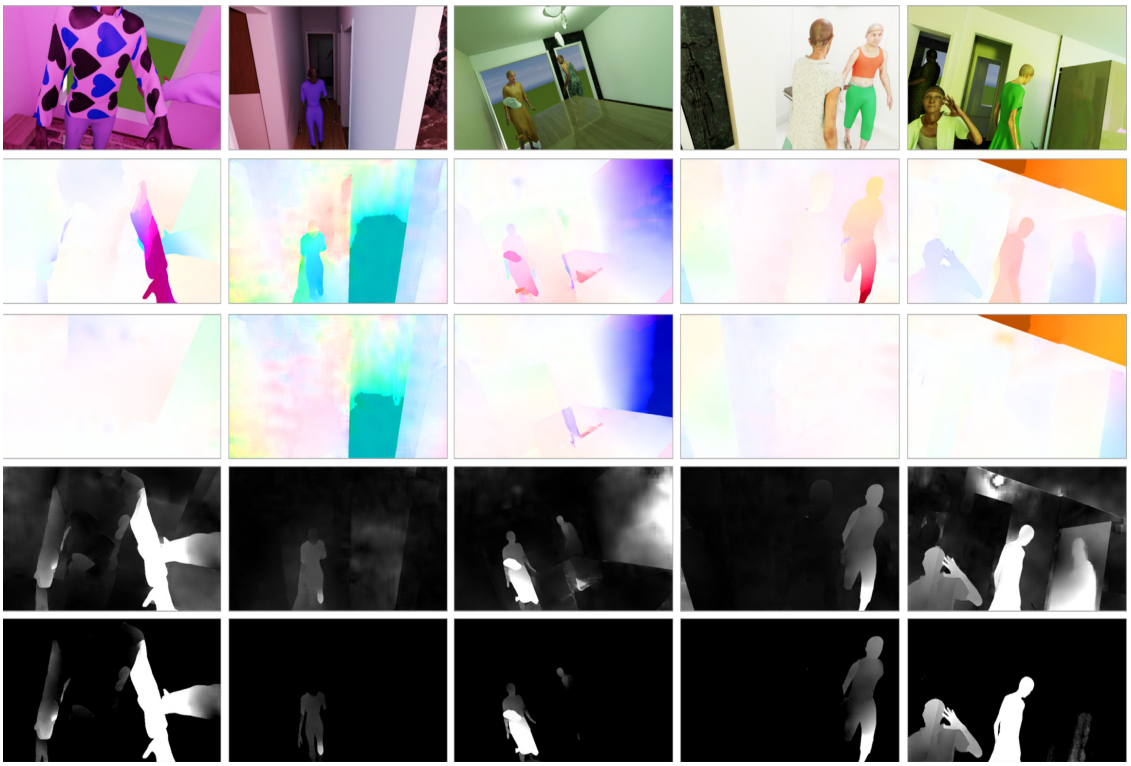


Figure 4.7: Step-by-step visualization of the moving region estimation and final motion probability for five different synthetic frames. While we show in the first row the original RGB frame, in the second and third ones we show the flow estimation on dynamic scenes $\mathcal{F}(I_d^t, \widetilde{I}_d^{t+i})$ and static scenes $\mathcal{F}(I_s^t, \widetilde{I}_s^{t+i})$ respectively. Notice the similar estimation errors in common static regions. We report the result of the moving region estimation in the fourth row. The fifth row shows the final motion probability for the frames, obtained after combining them with the corresponding movable region estimation results (not shown).

### Motion Probability

The final *motion* probability $\mathcal{P}$ for the current frame is defined by integrating the per-pixel moving probabilities $\mathcal{M}$ (Equation (4.6)), with the movable ones $\mathcal{D}$ (Equa-

tion (4.3)), through pixel-wise multiplication:

$$\mathcal{P} = \mathcal{D} \odot \mathcal{M} \tag{4.7}$$

This multi-step processing allows us to effectively reduce noise resulting from hallucinations (e.g. in the background inpainting) or wrong estimations (e.g. in the optical flow computation) by weighing the two factors, as shown by our ablation studies. An example of such probabilities is depicted in Figures 4.1 and 4.7.

### 4.4.2 Tracking and Pose Optimization

We incorporate estimated motion probabilities into the front-end *tracking* and back-end *pose optimization* process within the ORB-SLAM2 framework [152]. Different from prior works [17,97,255], our method seeks to use **all stationary objects** (or their **stationary parts**) to improve localization performance, while actively preventing their negative effects once they resume their motion.

The tracking module consists of the first-stage coarse estimation (i.e. *track motion model*, *track reference frame*) followed by a more precise second-stage estimation as *track local map*, while the backend module contains *local BA* and *global BA* for the optimization of camera poses and map point locations. Based on this, we improve the map point selection process and weighted bundle adjustment of ORB-SLAM2. The latter directly affects both the tracking and backend optimization modules.

#### Local Tracking

To prevent tracking failures, we seek to preserve more keypoints, even those associated with *stationary movable* or *slightly moving* objects, particularly in scenarios where movable objects dominate the frame. Another concern is that the temporarily stationary keypoints may start to move in future frames, disrupting the pose estimation process. Therefore, we first perform a coarse pose estimation (i.e. *motion-only BA*) using **all keypoints and their respective weights** during the tracking thread. Then, thresholds $\upsilon_{add}$ and $\upsilon_{del}$ are set to ensure that only higher-confidence and static keypoints are selected as map points for fine pose optimization (i.e. during *full* or *local BA*). Then, given a frame $\mathcal{I}^t$ and its estimated per-pixel motion probability $\mathcal{P}$, all keypoints $\mathcal{K}^t = \{k_1^t, \ldots, k_n^t\}$ are assigned with motion probabilities $\mathcal{P}(k_i^t)$ based on their coordinates. Upon identifying a current frame $\mathcal{I}^t$ as a keyframe, its keypoints $\mathcal{K}^t$ are considered *candidate* map points. Among these, keypoints with $\mathcal{P}(k_i^t) \leq \upsilon_{add}$ are promoted to map points $\mathcal{Y} = \{y_j | y_j \in \mathbb{R}^3, \mathcal{P}(y_j^t) = \mathcal{P}(k_i^t)\}$. The motion probability for each map point is continually updated by matched keypoints from new frames. Any map points exhibiting motion, indicated by $\mathcal{P}(y_j^t) \geq \upsilon_{del}$, will be eliminated from the map. In our implementation, we set $\upsilon_{add} = 0.05$ and $\upsilon_{del} = 0.1$.

### Weighted Bundle Adjustment

Inspired by [83, 90, 267], the respective dynamic weights are integrated into the BA optimization process (Equation (1.14)). The reprojection cost is formulated as:

$$C = \arg\min_{\mathbf{R},\mathbf{t}} \frac{1}{2} \sum_{i=1}^{n} w_i ||\mathbf{x}_i - \pi(\mathbf{R}\mathbf{X}_i + \mathbf{t})||_\rho \tag{4.8}$$

where $\|\cdot\|_\rho$ denotes the robust Huber cost function, $\mathbf{R}$ and $\mathbf{t}$ represent the camera's orientation and translation, respectively. $\mathbf{X}_i \in \mathbb{R}^3$ indicates a 3D point in the scene, with $\mathbf{x}_i$ as its matched 2D keypoint coordinates. The function $\pi(\cdot)$ is used for camera projection. Error term weights are determined by $w_i = 1 - \mathcal{P}(k_i^t)$, where a higher $w_i$ indicates a greater probability of the point being static. The coarse estimation applies this solely to refine camera poses with weights between $[0, 1]$. For fine optimization, we simultaneously refine camera poses and map point locations using the same cost function described in Equation (4.8), with weights constrained to [1-$\upsilon_{del}$, 1]. Both optimizations are solved using the Levenberg-Marquardt method within the g2o framework. Consequently, different from previous approaches that just discarded features based on a binary classification, we retain more keypoints that can now provide a *weighted* contribution to the pose optimization without being indiscriminately discarded. Notably, as seen above, if the keypoint or the feature *becomes* dynamic, it is completely removed from the system.

## 4.5 Experiments

We evaluate our system using sequences from both the TUM RGB-D [211] and the GRADE datasets. TUM RGB-D includes multiple indoor dynamic sequences, often adopted to benchmark visual SLAM approaches. Specifically, we use the four *fr3/walking* sequences {*halfsphere, static, rpy, xyz*}, with respective durations of 35.81, 24.83, 30.61, and 28.83 seconds. From GRADE, we adopt the same 60-second-long D, DH, WO, WOH, S, SH, F, FH experiments used in the previous section (3.5 m depth, no additional noise). They represent long-term synthetic sequences for dynamic environments that have proven challenging for many Dynamic SLAM methods. Recall that S[H] represents experiments recorded in static environments, D[H] has additional moving people, F[H] features randomly flying objects, and WO[H] involves occlusions of the camera sensor. The H, where present, indicates that the camera is kept horizontal throughout the entire sequence. All experiments are rendered both with and without the dynamic objects, i.e. the camera is positioned in the same location, and the scene is rendered twice using the experiment-repetition tool.

We use the E2FGVI [126] video inpainting method to obtain static background images. To avoid failures on long-term sequences due to excessive GPU memory usage, we adapt our input frame strategy to a custom 50-frame sliding window approach and a 100-frame bootstrap with reference frames. Despite the original framework show-

ing remarkable performance on static/slightly moving camera views, it hardly adapts to varying environments with long-term moving cameras. Indeed, the method makes use of *neighboring frames* and *reference frames* (extracted based on specific time intervals) to inpaint the current image. Therefore, to further provide sufficient background information in long-term sequences, we modify the selection strategy of the reference frames to remove irrelevant frames that are observing irrelevant areas. Using the ground-truth camera poses, we select the frames with the closest viewpoint as reference frames. In this way, these new reference frames can largely cover the possible background regions to improve the inpainting process. Note that E2FGVI requires a mask input to select the inpainting region, and we set it such that it covers the people in the frame. Examples of inpainted frames are illustrated in Figure 4.8. Moreover, inpainting synthetic sequences is challenging due to the domain gap. Therefore, for those, we employ the available static sequences as static background images, i.e. without flying objects and people. Notably, the blurriness resulting from the hallucinations due to the inpainting process and varying lighting conditions in inpainted frames. This also introduces noise into our method and potentially reduces the number of available features. Due to the low tracking rate and high ATE of the original ORB-SLAM2 and DynaSLAM methods experienced in many GRADE sequences, we rely on the available camera poses for both the inpainting and splatting view synthesis. For consistency, we do so for both synthetic and real-world experiments. It is worth noting that neighboring *frame-by-frame* pose variations can often be estimated or optimized through various visual (inertial) odometry or deep learning methods with high accuracy. We relax this constraint in our ablation studies for real-world sequences.

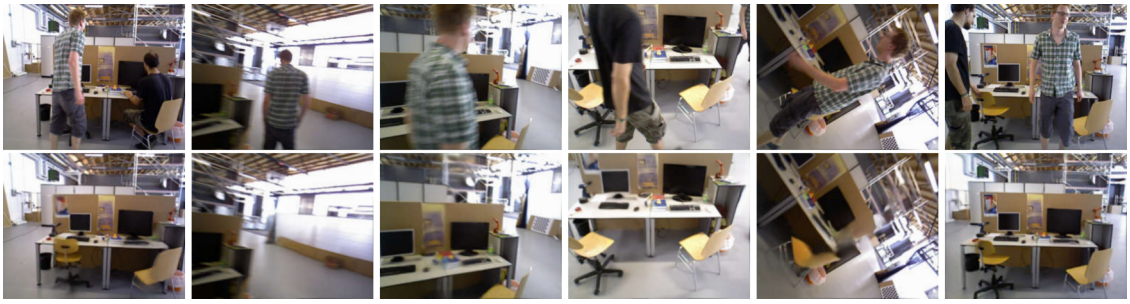


Figure 4.8: Examples of inpainted frames of the TUM RGB-D sequences.

We benchmark DynaPix, including all components detailed in Section 4.4, and DynaPix+, which extends DynaPix based on DynaSLAM [17]. DynaPix+ integrates semantic cues solely into *map point selection*, allowing keypoints associated with dynamic classes to participate in tracking but not to be promoted as map points for subsequent fine optimization. We compare against their unmodified underlying methods, ORB-SLAM2 and DynaSLAM. Those are the main baseline methods since they utilize the same back-end system, i.e. ORB-SLAM2. For completeness, we also evaluate DynamicVINS [131] (indicated with DynaVINS) and WF-SLAM [267] (for real-world sequences), although

the use of different backends inherently makes those comparisons less informative. We perform ablation studies (Section 4.5.3) on dynamic sequences to clarify the effect of each component with both DynaPix and DynaPix+. We use the ATE and the tracking rate (TR) to measure accuracy and robustness. The TR represents the ratio of tracked time over the entire sequence duration, a metric often overlooked in many studies [26]. Although it is not a precise estimation of a future ATE forecast, we also combine ATE and TR in a mixed metric, which we name **ATR**. This is obtained by computing the ratio $ATE/TR$. Clearly, the ATR will be negatively impacted by low TR and positively affected by low ATE, serving as an aggregating measure to compare the results. Our results on the GRADE dataset are reported in Tables 4.6 and 4.8, while the ones on the TUM RGB-D sequences are detailed in Tables 4.7 and 4.9. We repeat each experiment ten times using the default settings and report the mean and standard deviation results.

### 4.5.1 Static Sequences

We use the same static sequences used in the previous section to demonstrate that DynaPix and DynaPix+ do not degrade the performance of their underlying SLAM frameworks in static scenes. Generally, as can be seen in Table 4.6 (where we omit the '-static' for simplicity), both DynaPix and DynaPix+ perform well in such scenarios, often outperforming the base SLAM methods in most experiments. This is except for the WO sequence, in which the TR of DynaPix is nearly half that of ORB-SLAM2. However, as indicated by the standard deviation, the significant variation in both ATE and TR for this sequence is due to a set of challenging featureless frames around the 26-second mark. This highlights the importance of using TR alongside ATE for comprehensive analysis of estimation results. Specifically, in the WO experiment, the ATE remains similar in both methods, although the TR is significantly different. Focusing on the DynaPix+ and DynaSLAM results, our approach achieves better TR and ATE results overall, with $\sim 26\%$ lower average ATE and a $\sim 1\%$ increase of TR. These results also show how having a semantic segmentation system that blindly masks features out of the SLAM backend can adversely affect performance as a result of noise or incorrect detection, as DynaSLAM exhibits the lowest overall TR despite the static nature of these experiments. Finally, we notice how DynaVINS obtains a better tracking rate in the DH and WOH sequences but with at least eight times higher average ATE.

### 4.5.2 Dynamic Sequences

As shown in Tables 4.6 and 4.7, DynaPix and DynaPix+ consistently outperform their base methods by considerable amounts across both synthetic and real-world sequences. The TR of WO with DynaPix+, for example, is 2.5 times better than DynaSLAM, with a lower ATE as well. For the same sequence, DynaPix shows a remarkable 86% improvement in ATE over ORB-SLAM2, maintaining the *same* TR.

| | | GRADE – Static | | | | | | | | | | GRADE – Dynamic | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | DynaPix | | ORB-SLAM2 | | DynaPix+ | | DynaSLAM | | DynaVINS | | DynaPix | | ORB-SLAM2 | | DynaPix+ | | DynaSLAM | | DynaVINS | |
| | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| FH | mean | 0.006 | 1.00 | 0.010 | 1.00 | 0.007 | 1.00 | 0.012 | 1.00 | 0.020 | 0.99 | 0.035 | 1.00 | 0.248 | 1.00 | 0.053 | 1.00 | 0.232 | 0.98 | 0.220 | 0.99 |
| | std | 0.000 | 0.00 | 0.002 | 0.00 | 0.000 | 0.00 | 0.005 | 0.00 | 0.002 | 0.00 | 0.016 | 0.00 | 0.103 | 0.00 | 0.029 | 0.00 | 0.041 | 0.02 | 0.058 | 0.00 |
| F | mean | 0.230 | 0.86 | 0.330 | 0.90 | 0.323 | 0.88 | 0.529 | 0.82 | 1.176 | 0.83 | 0.291 | 0.20 | 0.359 | 0.35 | 0.579 | 0.64 | 0.864 | 0.42 | 1.532 | 0.84 |
| | std | 0.426 | 0.00 | 0.507 | 0.01 | 0.505 | 0.01 | 0.526 | 0.04 | 0.404 | 0.26 | 0.108 | 0.01 | 0.156 | 0.03 | 0.465 | 0.23 | 0.217 | 0.07 | 0.504 | 0.23 |
| DH | mean | 0.005 | 0.18 | 0.005 | 0.18 | 0.006 | 0.18 | 0.013 | 0.07 | 1.548 | 0.73 | 0.006 | 0.18 | 0.005 | 0.18 | 0.005 | 0.18 | 0.011 | 0.10 | 1.582 | 0.64 |
| | std | 0.001 | 0.00 | 0.001 | 0.01 | 0.005 | 0.00 | 0.009 | 0.03 | 0.192 | 0.22 | 0.006 | 0.00 | 0.001 | 0.01 | 0.001 | 0.00 | 0.003 | 0.02 | 0.468 | 0.30 |
| D | mean | 0.023 | 0.98 | 0.018 | 0.97 | 0.019 | 0.98 | 0.024 | 0.98 | 0.366 | 0.99 | 0.032 | 0.98 | 0.317 | 0.99 | 0.062 | 0.98 | 0.050 | 0.89 | 1.450 | 0.89 |
| | std | 0.006 | 0.01 | 0.002 | 0.03 | 0.005 | 0.00 | 0.008 | 0.02 | 0.330 | 0.00 | 0.010 | 0.02 | 0.043 | 0.01 | 0.043 | 0.03 | 0.009 | 0.05 | 0.441 | 0.10 |
| WOH | mean | 0.012 | 0.54 | 0.013 | 0.54 | 0.008 | 0.54 | 0.015 | 0.54 | 1.473 | 0.98 | 0.009 | 0.54 | 0.016 | 0.54 | 0.012 | 0.54 | 0.012 | 0.54 | 1.474 | 0.81 |
| | std | 0.008 | 0.00 | 0.010 | 0.00 | 0.004 | 0.00 | 0.017 | 0.00 | 0.264 | 0.01 | 0.001 | 0.00 | 0.008 | 0.00 | 0.002 | 0.00 | 0.002 | 0.00 | 0.548 | 0.12 |
| WO | mean | 0.040 | 0.44 | 0.038 | 0.83 | 0.101 | 0.73 | 0.043 | 0.78 | 0.806 | 0.99 | 0.023 | 0.20 | 0.168 | 0.20 | 0.044 | 0.20 | 0.083 | 0.08 | 1.219 | 0.91 |
| | std | 0.023 | 0.35 | 0.021 | 0.32 | 0.202 | 0.34 | 0.022 | 0.28 | 0.442 | 0.00 | 0.002 | 0.00 | 0.022 | 0.00 | 0.006 | 0.00 | 0.010 | 0.00 | 0.291 | 0.09 |
| SH | mean | 0.010 | 1.00 | 0.012 | 1.00 | 0.009 | 1.00 | 0.010 | 1.00 | 0.029 | 0.99 | — | — | — | — | — | — | — | — | — | — |
| | std | 0.001 | 0.00 | 0.002 | 0.00 | 0.001 | 0.00 | 0.001 | 0.00 | 0.005 | 0.00 | — | — | — | — | — | — | — | — | — | — |
| S | mean | 0.010 | 1.00 | 0.011 | 1.00 | 0.011 | 1.00 | 0.010 | 1.00 | 0.036 | 0.99 | — | — | — | — | — | — | — | — | — | — |
| | std | 0.001 | 0.00 | 0.001 | 0.00 | 0.002 | 0.00 | 0.002 | 0.00 | 0.003 | 0.00 | — | — | — | — | — | — | — | — | — | — |
| Average | | 0.042 | 0.75 | 0.055 | 0.80 | 0.060 | 0.79 | 0.082 | 0.78 | 0.682 | 0.94 | 0.066 | 0.52 | 0.185 | 0.54 | 0.126 | 0.59 | 0.209 | 0.50 | 1.246 | 0.85 |

Table 4.6: ATE [m] and Tracking Rate (TR) on both **static** and **dynamic** GRADE sequences. In bold the best ATR.

| | | DynaPix | | ORB-SLAM2 | | DynaPix+ | | DynaSLAM | | WF-SLAM | | DynaVINS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| w_half | mean | 0.030 | 1.00 | 0.607 | 0.79 | 0.027 | 1.00 | 0.029 | 1.00 | 0.030 | 0.50 | 0.097 | 0.97 |
| | std | 0.002 | 0.00 | 0.180 | 0.11 | 0.002 | 0.00 | 0.001 | 0.00 | 0.004 | 0.35 | 0.080 | 0.00 |
| w_rpy | mean | 0.043 | 1.00 | 0.744 | 0.99 | 0.038 | 0.99 | 0.040 | 0.86 | 0.031 | 0.59 | 0.130 | 0.97 |
| | std | 0.007 | 0.00 | 0.115 | 0.01 | 0.007 | 0.02 | 0.008 | 0.03 | 0.007 | 0.07 | 0.027 | 0.01 |
| w_static | mean | 0.012 | 1.00 | 0.355 | 1.00 | 0.008 | 1.00 | 0.007 | 0.98 | 0.007 | 0.97 | 0.542 | 0.97 |
| | std | 0.002 | 0.00 | 0.121 | 0.00 | 0.001 | 0.00 | 0.000 | 0.00 | 0.000 | 0.02 | 0.757 | 0.03 |
| w_xyz | mean | 0.018 | 1.00 | 0.732 | 0.84 | 0.015 | 1.00 | 0.016 | 0.92 | 0.017 | 0.99 | 0.055 | 0.97 |
| | std | 0.002 | 0.00 | 0.102 | 0.11 | 0.000 | 0.00 | 0.001 | 0.00 | 0.001 | 0.02 | 0.014 | 0.00 |
| Average | | 0.026 | 1.00 | 0.610 | 0.91 | 0.022 | 1.00 | 0.023 | 0.94 | 0.021 | 0.76 | 0.206 | 0.97 |
| ATR | | 0.026 | | 0.670 | | **0.022** | | 0.024 | | 0.028 | | 0.212 | |

Table 4.7: ATE [m] and Tracking Rate (TR) on the TUM RGB-D sequences. In bold the best ATR.

An exception is observed in the F sequence with DynaPix, showing a 15% shorter tracking time compared to ORB-SLAM2. This is attributed to the camera facing featureless walls, immediately followed by an area with multiple dynamic entities, causing the SLAM system to lose track in some trials. Nonetheless, average results from synthetic sequences demonstrate the superiority of DynaPix and DynaPix+ over unmodified methods, with up to 3 times lower ATE, and for DynaPix+ $\sim$ 20% higher TR. DynaVINS, probably thanks to the different backend system, achieves the best tracking rate (+25%), at the expense of a ten times higher ATE when compared with DynaPix+. Overall, the best ATR, i.e. 0.127, is obtained by using DynaPix, with DynaPix+ being a close second with 0.214. DynaVINS, despite the higher TR, lags with an ATR of 1.471. Similarly, evaluations on TUM RGB-D (Table 4.7) reveal DynaPix achieving 23 times better ATE than ORB-SLAM2, with 100% TR (9.8% improvement). DynaPix+ has almost equal ATE as DynaSLAM, but with a 6% higher TR, primarily driven by 16% and 8% improvements in the *rpy* and *xyz* sequences, respectively. Meanwhile, WF-SLAM achieves the best overall ATE but is associated with the worst TR, approximately 24%. DynaVINS, on the other hand, cannot track the full trajectories, showing a TR of 97%. Moreover, it also exhibits higher tracking rates in general. Therefore, its ATR is roughly ten times higher than the one obtained with DynaPix+. This further highlights the importance of not *only* relying on the best ATE to validate the results of Dynamic SLAM methods, but also of the TR performance. Overall, these results indicate how retaining more features and keypoints, while dynamically weighting their contribution within the backend, can have a positive impact on both ATE and TR.

### 4.5.3 Ablation Studies

Our ablation studies for DynaPix and DynaPix+ on synthetic and real-world dynamic sequences are reported in Table 4.8 and Table 4.9. The studies include i) disabling movable estimation, i.e. the background differencing, ii) disabling moving estimation, i.e. the optical flow component, iii) removing dynamic weighting, i.e. the threshold-based keypoint selection without weighted BA, or iv) using estimated poses for both inpainting and splatting synthesis. This last aspect is applied only to TUM RGB-D sequences due to the lack of methods that achieve stable tracking on all GRADE sequences.

Disabling the movable estimation module affects both tracking rates and trajectory errors. On real-world sequences, DynaPix shows an average 35% reduction in tracking rate, highlighting how relying solely on optical flow is insufficient. Conversely, DynaPix+ is more stable, largely thanks to the segmentation module from DynaSLAM. However, trajectory errors are most affected by this ablation in synthetic sequences, particularly those with unknown flying objects that (as expected) pose challenges for pre-trained detectors. Excluding the moving estimation module instead impacts both trajectory errors and experiment repeatability, with greater variances observed in both sets of experiments. The overall increase in trajectory errors indicates that *only* relying on potentially moving elements from the scene is ineffective in all situations.

| | | DynaPix *w/o movable est.* | | DynaPix *w/o moving est.* | | DynaPix *w/ threshold* | | DynaPix+ *w/o movable est.* | | DynaPix+ *w/o moving est.* | | DynaPix+ *w/ threshold* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| FH | mean | 0.285 | 1.00 | 0.042 | 0.03 | 0.138 | 1.00 | 0.103 | 1.00 | 0.040 | 0.04 | 0.135 | 1.00 |
| | std | 0.127 | 0.00 | 0.007 | 0.00 | 0.136 | 0.00 | 0.135 | 0.00 | 0.033 | 0.00 | 0.097 | 0.00 |
| F | mean | 0.454 | 0.32 | 0.295 | 0.23 | 0.149 | 0.45 | 0.770 | 0.67 | 0.758 | 0.67 | 0.654 | 0.64 |
| | std | 0.341 | 0.19 | 0.159 | 0.02 | 0.188 | 0.24 | 0.459 | 0.18 | 0.460 | 0.18 | 0.491 | 0.22 |
| DH | mean | 0.005 | 0.18 | 0.008 | 0.18 | 0.006 | 0.18 | 0.004 | 0.18 | 0.005 | 0.18 | 0.007 | 0.18 |
| | std | 0.000 | 0.01 | 0.004 | 0.01 | 0.002 | 0.01 | 0.001 | 0.01 | 0.001 | 0.01 | 0.005 | 0.01 |
| D | mean | 0.049 | 0.97 | 0.106 | 0.69 | 0.023 | 0.99 | 0.038 | 0.98 | 0.079 | 0.86 | 0.051 | 0.99 |
| | std | 0.055 | 0.06 | 0.066 | 0.14 | 0.004 | 0.00 | 0.007 | 0.03 | 0.039 | 0.15 | 0.025 | 0.00 |
| WOH | mean | 0.084 | 0.54 | 0.047 | 0.54 | 0.011 | 0.54 | 0.061 | 0.54 | 0.021 | 0.54 | 0.012 | 0.54 |
| | std | 0.038 | 0.00 | 0.017 | 0.00 | 0.002 | 0.00 | 0.014 | 0.00 | 0.017 | 0.00 | 0.001 | 0.00 |
| WO | mean | 0.040 | 0.20 | 0.065 | 0.20 | 0.029 | 0.20 | 0.038 | 0.20 | 0.053 | 0.20 | 0.040 | 0.20 |
| | std | 0.008 | 0.00 | 0.033 | 0.00 | 0.004 | 0.00 | 0.005 | 0.00 | 0.005 | 0.00 | 0.005 | 0.00 |
| Average | | 0.153 | 0.53 | 0.094 | 0.31 | 0.060 | 0.56 | 0.169 | 0.59 | 0.159 | 0.41 | 0.150 | 0.59 |
| ATR | | 0.289 | | 0.303 | | 0.107 | | 0.286 | | 0.388 | | 0.254 | |

Table 4.8: ATE [m] and Tracking Rate (TR) of DynaPix and DynaPix+ ablation studies on GRADE **dynamic** sequences.

| | | DynaPix *w/o movable est.* | | DynaPix *w/o moving est.* | | DynaPix *w/ threshold* | | DynaPix *w/ est. poses* | | DynaPix+ *w/o movable est.* | | DynaPix+ *w/o moving est.* | | DynaPix+ *w/ threshold* | | DynaPix+ *w/ est. poses* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR | ATE | TR |
| w_half | mean | 0.053 | 0.31 | 0.214 | 1.00 | 0.096 | 1.00 | 0.091 | 1.00 | 0.026 | 1.00 | 0.031 | 1.00 | 0.026 | 1.00 | 0.028 | 1.00 |
| | std | 0.028 | 0.04 | 0.076 | 0.29 | 0.089 | 0.36 | 0.090 | 0.36 | 0.001 | 0.00 | 0.009 | 0.00 | 0.001 | 0.00 | 0.002 | 0.00 |
| w_rpy | mean | 0.046 | 0.64 | 0.206 | 0.99 | 0.050 | 0.97 | 0.074 | 0.99 | 0.034 | 0.90 | 0.041 | 1.00 | 0.029 | 0.96 | 0.038 | 1.00 |
| | std | 0.013 | 0.09 | 0.168 | 0.16 | 0.162 | 0.18 | 0.155 | 0.19 | 0.004 | 0.01 | 0.006 | 0.00 | 0.005 | 0.03 | 0.004 | 0.01 |
| w_static | mean | 0.011 | 1.00 | 0.153 | 1.00 | 0.010 | 1.00 | 0.013 | 1.00 | 0.007 | 1.00 | 0.013 | 1.00 | 0.007 | 1.00 | 0.008 | 1.00 |
| | std | 0.001 | 0.00 | 0.058 | 0.00 | 0.067 | 0.00 | 0.082 | 0.00 | 0.000 | 0.00 | 0.011 | 0.00 | 0.001 | 0.00 | 0.001 | 0.00 |
| w_xyz | mean | 0.019 | 0.90 | 0.052 | 1.00 | 0.167 | 1.00 | 0.026 | 1.00 | 0.015 | 1.00 | 0.015 | 1.00 | 0.015 | 1.00 | 0.015 | 1.00 |
| | std | 0.001 | 0.01 | 0.022 | 0.04 | 0.022 | 0.05 | 0.022 | 0.05 | 0.001 | 0.00 | 0.000 | 0.00 | 0.000 | 0.00 | 0.001 | 0.00 |
| Average | | 0.032 | 0.65 | 0.156 | 1.00 | 0.081 | 0.99 | 0.051 | 1.00 | 0.021 | 0.98 | 0.025 | 1.00 | 0.019 | 0.99 | 0.022 | 1.00 |
| ATR | | 0.049 | | 0.156 | | 0.082 | | 0.051 | | 0.021 | | 0.025 | | 0.019 | | 0.022 | |

Table 4.9: ATE [m] and Tracking Rate (TR) of DynaPix and DynaPix+ ablation studies on TUM RGB-D Walking sequences.

Notably, in this scenario, DynaPix's TR for the FH sequence drops to 0.03 (0.04 for DynaPix+) due to the numerous *potentially* dynamic objects observed by the camera at the beginning of the experiment. This highlights again the importance of jointly analyzing ATE and TR, especially in complex scenarios.

We then adopt standard BA optimization, i.e. without weights, and set a motion probability threshold of 0.05 for keypoints, i.e. features with higher values are excluded from the process. This significantly impacts trajectory errors in both experiment sets, suggesting a beneficial effect of integrating weights within the SLAM backend. Notably, DynaPix under this setting performs better on GRADE sequences, but with increased variances. Conversely, DynaPix+ demonstrates slightly lower trajectory errors and comparable tracking rates on TUM RGB-D, due to the segmentation module effectively removing the majority of keypoints associated with humans.

Ultimately, we use pose estimates from a previous run of DynaSLAM for splatting view synthesis and inpainting on TUM RGB-D sequences. For unavailable poses, such as during periods of non-tracking, we resort to the estimated poses from the nearest neighboring frames, where most errors can be reduced by *flow differencing*.

Overall, these experiments illustrate the positive effect of our introduced components on both tracking rate and trajectory error. This is also closely related to the specific characteristics of each testing sequence. Indeed, for scenes where humans are the only moving entities and positioned far from the camera, a simple segmentation method may suffice. However, the presence of additional moving objects, as seen in the F and FH sequences from GRADE, or failures in the detection network, as with rotated humans in *rpy* sequence, necessitates a more generalized approach like DynaPix.

## 4.6 Conclusions

In this chapter, we not only verified that GRADE can be used to extensively test Visual SLAM approaches, but we also introduced DynaPix. DynaPix is an offline Dynamic Visual SLAM method that innovatively integrates pixel-wise motion probability estimation into a customized SLAM framework. The dual-stage process blends *movable* and *moving* estimations from frame differencing and splatted optical-flow subtraction, respectively. These motion probabilities are then employed in the tracking and backend optimization procedures of ORB-SLAM2 via a dynamic weighting mechanism. Through extensive testing on synthetic and real-world sequences, we underscore the importance of jointly analyzing trajectory errors and tracking rates in SLAM evaluation and showcase the superior performance of DynaPix and DynaPix+ over ORB-SLAM2, DynaSLAM, DynamicVINS, and WF-SLAM. Our findings highlight the advantages of distinguishing between static and moving parts of dynamic entities and weighting their influence based on motion status, thereby improving the robustness and accuracy of dynamic V-SLAM methods.

# Chapter 5

# Using Synthetic Data for UAV-Based Zebra Detection and Pose Estimation

Unlike the case for human-centered data, large-scale annotated datasets for wild animals are scarce. Wildlife monitoring also presents additional challenges, including out-of-distribution aerial viewpoints (to avoid robot-animal interference), diverse environments, and significant domain shifts. Moreover, collecting and annotating real-world data in these settings is costly, labor-intensive, error-prone, and often impractical. Thus, synthetic data is increasingly being used to overcome these issues. However, most prior works still rely on real images, style adaptation techniques, or pre-trained models to bridge the syn-to-real gap.

In Chapter 3, we demonstrated that syn-to-real learning is effective for human detection and segmentation in indoor environments. A key factor in this success was the high variability in human shapes, poses, and appearances in our generated datasets. Building on these findings, in this chapter we use *only* synthetic data to train models capable of detecting and estimating the 2D poses of wild animals — specifically zebras — observed from aerial viewpoints. While detection is crucial for localizing animals, pose estimation enables activity recognition and can serve as a prior for shape estimation and health assessment [22]. To this end, we extend GRADE to generate a fully photorealistic dataset of zebras observed from multiple aerial perspectives. Our approach ensures high variability and control over environmental factors, avoids real-world data collection challenges, and finally enables training models exclusively on synthetic data. Moreover, previous works mostly focus on pose estimation and often assume that accurate detections are readily available. However, as we will show, this is often not the case as these two tasks, while related, have distinct challenges. Detection must handle occlusions, background clutter, and scale variations, whereas pose estimation focuses on capturing an animal's skeletal structure. Our approach, called ZebraPose, is the first full top-down approach that jointly addresses both tasks.

To assess our performance we conduct extensive experiments using YOLOv5 (detection) and ViTPose+ (2D pose estimation). Specifically, we examine how models trained purely on synthetic images generalize to real-world aerial and non-aerial datasets, analyzing different training strategies, dataset compositions, and backbone initializations. Our results demonstrate that synthetic data alone can be sufficient for training models that generalize effectively to real images, even in challenging scenarios. To further support our analysis, we introduce a new, extensive real-world dataset collected at the Wilhelma Zoo in Stuttgart.

This chapter is further structured as follows: we first introduce the problem (Section 5.1) and review the state of the art in top-down animal 2D pose estimation (Section 5.2). We then describe our datasets. First, our generation process using GRADE in Section 5.3, detailing how we ensure visual realism and diversity, and then our real-world dataset (Section 5.4). Next, we perform an extensive analysis of model performance across various real and synthetic test sets and modalities in Section 5.6. Through this study, we establish that synthetic data is a viable alternative for training robust animal detection and pose estimation models, with significant implications for wildlife conservation and ecological monitoring (Section 5.7).

Figure 5.1: An example image of our synthetically generated zebras in a Savanna environment.

## 5.1 Top-Down 2D Zebras Pose Estimation

Having a large dataset that includes realism and diversity in features is a fundamental building block for obtaining any working and reliable deep-learning model. This is especially true when dealing with visual tasks such as detection, semantic segmentation, and shape estimation. A variety of datasets have been introduced during the last decades to address various image-based tasks like MNIST [115], COCO [129], and PASCAL-VOC [62]. These have historically been based on real-world data, be these images or videos, often manually labeled by humans. However, manual annotation can be costly, time-consuming, and prone to errors [184], as we also show in Figures 5.3 to 5.5. Alternatively, applications like human pose estimation can also leverage large-scale datasets obtained in controlled environments and motion capture systems [140] that help in this process. Still, semi-automatic systems such as VICON halls are impractical for animals, and they generally present several limitations, such as the limited number of subjects they can track or the constrained scenarios. These limitations are particularly significant in problems such as aerial human pose estimation [187], animal pose estimation [270, 271], or aerial wild-animal detection. Moreover, these datasets are usually application-specific and can hardly be generalized to different scenarios, tasks, or data. For example, wild animals are widely under-represented in datasets such as COCO or PASCAL-VOC [31, 254, 256]. Indeed, apart from a limited number of labeled images and videos of uncommon animal species such as zebras, hippopotami, and giraffes, there

is also a general lack of variety of scenarios in which those are recorded. Taking zebras as an example, there are only 1916 training and 85 validation images containing at least one instance of them in the COCO dataset. For these reasons, as we also saw previously in this thesis, methodologies to generate synthetic data have become more ubiquitous since the advent of rendering engines such as Unity, Blender, Unreal Engine, and Isaac Sim. These are advantageous in multiple aspects since they allow the generation and automatic labeling of ground truth data through full customization possibilities and with minimum human effort. Indeed, synthetic data has been used in a variety of tasks, such as human detection [59], pose and shape estimation of humans [20, 187], and semantic segmentation [189]. However, they usually lack the visual realism necessary to generalize well to real-world data if used alone. Thus, a combination of real and synthetic data, or other adaptation techniques, are often utilized [59, 99, 100].

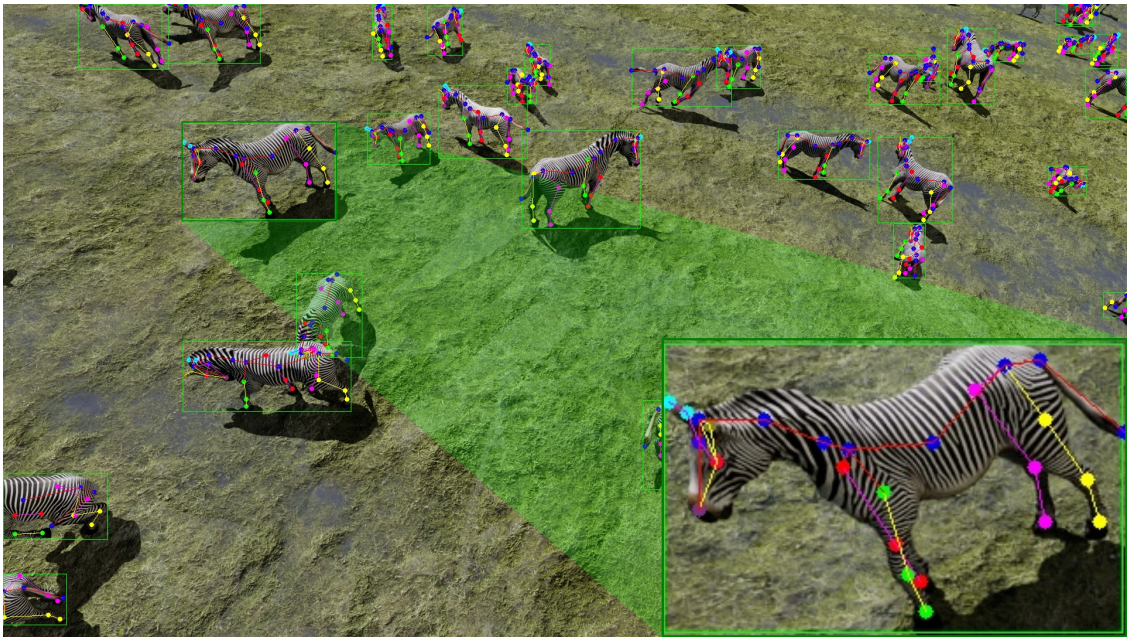


Figure 5.2: A sample of our synthetic data. Zoomed inset: an individual with all the 27 keypoints labeled.

In this chapter, our focus is on aerial-based detection and pose estimation of wild animals. This is instrumental in providing valuable insights by enabling non-intrusive monitoring of animal health, motion patterns, and interactions with their natural environment [269]. However, the availability of animal-focused datasets collected in the wild remains scarce. Additionally, many existing ones lack aerial perspectives and out-of-distribution viewpoints [13]. This domain gap remains a critical bottleneck for conservation-focused DL applications, limiting their real-world in-the-wild applicability. Moreover, wild creatures cannot be easily restrained without inducing stress, altering their behavior, or removing them from their natural habitats [23, 226]. As such, obtaining

diverse and accurately labeled animal data remains a significant challenge, making research in unconventional domains, such as wildlife monitoring, animal pose estimation, and aerial image analysis, hindered by the lack of labeled data [98, 99, 100]. The challenge lies in acquiring diverse, high-quality datasets that enable DL models to generalize effectively across different environments and observation points. This is particularly pressing for species like Grévy's zebra, i.e. the focus of this chapter, an endangered herbivore whose population dynamics can influence entire ecosystems [204].

As aforementioned, a promising approach to overcoming these limitations is synthetic data [119, 271]. Synthetic datasets enable precise annotations, controlled environmental conditions, and the generation of large-scale, diverse samples. However, while both detection and pose estimation can benefit from synthetic data, their requirements differ. Detection performance depends on background realism, camera-subject distance, and viewpoint variability, whereas pose estimation relies on capturing intra-body relationships and depth cues [59, 98]. Generating effective synthetic datasets thus requires both (i) an automated procedure to cover the target data distribution and (ii) strategies to bridge the synthetic-to-real gap. Many existing approaches address only one of these aspects, often simplifying backgrounds or relying on domain adaptation techniques such as style transfer or real-image supervision [13, 119]. As a result, achieving robust real-world generalization using purely synthetic data remains an open challenge [98].

To solve this problem, and building upon our previous findings (Section 3.6.2), we generate a new synthetic dataset using our GRADE framework, a publicly available animated zebra model, and environments from the Unreal Engine marketplace. An example of the generated data can be seen in Figures 5.1 and 5.2. Using this, we propose a unified approach for *both* detection and 2D pose estimation of zebras using *only* synthetic data, i.e. a full top-down method. Unlike prior pose estimation works that often assume a pre-trained detector, we show that detection itself can be a bottleneck, especially in aerial settings (Section 5.6). First, we use the generated data to train YOLOv5s models for zebra detection and evaluate them on (i) existing real-world datasets and (ii) our newly introduced high-resolution dataset of 104K precisely labeled aerial images, showing that training with our synthetic data outperforms the baseline models trained on real-world datasets. However, our evaluations also indicate how the generated data can hardly generalize to common, non-aerial, close-to-the-zebras viewpoints. This is a consequence of our data generation strategy, which was initially focused on far and aerial point-of-view. Indeed, after increasing the dataset variability through systematic image cropping, scaling, and augmentation, the generalization to non-aerial images (e.g. the ones in Figure 5.9) significantly improves, showing once more how the quality of the data and the domain coverage are important factors. After that, we automatically annotate all synthetic animals with 27 ground-truth keypoints (i.e. the 2D pose) extracted from their 3D meshes and train a ViTPose+ [248] model using both pre-trained and randomly initialized backbones. Extensive benchmarking confirms the generalization of our models to real-world zebra imagery. We further demonstrate that minimal real data enables effective adaptation to horse pose estimation.

In summary, our main contributions are:

- A large-scale, richly annotated, synthetic dataset of zebras collected in different wild environments.
- A *generalized* detection and 2D pose estimation pipeline for zebras trained exclusively on synthetic data, validated through extensive benchmarking on real-world datasets.
- An in-depth study on the synthetic-to-real gap for both detection and pose estimation, analyzing how dataset variability and augmentation improve generalization.
- A large dataset of zebras observed by UAVs and precisely labeled with bounding boxes.

## 5.2 Related Works

There are not many animal-based datasets available in the literature [31, 125, 254], especially considering full 3D-vertices information and precise segmentation. This is related to the difficulties of collecting and labeling ground truth data in outdoor scenarios. Various approaches have been applied to overcome this problem, ranging from using toy models [270, 271], merging different datasets [254], or using synthetic data [119, 125]. However, all of them fall short in some aspects, such as a lack of animal species variability, size, pose and shape information, skeletal joints location, or limited capturing settings. For example, Horse-10 [143] has only horses moving left-to-right. The Grévy's zebra dataset [77] consists only of 900 low-resolution images that do not contain either correct bounding boxes or labels for all animals. An example of this is provided in Figure 5.3. AnimalPose [31] focuses on a limited set of animals, in which zebras are not included. The 4DComplete dataset [125], although it contains various animal animations, textures and FBX files were not released with it, making it impossible to customize. While they provide rendered RGB-D images and scene flow, the renderings do not include background information. Other synthetic datasets, such as the one from Mu et al. [150], contain data that cannot be used to train a successful detector since they are generated with unrealistic backgrounds and textures [150]. These also suffer from the low viewpoint variability and diversity of scenarios, such as the data from COCO. We must also note that, as shown in Figures 5.3 to 5.5, COCO and other datasets are not exempt from wrong or imprecise labeled data.

Moreover, due to the diversity observed within the animal kingdom—even among closely related species such as *Canidae* and the challenges in acquiring (pseudo)ground-truth data, it is difficult to construct a universal animal model analogous to what SMPL is for humans [138]. As a consequence, synthetic data generation for animals typically involves stitching onto randomized backgrounds CAD or SMAL-based models [270, 271] that are animated via manual manipulation, pre-fitting to real images, or

Figure 5.3: Examples of missing bounding boxes and keypoints from the Grévy's zebra [77] dataset. Image IDs 869 (left) and 882 (right).

posing VAEs [31, 99, 100, 119, 150]. While this approach enables rapid data generation, it often introduces scale inconsistencies, poor blending with the scene, and unrealistic lighting. Indeed, existing synthetic datasets are primarily designed for 2D pose estimation (i.e. keypoints estimation), focusing on joint orientation and positioning rather than overall visual fidelity [98]. This limitation, coupled with the absence of robust priors, leads many studies to face real-world generalization issues. To mitigate the synthetic-to-real gap, researchers frequently rely on domain adaptation or semi-supervised learning techniques—leveraging large quantities of unlabeled real images or enforcing consistency, style, and other constraints during data generation [100, 119, 120, 150]. However, despite some performance gains from incorporating unlabeled data, these methods often suffer from overfitting and noise in the refinement of synthetic data predictions [98, 99, 120], and they are typically evaluated on only a limited set of datasets. Moreover, while recent approaches such as [132, 263] have integrated large language models (LLMs) with synthetic data to address common issues like left/right flipping, their performance still lags behind state-of-the-art models such as ViTPose+ [248]. Recent work on human pose estimation [20] further highlights the importance of photorealism in bridging the synthetic-to-real domain gap.

Finally, a critical limitation of current keypoints estimation methods using synthetic data is the underlying assumption that detection is already solved [98,99]. Most synthetic datasets consist of tightly cropped images around the animal [119,150], which speeds up rendering and yields crisp images ideal for pose estimation. However, this cropping renders the data unsuitable for training detection models since the animal always occupies the full image and occlusions are rare.


(a) Elephant labeled as a zebra

(b) Two unlabeled individuals

(c) Bounding box is not tight

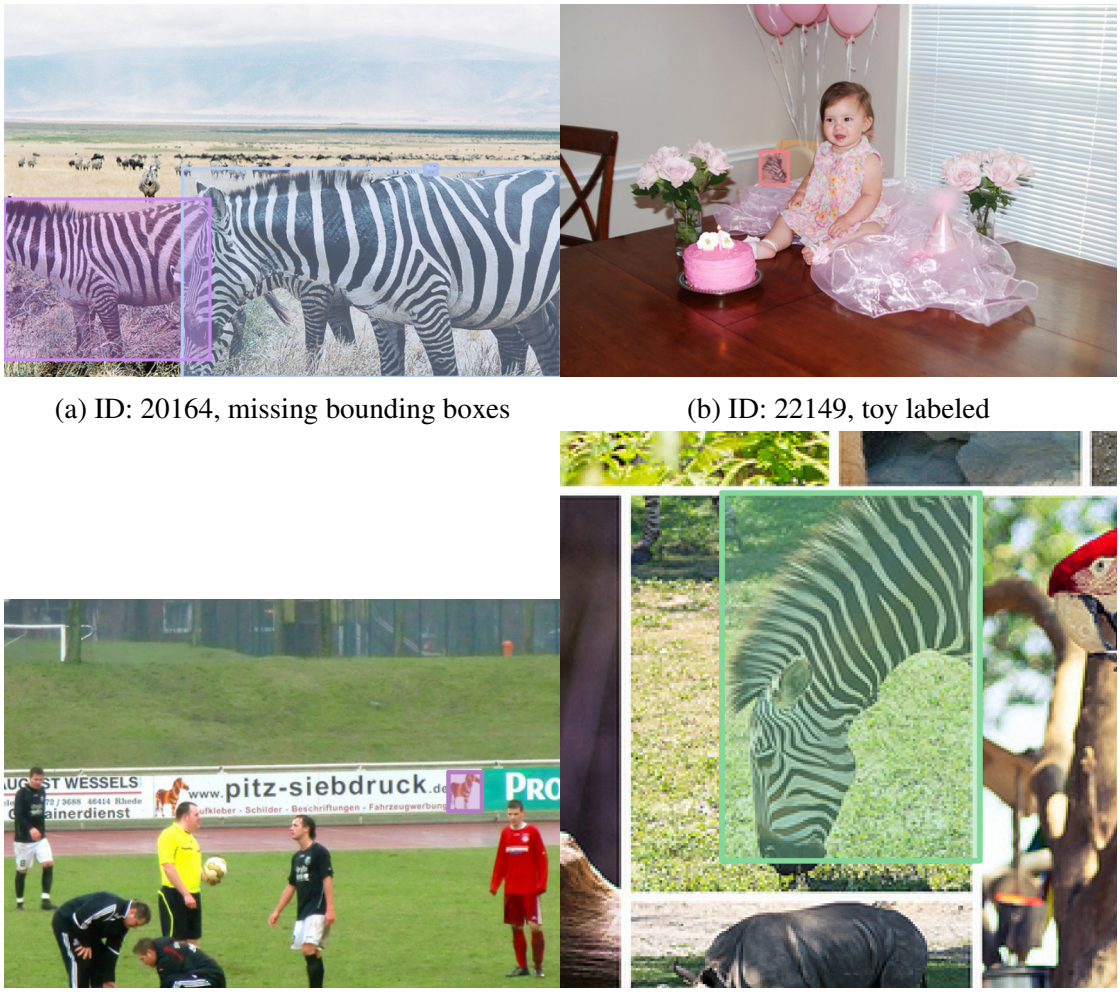
(d) Bounding boxes are not tight

Figure 5.4: Examples of annotation errors in the APT-36K dataset.

Therefore, most works do not evaluate the joint task of detection and pose estimation using synthetic data, assuming off-the-shelf, high-quality detections. In ZebraPose, we leverage GRADE first to learn to detect animals directly from synthetic images. In our approach, we generate visually realistic data using easily obtainable models within a photorealistic simulator. Moreover, we evaluate the impact of image resolution in either training or validation and show how it might significantly impact performance. While doing that, we also implement an effective data augmentation technique that allows us to use our synthetic data, whose viewpoint generation strategy predominantly produces aerial perspectives, to train models working in commonly available real-world datasets. Finally, we employ the same dataset to learn a 2D pose estimation method, obtaining the first full top-down 2D pose estimation approach trained only on synthetic images.

## 5.3 Synthetic Data Generation

Using the GRADE framework, we generate a dataset of zebras living within outdoor environments. Notably, thanks to the flexibility of GRADE, this approach will be easily applicable also to other animal species or setups. Similarly to the previous section, here we proceed to describe the environments, assets, and generation procedures.




(a) ID: 20164, missing bounding boxes

(b) ID: 22149, toy labeled

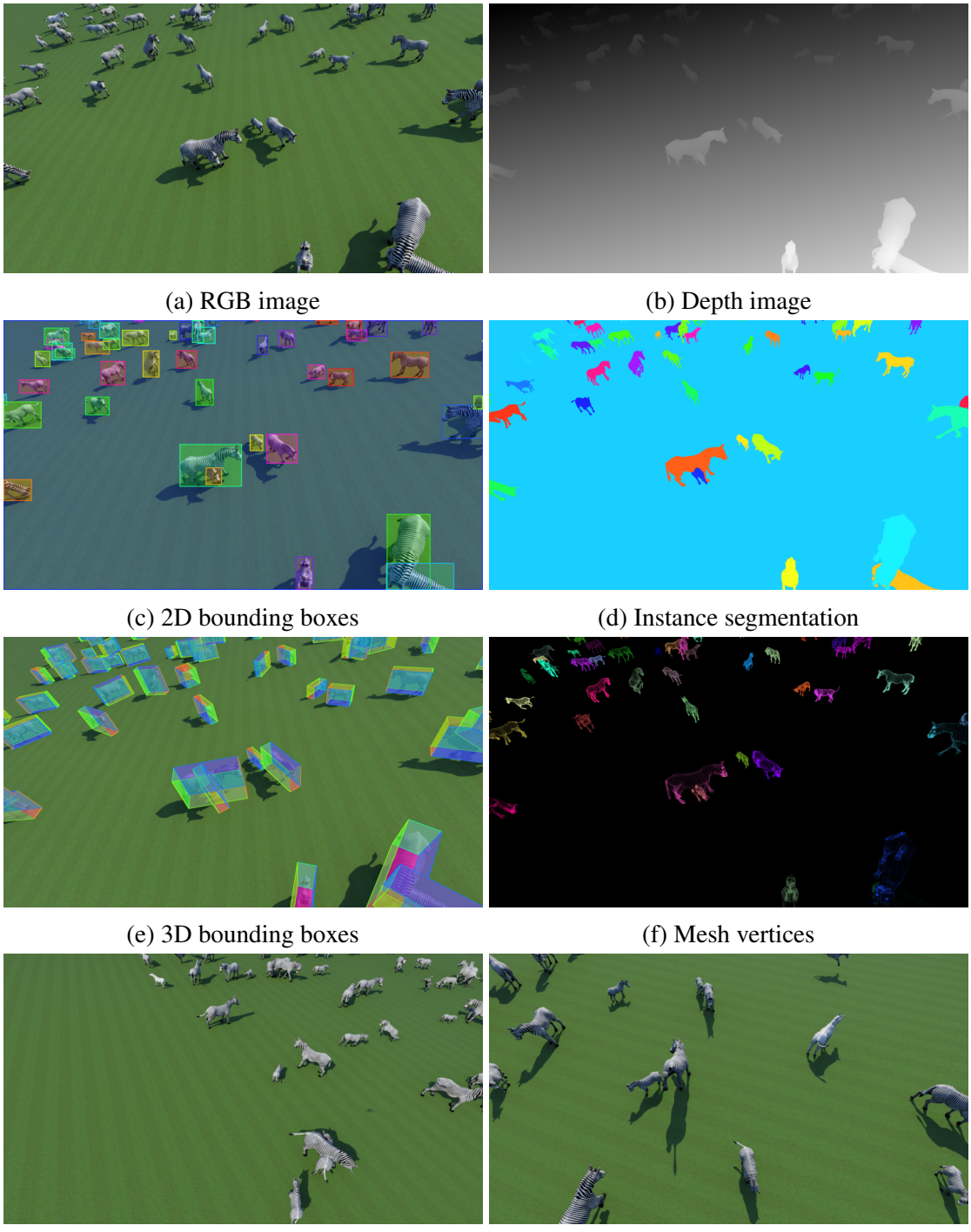
(c) ID: 32206, wrong bounding box

(d) ID: 533961, imprecise bounding box

Figure 5.5: Examples of wrongly labelled zebras from the COCO [129] dataset.

### 5.3.1 Environments

We selected nine commercial and one freely available environments from the Unreal Engine marketplace. We used the Unreal Engine Omniverse connector to convert them to the USD file format. We list the environments with the corresponding shortened URL in Table 5.1. For every environment, we use the available demos and pre-built scenarios directly, without introducing any custom changes. Then, we proceeded to remove the original sky sphere and fix textures when necessary, as the connector does not support full export of the terrains from Unreal Engine, resulting in a lower level of detail, e.g. missing 3D grass, some textures, and level of details. We replaced the textures with some taken from Isaac Sim itself, resembling the *color* of the grass.



(a) RGB image (b) Depth image

(c) 2D bounding boxes (d) Instance segmentation

(e) 3D bounding boxes (f) Mesh vertices

(g) Second view (h) Third view

Figure 5.6: An example of the generated data following the pipeline described in Section 5.3.4.

| Environment Name | URL |
|---|---|
| Bliss | https://bit.ly/3HD3zYP |
| Forest | https://bit.ly/3mYQv8Z |
| Grasslands | https://bit.ly/3HD3zYP |
| Iceland | https://bit.ly/3Ax8zKi |
| L_Terrain | https://bit.ly/3V6H7MU |
| Meadow | https://bit.ly/3Hgxk1n |
| Moorlands | https://bit.ly/3oHT1ku |
| Rural Australia (Free) | https://bit.ly/3i5j6Hi |
| Windmills | https://bit.ly/3AvVTDK |
| Woodland | https://bit.ly/3mYQv8Z |

Table 5.1: Names and shortened URLs of the used environments.

### 5.3.2 Dynamic Assets

We use the same zebra model from SketchFab we used in Section 3.4.1. It consists of 17 different in-place animation sequences, i.e. without root translation or rotation movements, for a total of 888 animation frames. We converted each animation sequence to the USD format using Blender and its Omniverse connector, leveraging the work we already did for humans. Then, we post-processed the sequence to obtain per-frame vertex positions and skeletal information. This allows us, for every generated frame, to have corresponding ground truth information on these two characteristics. The vertices are used to compute oriented bounding boxes we then employ in the placement procedure.

### 5.3.3 Placement of the Zebras

Zebra placement is based on an ad-hoc procedure, repeated every time a frame is generated. We select a specific mesh for every environment as `terrain`, representing the area over which we will place the zebras. The placement consists then of four main steps: i) selecting a random rectangular area of the terrain, ii) randomly selecting a set of zebras, iii) choosing for each zebra a frame of its animation sequence, a scaling factor, and a global orientation of the zebra, iv) place the zebra in the rectangular area considering the bounding-box occupancy. The sides of the rectangle are randomly selected to be between 40 and 120 meters, while the scaling factor ranges between 40% and 100%, allowing us to obtain a higher degree of variability. The placement is an iterative procedure that considers one zebra model after another. Any model that cannot be placed following a detected collision is removed from the simulation. The final results depend mostly on the resolution of the terrain mesh for both collisions between meshes and contact of the zebras with the ground. In general, we noted that collisions are rare and that contacts

with the ground are good, although the hoofs of the animals sometimes fall below the surface, as we do not optimize for the roll and pitch angles of the asset.

### 5.3.4 Data Generation Methodologies

Contrary to what we did for indoor environments in Section 3.5, here we focus on image generation rather than video sequences. We also perform a series of randomization for *each* captured frame, i.e. the i) time of day, ii) number of zebras in the environment, iii) their scaling factor, iv) their specific animation frame, and the v) placement of three cameras that will record the scene. Specifically, given any environment, we set up three aerial cameras and randomly pre-loaded 250 zebras at the beginning of each experiment. We then uniformly select the number of zebras that will be placed in the next frame. This number is set to be between 2 and 250. Note that this is *not* the number that will appear in each frame, nor is it the final number of zebras that are actually placed. As explained above, the placement strategy may remove some zebras, and the camera may not observe all the zebras from a given standpoint. Once the placement happened, we randomized the location of the cameras and the time of day three times. The time of day will be 90% of the time between 5 am and 8 pm, which results in good lighting conditions given our current settings, and 10% of the time in the remaining hours, resulting in dusk-to-night light settings. This further varies the appearances of both the generated frames and the shadows. Cameras are placed using the average location of the zebras as a pivot point. For the placement of the cameras, we distinguish between two slightly different image-generation procedures: one more general and one more focused on capturing zebras from a nearer viewpoint. We first describe the former and then identify the minor modifications that we applied to the latter. From the pivot point, we randomize the distance in the x-y plane and the height of the camera. The height is set to be between 5 and 20 meters more than the average of the zebras, while the x and y are set to be between -/+ 100 meters. Once the position is fixed, we can compute and randomize roll, pitch, and yaw. Roll is set to be within $[-10,10]$ degrees, yaw is set to be the ray that connects the camera and the pivot point with an additional random $[-30,30]$ degrees. Pitch is computed as the $atan2(pivot_z - cam_z, d(pivot, camera)) + 15$ degrees, where $d(pivot, camera)$ is the distance in the x, y plane. The modifications applied to this methodology during the second image-generation procedure are as follows: the x and y positions are set to be within 5 meters of the virtual bounding box containing all zebras, the yaw has an additional $[-15,15]$ degrees component instead of the $[-30,30]$. This results in images that are closer to the zebras than the ones obtained from the former camera placement strategy.

For each environment, we randomly place the zebras 200 times, resulting in 600 frames per experiment and camera, i.e. 1.8K frames in total. After the generation process is complete for all ten environments, this totals 18K frames captured with the first camera placement strategy, and 18K with the camera set to be closer, i.e. 36K frames. For each frame, we save the pose of the cameras, zebra skeletal pose and meshes vertices, ground

truth depth, and instance segmentation, as shown in Figure 5.6.

Figure 5.7: An example of before [left] and after [right] the cropping and scaling procedure (Section 5.3.5).

### 5.3.5 Synthetic Data Enhancement

By analyzing the performance of the detectors trained with the data generated using the above sections, we notice that the trained models cannot generalize to common images, especially when the zebra is close to the camera and the viewpoint is low. Since we rely on a photorealistic simulator, use realistic assets, and achieve good performance in the detection from aerial views (see Section 5.6.1), we can assume a good general quality of the data in terms of photorealism. We thus argue that the reason for the only synthetic-trained detector to fail on common images is related to the actual distribution of the viewpoints and the relationship that exists between the size of the individuals and images during testing. Indeed, since cameras' locations are uniformly randomized, the number of times they are near the animals is much lower than when they are far from them. Moreover, the training and validation input dimension of YOLOv5 can impact the performance of the network itself, as we show in Tables 5.6 and 5.7, since scaling influences both the size of the objects in the scene and the visual quality. For example, a 90px wide box in a $1920 \times 1080$ image when scaled to a $640 \times 480$ image becomes three times smaller, i.e. just 30px wide.

Following the intuitions above, instead of generating new synthetic data by creating a new pipeline, we augment the previous dataset to obtain wider coverage of the desired distribution. Since the original dataset has been rendered from random viewpoints, the individuals in the images tend to be small. Therefore, to create images with bigger, and thus virtually 'closer' to the camera, animals, we employ a new targeted crop and scale procedure. We first select all the animals whose bounding box area is greater than a given threshold for each image in the training and validation sets. Each box is randomized using a random offset between 0 and 150 pixels on each side, creating a rectangular area around the animal. We then cropped this box and rescaled it to the original image size (i.e. $1920 \times 1080$). The annotations are finally generated from the upscaled ground-

truth segmentation masks of the crop itself. Note that the upscaling operation degrades the quality of the final image, as it acts as a digital zoom, and we do not re-render the scene. We set the threshold on the original bounding box area to 5000 pixels to filter excessively small animals. An example of a zoomed-in individual is given in Figure 5.7, while the variation of the cumulative density function of the boxes' dimensions is shown in Figure 5.8.

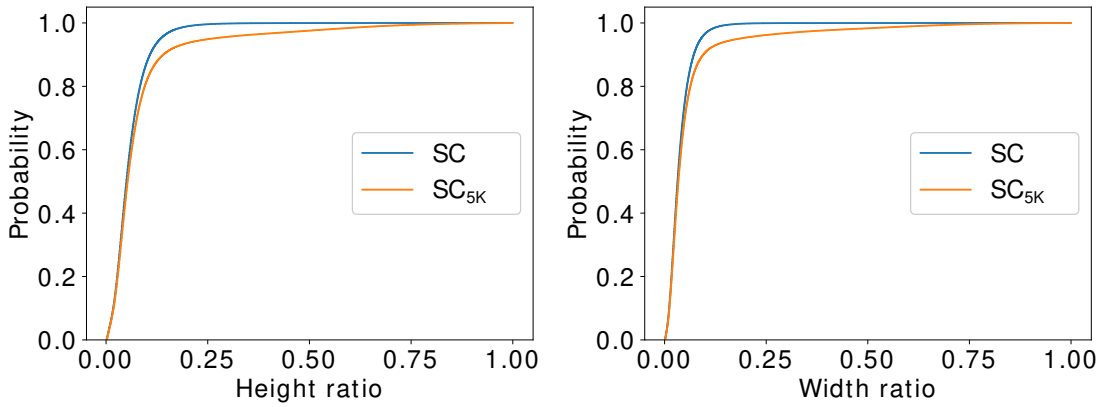
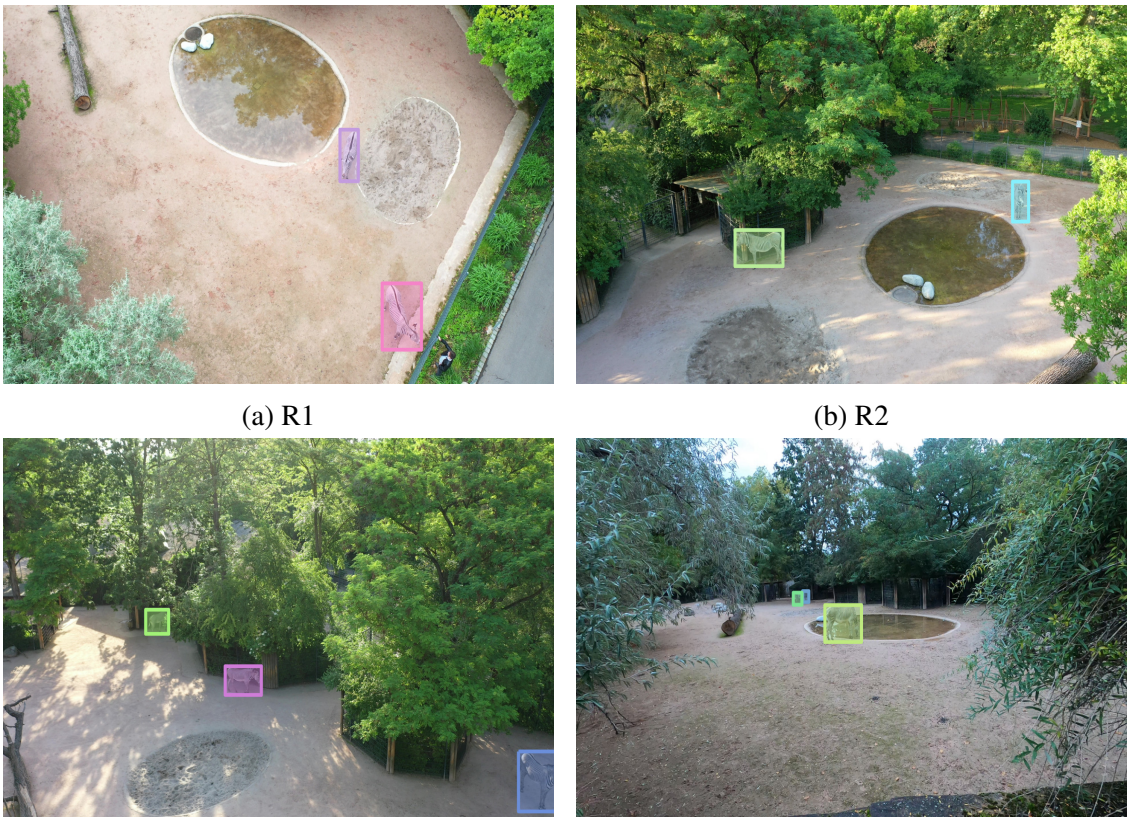



Figure 5.8: Cumulative Distributions of height and width ratio with respect to image size for SC and $SC_{5K}$ datasets.

### 5.3.6 2D Pose Labels

To obtain the keypoints information, we exploited the known 3D location of the camera and the 3D vertex location for each individual. We create 27 different groups of vertices, corresponding to 27 different keypoints. The resulting keypoints, depicted in Figure 5.2, are: four hoofs, four knees, four thighs, the tail start and end locations, eyes, ear tips and bases, start and end of the neck, nose, skull, body middle, and back end and front. From each set of vertices, we compute the 3D average location and project it to the image. Using the COCO convention, we mark as *visible* the keypoints within the *instance* mask of the animal, and as *occluded* the remaining ones. Note how this procedure is easily customizable. For example, by leveraging the known depth information or additional heuristics, one could skip some keypoints or decide to label all occluded ones. We label only the individuals whose maximum bounding box dimension, either width or height, exceeds 30 pixels. As we use a top-down approach, each annotation is separately cropped and scaled to the input size of the selected model and only then used for training and validation, with the loss computed only on the labeled keypoints. This automatically diminishes the impact of the relative distance between the camera and the animal, while influencing the sharpness of the final crop. Finally, we map the 17 canonical keypoints of datasets such as APT-36K to our 27 to be able to train models using mixed data. Note that

the whole set of animations of the synthetic zebra model consists of only **888** different poses. Out of those, **440** are part of *idling* ones, thus practically static; this makes the "animated" ones a rather limited amount if compared to other datasets [100, 150, 207].



(a) R1 (b) R2

(c) R3 (d) RP

Figure 5.9: Examples of our real world collected images used for testing from the specified dataset. Three aerial and one 'ground-level' view. Best viewed in color.

## 5.4 Real world data

We performed several data collection experiments in a controlled scenario within the Wilhelma Zoo in Stuttgart (DE). An example of the collected data can be found in Figure 5.9. We used two manually flown DJI Mavic 2 Pro drones, recording images at 29.97 fps at a resolution of $3840 \times 2160$, and three GoPro Hero8, also with a resolution of $3840 \times 2160$ at 59.94 fps. None of the GoPros had fixed locations between experiments. The data was manually synchronized using a recorded light signal that was visible to all cameras at the same time. We then extracted one frame every five seconds from all the videos. Of these, 905 images were randomly selected and annotated manually and

precisely. These annotations were then used to train an SSD multibox [133] detector, which, with *Smarter-Labelme* [165], allowed us to obtain bounding boxes on our video sequences with ease. Out of all the data available, we finally selected three collections during which the zebras were visible by both drones. The boxes on those sequences were then manually refined in a final step. This procedure thus resulted in 905 precisely annotated images and 104K frames annotated with [165], then manually checked and refined.

## 5.5 Datasets and Metrics

### 5.5.1 Datasets

In the next sections, we will use the short names of the datasets introduced here (in bold). When mixed datasets are to be used during training procedures, we will concatenate the naming using the '+' symbol. A recap of the datasets used, the animal(s) within them, and their training/validation sizes are reported in Tables 5.2 to 5.4.

#### Synthetic

The Synthetic Full dataset, **SF**, is the dataset containing all the 36K synthetically generated images. These are then randomly shuffled and split into 80/20 train/validation sets. Synthetic Closeby, **SC**, is the synthetic data generated only by the second strategy, as described in Section 5.3.4, i.e. 18K images for which the camera is within 5 meters of the bounding box containing all the zebras. This data is also divided randomly with an 80/20 ratio. For both SC and SF we also have $\mathbf{SC}_{640}$ and $\mathbf{SF}_{640}$, to indicate a different scaling factor of $1920 \times 1920$ and $640 \times 640$ respectively during YOLOv5 trainings. We then use SC to generate both the keypoints (Section 5.3.6) annotations and an augmented dataset (Section 5.3.5), called $\mathbf{SC}_{\mathbf{5K}}$. $\mathrm{SC}_{5K}$ consists of 29K images divided again with an 80/20 ratio. We also use the zebra images from the **SpacNet** [100] data, consisting of 3000 generated frames with style transfer applied to them. We use this to evaluate the quality of the synthetic data generated on the 2D pose estimation task, comparing a vanilla generation with a more complex method (fully described in [100]).

| | SC | $\mathrm{SC}_{640}$ | $\mathrm{SC}_{5K}$ | SF | $\mathrm{SF}_{640}$ | SpacNet [100] |
|---|---|---|---|---|---|---|
| Train | 14401 | 14401 | 23184 | 28801 | 28801 | 2640 |
| Valid | 3599 | 3599 | 5798 | 7199 | 7199 | 360 |

Table 5.2: Summary of the *synthetic* datasets used in this chapter. The numbers indicate the number of images in each set. Both $\mathrm{SC}_{640}$ and $\mathrm{SF}_{640}$ are used to indicate that the images of those datasets are scaled to $640 \times 640$ and not $1920 \times 1920$ during YOLOv5 *training*.

### Real — Aerial Viewpoints

**R1**, **R2**, and **R3** are the three sets of real world data that were collected and labeled by us with (almost) pixel-level precision bounding boxes as described in Section 5.4. To distinguish between which drone captured the given sequence, we use the suffixes *_D1* and *_D2*. R1 consists of 19.7K images, R2 of 23.4K, and R3 of 8.8K, for each drone. Therefore, the combined **R123** dataset contains 104K images labeled with precise bounding boxes. Moreover, we use $R3_{100}$ to indicate a small subset of R3 of 100 training and 100 validation images that we use in YOLO training. We will finally use **RP** to indicate the set of the 905 real world images precisely labeled by us, sampled from representative images from the previous Rx datasets and additional images captured with the GoPros during the same experiments. Of them, 720 are randomly used in training and 185 for validation. As before, $RP_{640}$ is used to indicate that the same dataset scaled to $640 \times 640$ pixels is used during training of YOLOv5 instead of $1920 \times 1920$. An example of the bounding boxes of our real world data is provided in Figure 5.9. We also provide two zoomed-in examples of not pixel-perfect labels in Figure 5.10. Note that other datasets, e.g. COCO, also present such approximations, as in Figures 5.3 to 5.5.

| | R1_D[1,2] | R2_D[1,2] | R3_D[1,2] | $R3_{100}$ | R123 | RP | $RP_{640}$ |
|---|---|---|---|---|---|---|---|
| Train | — | — | — | 100 | — | 720 | 720 |
| Valid | 19725 (x2) | 23441 (x2) | 8874 (x2) | 100 | 104080 | 185 | 185 |

Table 5.3: Summary of **our** real world datasets of Zebras observed from aerial views used here. The numbers indicate the number of images in each set. With Rx_D[1,2] we indicate to which drone the images belong, while with $R3_{100}$ the subset of 100 images that we used in training YOLOv5. R123 is the combination of all the Rx_Dy. $RP_{640}$ is used to indicate that the images of RP are scaled to $640 \times 640$ pixels and not $1920 \times 1920$ during YOLOv5 *training*.

### Real — Common Viewpoints

Contrary to previous works, we use multiple real-world datasets to evaluate our data and thoroughly test our method's generalization and flexibility. This is necessary as some datasets, such as **Zebra-300** [99], are simpler than others (Section 5.6). We adopt the **A**P-**10**K [256] dataset, named **A10** in this thesis. The full dataset consists of 10015 images of various species already divided into training, validation, and test sets. Like other prior works [98, 100, 119, 150], we adopt the first split among the ones available. We subdivide A10 into $\mathbf{A10_{OZ}}$, i.e. the subset containing **O**nly **Z**ebras, and $\mathbf{A10_{99}}$, i.e. a random subset of **99** zebras (described in [99]). We also use the **A**PT-**36**K [252] dataset, named as **A36**, consisting of 2400 videos, summing up to 35K frames, of different animals. Since an official split has not been released, we divide the dataset using an 80/20 ratio,

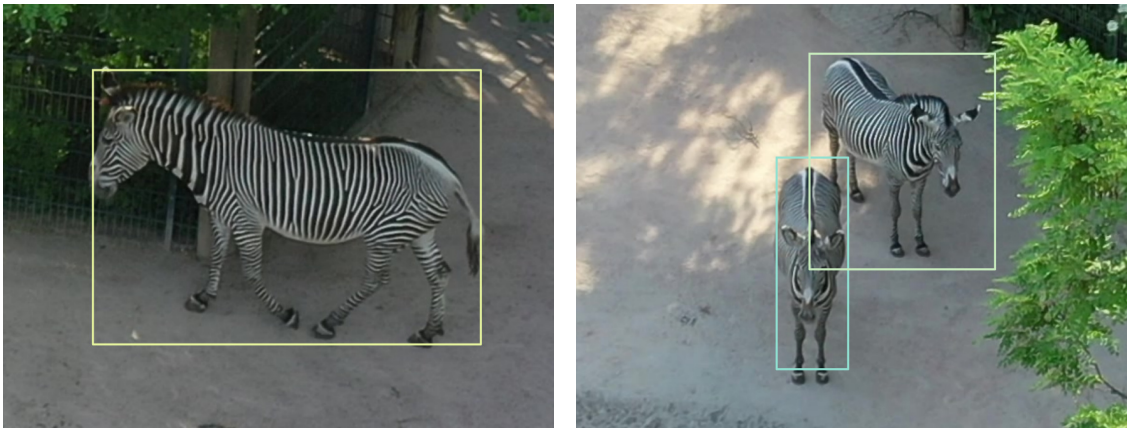


Figure 5.10: Two zoomed-in examples of imprecisely automatically labeled data from the R123 dataset. Specifically, the bounding boxes can either be slightly too loose or too tight on the zebras.

keeping videos in either of the sets to avoid overlap. Again, we filter from A36 **O**nly the **Z**ebras labels, obtaining $\mathbf{A36_{OZ}}$. Other zebras-only datasets we used are the **Zebra-300** and **Zebra-Zoo** [99], which have been used only for validation purposes. Zebra-300 contains 40 images from the AP-10K test set, 160 images from the AP-10K unlabeled set, and 100 from the Grévy's zebra [77] dataset according to [99]. To evaluate the generalization capabilities, we use the *horse* subset of the **T**ig**D**og [53] dataset, which we called **TDH**. We use the preprocessing from [119] to crop the images around the horses and get train and validation sets. We also create a filtered $\mathbf{TDH_{99}}$ 99-images subset. To train YOLOv5s we also use the images containing zebras belonging to the COCO [129] dataset. We used this in two variants $\mathbf{CZ}_{1920}$ and $\mathbf{CZ}_{640}$, signifying a different scaling factor of $1920 \times 1920$ and $640 \times 640$ respectively.

| | A10 | $A10_{OZ}$ [256] | $A10_{99}$ | A36 | $A36_{OZ}$ [252] | Zebra-300 [99] | Zebra-Zoo [99] | TDH | $TDH_{99}$ [53] | $CZ_{[1920,640]}$ [129] |
|---|---|---|---|---|---|---|---|---|---|---|
| Train | 7023 | 140 | 80 | 28457 | 960 | — | — | 8380 | 80 | 1916 |
| Valid | 995 | 20 | 19 | 7026 | 240 | 300 | 100 | 1772 | 19 | 85 |
| Test | 1997 | 40 | — | — | — | — | — | — | — | — |
| Animal | Various | Zebras | Zebras | Various | Zebras | Zebras | Zebras | Horses | Horses | Zebras |

Table 5.4: Summary of the real world datasets, containing for the most part common viewpoints, used in this thesis. The numbers indicate the number of images in each set. The type of animal(s) contained in the dataset is indicated in the last row. $CZ_{1920}$ and $CZ_{640}$ are used to indicate that the images of CZ, i.e. the zebras in the COCO dataset, are scaled to $640 \times 640$ and $1920 \times 1920$ pixels during YOLOv5 *training*.

### 5.5.2 Evaluation Metrics

The detection models are evaluated using the **mAP** and **mAP50**, i.e. the COCO standard average precision metric averaged over different IoU thresholds (mAP@[.5, .95]) and the PASCAL VOC's metric (mAP@.5). The pose estimation models are instead evaluated using the percentage of correct keypoints (PCK). The threshold for the PCK is set to 5% and 10% of the maximum size of the bounding box. We indicate those respectively $\mathbf{P}_{0.05}$ and $\mathbf{P}_{0.1}$. For all the metrics, the perfect result is 1.

## 5.6 Experiments

Here, we seek first to demonstrate that the synthetic data generated by our method can be used effectively for a vision-based task highly related to image features and context, such as the detection of zebras in outdoor wild environments from an aerial point of view. We hypothesize that, by training a model using only synthetic data acquired in a realistic simulation environment, we can achieve detection performance on real images comparable to a model trained on a manual and very precisely labeled set of real images. Our first goal is therefore proving that synthetically generated data *alone* can be used to train a network capable of detecting zebras with high accuracy in real world images in uncommon scenarios. Thereafter, we seek to improve the performance of the said detector, which does not generalize well to common images, to have a good overall average accuracy also when considering common images. After achieving that, we use the same synthetic data to train a 2D pose estimator. With those components, we seek to obtain a complete top-down system fully based on synthetic data, without relying on real world data or syn-to-real transfer techniques.

### 5.6.1 YOLOv5s Based Detection

All the YOLOv5s training runs are made from scratch with the default hyperparameters and for the standard 300 epochs. We do not introduce any additional data augmentation technique different from the one applied by default by the YOLOv5 code. This consists of some randomization in the scale, horizontal flip, translation, and HSV color space factors. We do not modify these values to have a fairer comparison across the models that would not require parameter grid searches or other steps when compared to the baseline pre-trained model. We save the best model, as evaluated on the specific validation set, and compare it over multiple datasets. With these comparisons, we demonstrate that, with our synthetic data, we can successfully capture real world features. This, while also obtaining trained models which show, in general, improved performance when compared to the pre-trained ones. Here, due to the limited availability of images, we use the *totality* of the $\text{A36}_{\text{OZ}}$ (1200 images) and $\text{A10}_{\text{OZ}}$ datasets (200 images). During these tests, we also noticed an overlap between $\text{A10}_{\text{OZ}}$ and CZ of at least nine images.

| Val. Set → | A36$_{GZ}$ | | R1_D1 | | R1_D2 | | R2_D1 | | R2_D2 | | R3_D1 | | R3_D2 | | RP (validation) | | W. avg. | | Avg. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Train Set ↓ | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP |
| SF$_{640}$ | 0.072 | 0.029 | 0.770 | 0.488 | 0.756 | 0.490 | 0.224 | 0.130 | 0.597 | 0.393 | 0.142 | 0.092 | 0.203 | 0.127 | 0.104 | 0.074 | 0.498 | 0.318 | 0.359 | 0.228 |
| SF | 0.103 | 0.055 | 0.853 | 0.568 | 0.958 | 0.646 | 0.873 | 0.540 | 0.957 | 0.616 | 0.608 | 0.443 | 0.651 | 0.484 | 0.287 | 0.191 | 0.853 | 0.563 | 0.661 | 0.443 |
| SC$_{640}$ | 0.121 | 0.046 | 0.714 | 0.455 | 0.830 | 0.529 | 0.147 | 0.084 | 0.513 | 0.316 | 0.156 | 0.097 | 0.375 | 0.202 | 0.092 | 0.061 | 0.482 | 0.299 | 0.369 | 0.224 |
| SC | 0.150 | 0.053 | 0.907 | 0.605 | 0.971 | 0.664 | 0.939 | 0.593 | 0.968 | 0.652 | 0.649 | 0.476 | 0.819 | 0.580 | 0.331 | 0.228 | 0.901 | 0.604 | 0.717 | 0.481 |
| RP$_{640}$ | 0.260 | 0.092 | 0.865 | 0.487 | 0.935 | 0.550 | 0.808 | 0.479 | 0.946 | 0.593 | 0.772 | 0.380 | 0.922 | 0.548 | 0.805 | 0.453 | 0.873 | 0.512 | 0.789 | 0.448 |
| RP | 0.161 | 0.066 | 0.937 | 0.615 | 0.980 | 0.663 | **0.989** | 0.653 | 0.982 | 0.666 | 0.801 | 0.532 | **0.986** | 0.680 | **0.914** | **0.636** | 0.950 | 0.636 | 0.844 | 0.564 |
| SC+CZ$_{1920}$ | 0.709 | 0.386 | 0.943 | 0.624 | 0.976 | 0.676 | 0.932 | 0.599 | 0.977 | 0.659 | 0.637 | 0.481 | 0.867 | 0.635 | 0.350 | 0.253 | 0.918 | 0.621 | 0.799 | 0.539 |
| RP+CZ$_{1920}$ | 0.837 | 0.526 | 0.967 | 0.639 | 0.984 | 0.681 | 0.980 | **0.656** | 0.968 | 0.684 | 0.768 | 0.493 | 0.981 | 0.674 | 0.911 | 0.626 | 0.956 | 0.650 | **0.925** | **0.622** |
| SC+CZ$_{1920}$+R3$_{100}$ | 0.704 | 0.378 | **0.975** | **0.655** | **0.994** | **0.707** | 0.963 | 0.636 | **0.991** | **0.691** | **0.986** | **0.733** | 0.961 | **0.708** | 0.432 | 0.308 | **0.975** | **0.676** | 0.878 | 0.602 |
| SC+CZ$_{1920}$+RP | 0.705 | 0.383 | 0.988 | 0.688 | 0.994 | 0.714 | 0.988 | 0.652 | 0.990 | 0.709 | 0.869 | 0.614 | 0.988 | 0.756 | 0.921 | 0.639 | 0.976 | 0.685 | 0.930 | 0.644 |
| Pretrained-COCO | **0.879** | **0.566** | 0.576 | 0.376 | 0.529 | 0.354 | 0.421 | 0.274 | 0.379 | 0.258 | 0.331 | 0.215 | 0.551 | 0.390 | 0.173 | 0.123 | 0.469 | 0.312 | 0.480 | 0.320 |

Table 5.5: Results of the evaluations of the trained YOLOv5s models, focusing on aerial images. We report mAP50 and mAP for each dataset, as well as both the average (Avg.) and weighted average (W. Avg., on the number of images) of these metrics. We set in bold the best results and underlined the best model **not** using the RP dataset during training in the RP validation column. For these evaluations, we use the *same* training and validation sizes by setting the `imgsz` option of YOLOv5.

### Detection — Aerial Images

Our baseline for comparison consists of the network pre-trained on the full COCO dataset. Following common practices, here we first train and evaluate the models using the *same* scaling option, i.e. models trained on $640 \times 640$ pixels are evaluated on the same image size. We perform training using both the default $640 \times 640$ image size and then increase that to $1920 \times 1920$ pixels. Note that all models trained with RP have been exposed to representing data coming from Rx_Dx, giving them an advantage in the corresponding evaluations.

All our results are reported in Table 5.5. Additionally, we present some qualitative samples using images from the CZ, A36, R2, and RP datasets for the main models in Figure 5.11. We immediately notice how the models trained on the bigger image size show better performance across all datasets and metrics. This is true for all synthetic, i.e. $SF_{640}$, SF, $SC_{640}$, SC, and real-world data, i.e. $RP_{640}$ and RP. The only exception is the model trained using $RP_{640}$ that performs $\sim 10\%$ better than RP when evaluated on the $A36_{OZ}$ data. At the same time, the model pre-trained on the COCO dataset works well only on the A36 dataset with a mAP50 of $\sim 88\%$, further showing the low variability of this data and the incapability to generalize to both different points of view or scenarios. Indeed, the YOLO model pre-trained on COCO achieves at most $\sim 58\%$ accuracy on our data, with an overall weighted average of $\sim 47\%$. The fact that the COCO data is representative of the $A36_{OZ}$ dataset is also manifested by the performance obtained by the model trained with RP+$CZ_{1920}$ dataset.

Considering now the synthetic data, i.e. SF and SC, we can see that the best model overall is SC by achieving $\sim 5\%$ higher mAP and mAP50 across all tests, with a peak of $\sim 15\%$ on the R3 dataset. This is probably related to the first of the two generation procedures, which resulted in long distances between the zebras and the cameras (Section 5.3.4). Our real-world data instead comprises mostly zebras that are reasonably "near" the drone, as seen from the pictures in Figure 5.9. Furthermore, the synthetic models may perform poorly on the RP validation set and $A36_{OZ}$ due to the generation process. These sets have diverse images, including zebras close to the camera in a side view or hidden behind bushes and trees, e.g. second and fourth rows in Figure 5.11. Finally, by comparing SC with the model pre-trained on COCO, we can see how across all data excluding $A36_{OZ}$, we obtain higher performances on both metrics of considerable amounts, ranging between $\sim 20\%$ and $\sim 45\%$.

We can then compare the differences between the models trained on synthetic data and real-world data. For this, we will focus on comparing SC and RP. The weighted average gap is only 4.9% in the mAP50 and 3.2% on the mAP. The big difference in the simple average is mostly linked to the results obtained in the validation set of RP, which was to be expected. Indeed, we can notice how the model trained on synthetic data performs considerably worse in the RP dataset, with a $\sim 58\%$ reduction in mAP50 and $\sim 41\%$ on mAP. A similar result is depicted when we consider tests on the R3_Dx data, with reductions of $\sim 16\%$ and $\sim 6-10\%$ for the two considered metrics.

(a) Ground-truth (b) COCO (c) RP (d) SC (e) SC+CZ$_{1920}$+R3$_{100}$

Figure 5.11: Sampled detections. We show in column (a) the ground truth and then the results obtained from (b) the default model (pre-trained on COCO), and the ones trained on (c) RP, (d) SC, and (e) SC+CZ$_{1920}$+R3$_{100}$. The images are randomly taken from the COCO (first row), A36, R2, and RP (last row) datasets.

Nonetheless, with all other datasets, the model trained on synthetic data is comparable to the one trained on real world captured images with differences of just about $1-5\%$. Recall that the RP model was trained on the RP dataset itself, composed of images from the Rx experiments and additional images from point-of-views not generated by our procedure. This clearly demonstrates that, on the considered datasets, the model trained solely with the synthetic data generated using the pipeline described above is perfectly capable of detecting zebras by achieving similar performance on all but two datasets when compared with RP, and significantly overcoming the model pre-trained on COCO in all but the A36 dataset.

After we establish that the bigger image size yields better performance and that SC is superior to SF, we train the network with mixed datasets. These are i) SC+$CZ_{1920}$, which combines SC training and validation sets with images from COCO's corresponding splits, ii) SC+$CZ_{1920}$+$R3_{100}$, which adds 100 train and 100 validation images randomly sampled from R3, and iii) RP+$CZ_{1920}$, which merges RP and CZ sets. Unsurprisingly, the model based only on real-world data, i.e. RP+$CZ_{1920}$, performs well on all datasets. The slight reduction in performance in the R2 and R3 datasets is well compensated for by the generalization obtained in A36. This is also the model with the highest average mAP and mAP50. We believe that this is mostly linked to how the dataset was built, with RP that contains data from all Rx experiments combined with the 1916 training images of COCO. Despite that, it is interesting to notice how the models trained with a mixture of synthetic and real-world data are generalizing across all the datasets as well. Specifically, combining SC and COCO, i.e. SC+$CZ_{1920}$, resulted in a significant improvement of the performance solely in the A36 dataset, with minor ones in other datasets as well. If to this data we add the 100 samples from R3, i.e. SC+$CZ_{1920}$+$R3_{100}$, we achieve considerable improvements with respect to SC on all datasets.

The most noticeable are the ones on A36, of around 55%, and on the RP dataset, of around 10%. The improvement on the R3 data is to be expected since we mixed 100 images from that set. Nonetheless, it is remarkable that just a small change in the data brought a $\sim 34\%$ increase in the mAP for this validation test. The SC+$CZ_{1920}$+$R3_{100}$ is the model with the highest weighted average precisions and is the second best when considering the average mAP and mAP50. We believe that this model would be further improved by having more samples from the COCO dataset in the validation set or, overall, a better-balanced set of samples. Considering that SC is made of 18K images, and both COCO and $R3_{100}$ make up for 2K training images and only 300 validation ones, we can expect an 'overfit' of the final selected model towards scenes which are strongly represented by the synthetic images. Also, in this case, the significant difference in the average mAP and mAP50 is mostly linked to the gap in the results in the RP validation set. For completeness, we also trained the SC+$CZ_{1920}$+RP model, i.e. using the closeby synthetic data, the coco data, and the small set of real-world data which was precisely labeled. As expected, this is the model that performs best in the majority of the tests, excluding the A36 dataset, where the pre-trained model performs best, and in R3_D1. However, we must note that RP contains data from all R1, R2, and R3 datasets in both

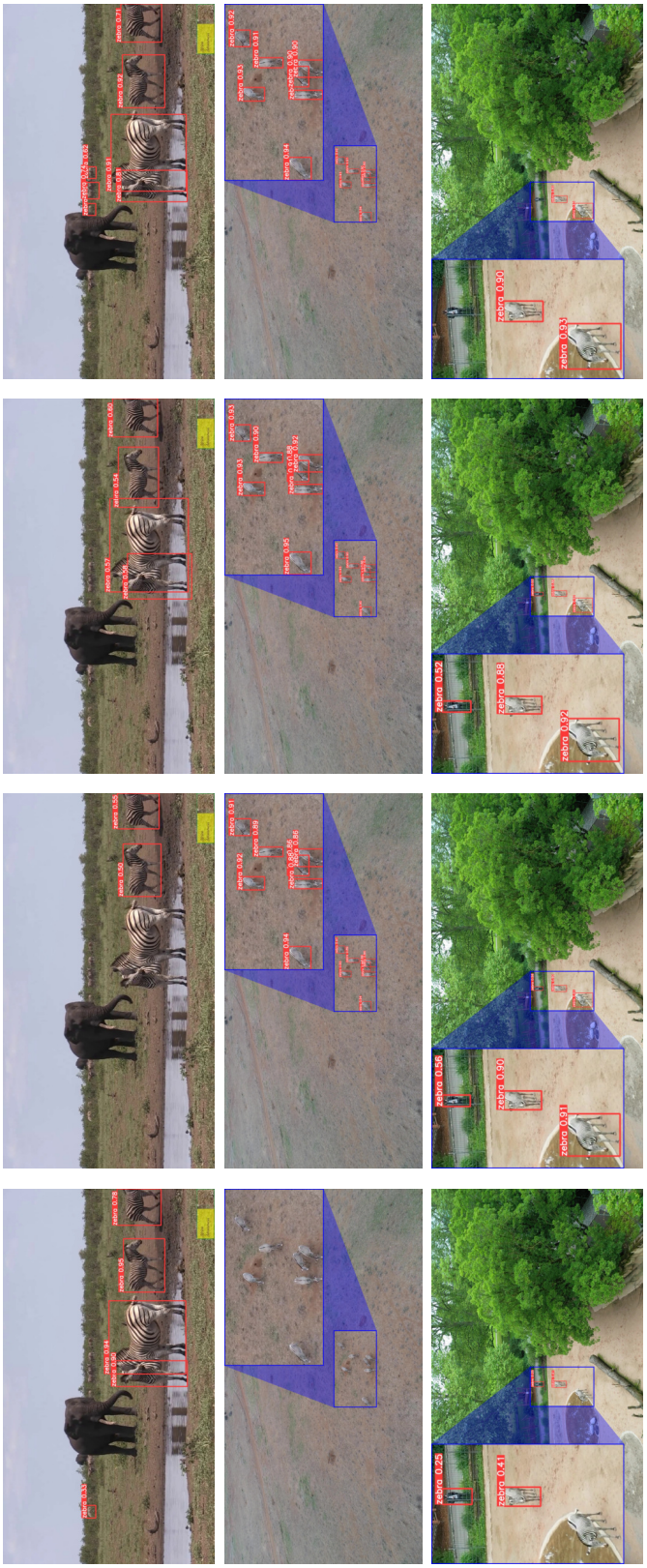
the training and validation sets. Thus, the results in these cases are clearly driven by this information. What is interesting to notice is that all mixed models perform similarly in the A36 dataset, with $\sim 70\%$ of mAP50 and $\sim 38\%$ mAP, further indicating that a better balancing in the validation set might further boost the performance of these models. Alternatively, a more representative generation strategy could be employed by including camera locations relative to the zebras more similar to the ones that we can find in the A36 or the COCO dataset. The results suggest that such an approach would be effective as well, perhaps in conjunction with a minimal amount of annotated real-world data. Finally, considering that zebra stripes are notoriously specific to the individual, it is interesting to notice how, even though we use the same texture for all our generated zebras, we are still able to generalize to different individuals well. This suggests that the network does not focus and learn specifically the pattern it is shown, but rather the general appearance of the animal itself. In the next section, we will address this limitation by leveraging the $SC_{5K}$ dataset we introduced in Section 5.3.5.

| Val. Set → | $A36_{OZ}$ | | $A10_{OZ}$ | | RP | | R123 | | Average | | W. Avg. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Train Set ↓ | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP |
| SC | 0.613 | 0.343 | 0.455 | 0.245 | 0.158 | 0.116 | 0.581 | 0.382 | 0.452 | 0.271 | 0.580 | 0.380 |
| $SC_{5K}$ | 0.676 | 0.361 | 0.583 | 0.318 | 0.164 | 0.119 | 0.605 | 0.408 | 0.507 | 0.302 | 0.605 | 0.407 |
| SC+RP | 0.690 | 0.414 | 0.552 | 0.315 | **0.471** | **0.256** | 0.734 | 0.502 | 0.612 | 0.372 | 0.732 | 0.500 |
| $SC_{5K}$+RP | 0.753 | 0.434 | 0.613 | 0.352 | 0.464 | 0.254 | 0.769 | 0.520 | 0.650 | 0.390 | 0.768 | 0.518 |
| SC+$CZ_{1920}$ | 0.827 | 0.495 | 0.889 | 0.601 | 0.163 | 0.119 | 0.565 | 0.375 | 0.611 | 0.398 | 0.568 | 0.377 |
| $SC_{5K}$+$CZ_{1920}$ | 0.846 | 0.521 | 0.917 | 0.632 | 0.171 | 0.124 | 0.646 | 0.436 | 0.645 | 0.428 | 0.648 | 0.437 |
| SC+$CZ_{1920}$+RP | 0.861 | 0.534 | 0.894 | 0.603 | 0.447 | 0.244 | 0.731 | 0.500 | 0.733 | 0.470 | 0.732 | 0.500 |
| $SC_{5K}$+$CZ_{1920}$+RP | 0.872 | 0.545 | **0.911** | 0.631 | 0.459 | 0.247 | **0.770** | **0.524** | **0.753** | **0.487** | **0.771** | **0.524** |
| $CZ_{1920}$ | 0.868 | 0.524 | 0.896 | 0.621 | 0.081 | 0.044 | 0.146 | 0.072 | 0.498 | 0.315 | 0.155 | 0.078 |
| $CZ_{640}$ | **0.888** | **0.575** | 0.903 | **0.646** | 0.208 | 0.135 | 0.411 | 0.249 | 0.602 | 0.401 | 0.417 | 0.253 |

Table 5.6: YOLOv5s models validated using images scaled to $640 \times 640$ pixels. We report mAP50 and mAP for each validation set (columns), compute the average and the weighted average (on the number of images), and bold the best metrics.

| Val. Set → | $A36_{OZ}$ | | $A10_{OZ}$ | | RP | | R123 | | Average | | W. Avg. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Train Set ↓ | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP | mAP50 | mAP |
| SC | 0.150 | 0.053 | 0.076 | 0.021 | 0.331 | 0.228 | 0.911 | 0.611 | 0.367 | 0.228 | 0.899 | 0.603 |
| $SC_{5K}$ | 0.512 | 0.197 | 0.268 | 0.094 | 0.340 | 0.236 | 0.935 | 0.623 | 0.514 | 0.288 | 0.928 | 0.617 |
| SC+RP | 0.207 | 0.102 | 0.084 | 0.030 | 0.903 | 0.638 | 0.980 | 0.683 | 0.544 | 0.363 | 0.969 | 0.675 |
| $SC_{5K}$+RP | 0.580 | 0.243 | 0.309 | 0.106 | 0.910 | 0.639 | 0.975 | 0.680 | 0.693 | 0.417 | 0.969 | 0.674 |
| SC+$CZ_{1920}$ | 0.709 | 0.386 | 0.466 | 0.234 | 0.350 | 0.253 | 0.922 | 0.625 | 0.612 | 0.374 | 0.918 | 0.621 |
| $SC_{5K}$+$CZ_{1920}$ | 0.740 | 0.447 | 0.517 | 0.277 | 0.370 | 0.268 | 0.940 | 0.638 | 0.642 | 0.407 | 0.936 | 0.634 |
| SC+$CZ_{1920}$+RP | 0.705 | 0.383 | 0.460 | 0.228 | 0.921 | 0.639 | 0.980 | **0.689** | 0.766 | 0.485 | 0.975 | **0.685** |
| $SC_{5K}$+$CZ_{1920}$+RP | 0.752 | 0.434 | 0.442 | 0.237 | **0.926** | **0.665** | **0.982** | 0.688 | **0.775** | **0.506** | **0.978** | 0.684 |
| $CZ_{1920}$ | **0.839** | **0.514** | **0.740** | **0.432** | 0.244 | 0.149 | 0.644 | 0.343 | 0.617 | 0.360 | 0.646 | 0.345 |
| $CZ_{640}$ | 0.496 | 0.269 | 0.220 | 0.116 | 0.441 | 0.311 | 0.856 | 0.561 | 0.503 | 0.314 | 0.850 | 0.556 |

Table 5.7: YOLOv5s models validated using images scaled to $1920 \times 1920$ pixels. We report mAP50 and mAP for each validation set (columns), compute the average and the weighted average (on the number of images), and bold the best metrics.



(a) $CZ_{1920}$ (b) SC (c) $SC_{5K}$ (d) $SC_{5K}$+$CZ_{1920}$+RP

Figure 5.12: YOLOv5s results on images from the A36 (top row), from [166] (mid row), and our R123 (bottom row) datasets, using $1920 \times 1920$ resolution.

### Detection — Generalization to Common Viewpoints

Although it is common practice to test models on images resized to the training resolution, we evaluate all models at both $640 \times 640$ (Table 5.6) and $1920 \times 1920$ (Table 5.7). In these experiments, we include $CZ_{640}$, which represents a YOLOv5s model trained using the zebras images from the COCO dataset using a $640 \times 640$ resolution.

We can see from Table 5.7 how models trained with the new synthetic dataset, $SC_{5K}$, consistently outperform those trained on the previous SC dataset. For instance, on $A36_{OZ}$, training with $SC_{5K}$ yields a +36% improvement in mAP50 compared to SC. On average, when YOLOv5s is trained solely on $SC_{5K}$, its performance matches or exceeds that of models trained exclusively on $CZ_{640}$, except on RP. Furthermore, the same model significantly outperforms $CZ_{1920}$ in both RP and R123, particularly on aerial images.

The results obtained using images scaled to $640 \times 640$, shown in Table 5.6, reinforce these findings. Models trained with synthetic data continue to perform well on average, while those trained solely on COCO struggle on R123 and RP, further highlighting the versatility of our synthetic data. Notably, the evaluation resolution has a significant impact on performance. This can be observed in the differences between the results of $CZ_{1920}$ and $CZ_{640}$ when evaluated at their corresponding or different validation image sizes. For instance, when tested on $1920 \times 1920$ images, $CZ_{640}$ outperforms $CZ_{1920}$ on RP and R123, while $CZ_{1920}$ achieves better results on $A36_{OZ}$ and $A10_{OZ}$. Additionally, performance on aerial images (R123, RP) degrades when the input is downscaled to $640 \times 640$ pixels, whereas models perform better on 'common' viewpoints under the same conditions (and vice versa). This is likely due to the relative size of individuals observed during training and validation. Specifically, large images, such as those from R123, when resized to $640 \times 640$, further reduce the size of objects, making detection more challenging—especially for models trained predominantly on high-resolution data (i.e. all except $CZ_{640}$). Conversely, not upscaling smaller images like those in A36OZ or $A10_{OZ}$ to $1920 \times 1920$ preserves the small object scale, which benefits models trained on larger images by a considerable margin. Therefore, there is no clear evidence that matching training and validation resolutions always lead to better performance; instead, the optimal resolution depends on the distribution of object sizes.

As expected, the best-performing models are those trained with a combination of RP, $CZ_{1920}$, and $SC_{5K}$. However, no single model achieves consistently strong performance across all datasets and scaling sizes. Nonetheless, the results clearly demonstrate that synthetic data alone can yield competitive performance, often matching or surpassing models trained solely on real-world data in both average and weighted mAP50 and mAP scores. Furthermore, our analysis highlights the importance of comprehensive testing and diverse training data. The significant performance drop on $A10_{OZ}$ compared to $A36_{OZ}$ (8–25%) suggests that detection performance cannot be assumed even when using real-world data. This reinforces the need for more robust benchmarks and diverse training samples to mitigate dataset biases.

Finally, the qualitative comparison in Figure 5.12 illustrates how the $SC_{5K}$ model im-

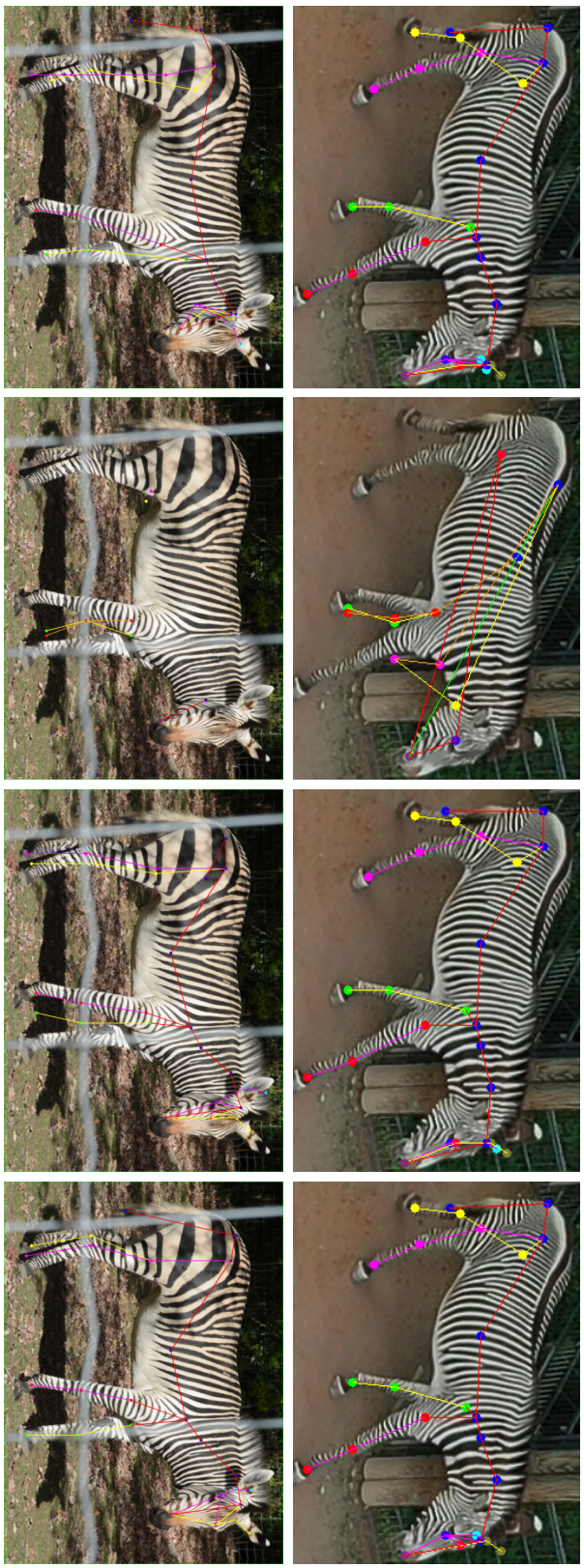
proves over the standard SC on the A36 dataset (top row, column (c)). In contrast, the baseline model struggles with aerial images (central and bottom rows, column (a)). As expected, the mixed model (last column) demonstrates the best overall performance. Additionally, when compared with Figure 5.4, we observe that zebras detected in the background of the A36 image do not contribute to accuracy due to mislabeling, whereas the correctly not labeled elephant lowers the computed accuracy, affecting the final results. This suggests that incorporating real-world data as validation for synthetic-based training could further enhance performance.

| | Only Zebras | | | | | | | | Various animals | | | | Horses | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Val. Set → | Zebra-300 | | $A10_{OZ}$ | | $A36_{OZ}$ | | Zebra-Zoo | | A10 | | A36 | | TDH | |
| Train Set ↓ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ |
| $A10_{OZ}$ | 0.372 | 0.621 | 0.353 | 0.569 | 0.252 | 0.507 | 0.353 | 0.625 | 0.084 | 0.191 | 0.079 | 0.180 | 0.288 | 0.521 |
| $A36_{OZ}$ | 0.378 | 0.608 | 0.358 | 0.552 | 0.328 | 0.544 | 0.251 | 0.476 | 0.055 | 0.133 | 0.062 | 0.143 | 0.165 | 0.345 |
| SpacNet | 0.207 | 0.411 | 0.275 | 0.465 | 0.245 | 0.419 | 0.139 | 0.327 | 0.050 | 0.122 | 0.042 | 0.106 | 0.086 | 0.227 |
| A10 | 0.685 | 0.889 | 0.629 | 0.823 | 0.532 | 0.768 | 0.664 | 0.876 | 0.516 | 0.718 | 0.465 | 0.661 | 0.596 | 0.853 |
| A36 | 0.691 | 0.883 | 0.602 | 0.783 | 0.585 | 0.777 | 0.495 | 0.774 | 0.366 | 0.557 | 0.531 | 0.722 | 0.638 | 0.875 |
| TDH | 0.080 | 0.192 | 0.059 | 0.158 | 0.039 | 0.126 | 0.050 | 0.160 | 0.057 | 0.141 | 0.064 | 0.156 | 0.949 | 0.974 |
| SC | 0.466 | 0.686 | 0.292 | 0.511 | 0.273 | 0.447 | 0.395 | 0.600 | 0.082 | 0.178 | 0.095 | 0.200 | 0.294 | 0.475 |
| *—filtered* | 0.608 | 0.790 | 0.345 | 0.573 | 0.340 | 0.508 | 0.522 | 0.707 | 0.090 | 0.184 | 0.106 | 0.207 | 0.324 | 0.505 |
| SC+$A10_{OZ}$ | 0.847 | 0.967 | 0.675 | 0.879 | 0.705 | 0.878 | 0.819 | 0.945 | 0.165 | 0.303 | 0.169 | 0.308 | 0.434 | 0.659 |
| *—filtered* | 0.864 | 0.966 | 0.657 | 0.869 | 0.716 | 0.867 | 0.796 | 0.932 | 0.174 | 0.307 | 0.180 | 0.315 | 0.473 | 0.681 |
| SC+$A36_{OZ}$ | 0.844 | 0.961 | 0.657 | 0.848 | 0.768 | 0.918 | 0.855 | 0.970 | 0.190 | 0.342 | 0.243 | 0.410 | 0.546 | 0.777 |
| *—filtered* | 0.858 | 0.956 | 0.651 | 0.831 | 0.779 | 0.920 | 0.851 | 0.963 | 0.198 | 0.341 | 0.257 | 0.415 | 0.568 | 0.782 |
| SC+$A10_{99}$ | 0.750 | 0.938 | 0.585 | 0.803 | 0.605 | 0.832 | 0.737 | 0.928 | 0.119 | 0.231 | 0.131 | 0.252 | 0.410 | 0.626 |
| *—filtered* | 0.828 | 0.950 | 0.591 | 0.809 | 0.636 | 0.824 | 0.750 | 0.918 | 0.129 | 0.236 | 0.144 | 0.259 | 0.443 | 0.644 |
| SpacNet+$A10_{99}$ | 0.525 | 0.777 | 0.462 | 0.674 | 0.437 | 0.676 | 0.482 | 0.768 | 0.091 | 0.200 | 0.080 | 0.179 | 0.199 | 0.408 |
| SC+$TDH_{99}$ | 0.490 | 0.707 | 0.321 | 0.528 | 0.336 | 0.538 | 0.396 | 0.580 | 0.128 | 0.260 | 0.155 | 0.303 | 0.658 | 0.873 |
| *—filtered* | 0.634 | 0.813 | 0.373 | 0.591 | 0.421 | 0.603 | 0.533 | 0.688 | 0.146 | 0.274 | 0.177 | 0.321 | 0.685 | 0.872 |
| SC+A10 | *0.891* | *0.977* | **0.714** | **0.894** | 0.756 | 0.911 | 0.882 | 0.977 | *0.643* | *0.825* | 0.626 | 0.798 | 0.717 | 0.918 |
| *—filtered* | **0.898** | *0.973* | *0.699* | *0.889* | 0.777 | 0.909 | 0.881 | 0.971 | **0.673** | **0.829** | 0.650 | 0.806 | 0.754 | 0.912 |
| SC+A36 | 0.861 | 0.965 | 0.697 | 0.867 | *0.787* | *0.936* | **0.908** | **0.985** | 0.522 | 0.709 | *0.678* | *0.840* | 0.729 | 0.923 |
| *—filtered* | 0.868 | 0.960 | 0.696 | 0.867 | **0.797** | **0.938** | *0.900* | *0.983* | 0.552 | 0.719 | **0.703** | **0.849** | 0.763 | 0.916 |
| SC+TDH | 0.524 | 0.754 | 0.374 | 0.595 | 0.334 | 0.555 | 0.465 | 0.714 | 0.146 | 0.295 | 0.189 | 0.379 | **0.969** | **0.989** |
| *—filtered* | 0.668 | 0.838 | 0.432 | 0.657 | 0.402 | 0.606 | 0.588 | 0.807 | 0.164 | 0.307 | 0.208 | 0.394 | *0.965* | *0.986* |

Table 5.8: PCK@0.05 and PCK@0.1 of the ViTPose+ models trained on the specified dataset (row) with all **weights randomly initialized** and evaluated on the specified data (columns). We put in bold the best results, and in italics the second best.

### 5.6.2 2D pose estimation performance

We employ the popular ViTPose architecture in its ViTPose+ [248] variation, following standard training settings: 210 epochs with decay steps at 170 and 200, Adam optimizer with a 5e−4 learning rate, and Gaussian heatmaps as the target type. Ground truth bounding boxes are used during both training and validation. We train the *small* network model using 17 keypoints for real-world datasets and 27 for synthetic or mixed ones.



(a) SC (b) SC+A10$_{OZ}$ (c) SC+A10$_{99}$ (d) SC+A10

Figure 5.13: ViTPose+ trained on the specified dataset using a randomly-initialized backbone and run on one of the images from the R123 (first row) and Zebra-zoo (second row) datasets, manually cropped around the zebra *after* inference.

Additionally, due to minor annotation misalignments in our synthetic data (specifically, the thighs and tail base differing by a few pixels), we also report a *filtered* average result that excludes these keypoints. Training is conducted both from scratch and with a default MAE pre-trained backbone.

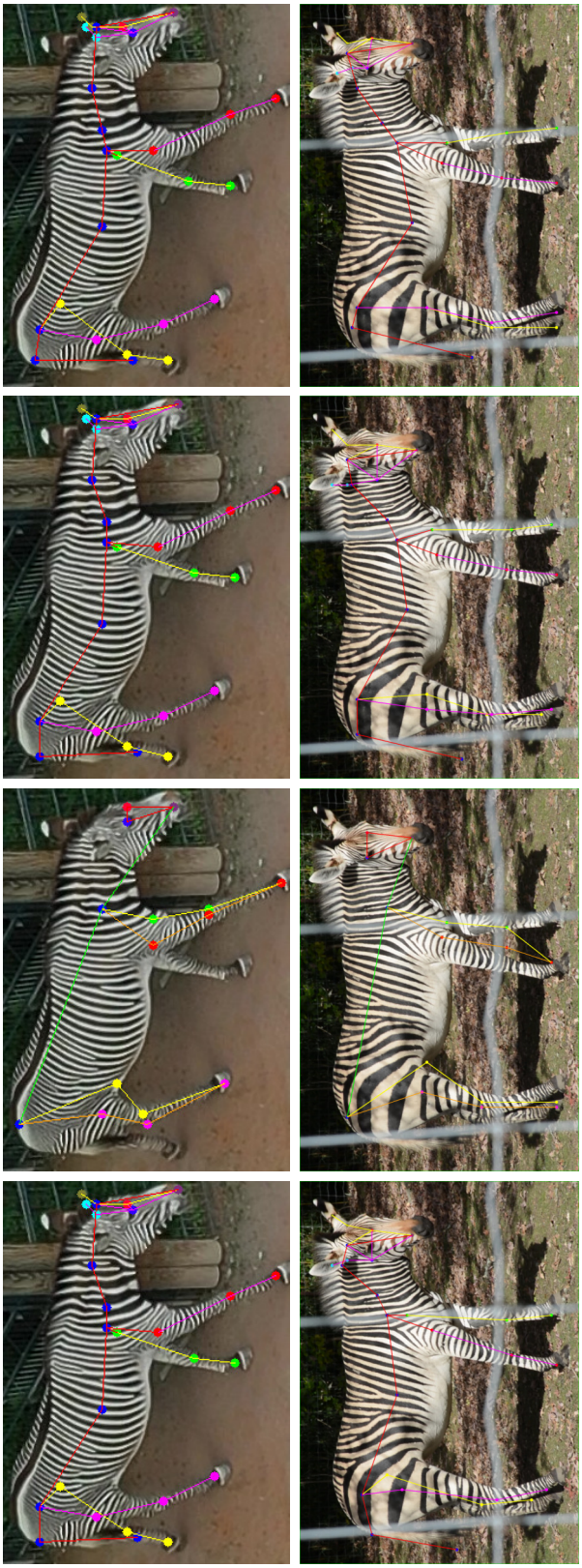
First, in Table 5.8, we analyze results obtained without pre-training. The model trained only on SC achieves performance comparable to those trained solely on zebra datasets ($A36_{OZ}$ or $A10_{OZ}$) for both PCK metrics. However, it demonstrates better generalization on horses (TDH) and across the full A10 and A36 datasets, with improvements exceeding 13%. A similar trend is observed when comparing SC to SpacNet, with gains ranging from 3% to 25%. Moreover, the *filtered* SC model significantly outperforms those trained on $A36_{OZ}$ and $A10_{OZ}$. Despite SC's limited pose diversity due to the constrained number of animated frames, it effectively trains a 2D pose estimator for real images. As expected, training on the full A10 and A36 datasets achieves the highest performance among single-dataset models, benefiting from domain transfer between similar species and diverse animal instances. However, when comparing Zebra-300 and Zebra-Zoo, the *filtered* SC model performs similarly or better than models trained on A10 and A36. Additionally, SC generalizes better to TDH horses than TDH data does to zebras. Integrating real-world data further enhances performance, even with minimal additions. For instance, SC+$A10_{99}$ significantly outperforms SpacNet+$A10_{99}$ in both zebra keypoint detection and generalization, with PCK improvements of up to 20%. Similarly, SC+$TDH_{99}$ achieves TDH performance comparable to models trained on the full A10 or A36 datasets. Finally, combining SC with the entirety of A10 or A36 yields minimal gains, except for improvements on non-zebra datasets. In Table 5.9, we show how using a pre-trained backbone generally enhances performance across all models, datasets, and metrics by leveraging real-world information. Training with $A36_{OZ}$ or $A10_{OZ}$ achieves PCK@0.05 values above 60% on all zebra-related validation sets. Notably, $A36_{OZ}$ consistently outperforms $A10_{OZ}$ on those sets but exhibits poorer generalization to the full A10, A36, and TDH datasets.

Using SpacNet data results in a notable drop in performance, with PCK@0.05 ranging from 42% to 53% and PCK@0.1 between 64% and 79%. In contrast, training with SC data achieves similar results to SpacNet, despite not employing any augmentation or style transfer techniques. The SC-*filtered* results, which account for keypoint misalignment, significantly improve PCK@0.05 in the Zebra-300 and Zebra-Zoo datasets by approximately 15–31% over SpacNet, demonstrating the superiority of our synthetic data. While SC-*filtered* performance is still about 15% lower than $A36_{OZ}$, it closely matches $A10_{OZ}$, both of which contain only real zebras. Similar to the non-pre-trained case, SC data exhibits strong generalization on horses, achieving a 48% PCK@0.1. Performance improves further as real-world data is incorporated, with mixed datasets consistently outperforming single ones. Interestingly, when comparing pre-trained and non-pre-trained models (Table 5.9 vs. Table 5.8), we observe that SC alone reduces the need for pre-training. Specifically, training with just SC+$A10_{99}$ (i.e. SC plus 99 images from A10) achieves results comparable to the best model trained with a pre-trained backbone.

Qualitative analysis shows how models trained on synthetic data, Figures 5.13 to 5.15, achieve good overall performance. Moreover, we can notice that most errors stem from swapped limbs or incorrect leg associations. These errors can be attributed to differences in data generation and annotation practices. In our synthetic dataset, keypoints are extracted directly from the surface of the animal, and bounding boxes are pixel-tight. In contrast, human annotators typically place keypoints where they estimate the limb to be, and bounding boxes may not be tightly cropped. Additionally, synthetic zebras sometimes intersect with the terrain, complicating the learning of hoof positions. Interestingly, one of the best-performing models, which was trained on A10 with a pre-trained backbone, still struggles with keypoint estimation in aerial views. In contrast, the model trained on synthetic data without pre-training performs well across all real-world images. Finally, the primary challenge in generalizing to horses can be identified in their uniform coloring, which lacks distinctive features. Additionally, unusual cropping confuses the model in determining the correct side of the animal and keypoint placement.

| | Zebras | | | | | | | | Various animals | | | | Horses | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Val. Set → | Zebra-300 | | $A10_{OZ}$ | | $A36_{OZ}$ | | Zebra-Zoo | | A10 | | A36 | | TDH | |
| Train Set ↓ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ | $P_{0.05}$ | $P_{0.1}$ |
| $A10_{OZ}$ | 0.753 | 0.902 | 0.673 | 0.815 | 0.654 | 0.833 | 0.740 | 0.887 | 0.264 | 0.453 | 0.243 | 0.414 | 0.463 | 0.742 |
| $A36_{OZ}$ | 0.849 | 0.954 | 0.690 | 0.833 | 0.778 | 0.923 | 0.865 | 0.962 | 0.230 | 0.410 | 0.271 | 0.458 | 0.422 | 0.738 |
| SpacNet | 0.533 | 0.792 | 0.441 | 0.644 | 0.434 | 0.655 | 0.419 | 0.724 | 0.111 | 0.235 | 0.092 | 0.202 | 0.198 | 0.414 |
| A10 | 0.888 | 0.979 | **0.779** | *0.900* | 0.789 | 0.918 | 0.892 | 0.980 | **0.710** | **0.868** | 0.672 | 0.822 | 0.707 | 0.908 |
| A36 | 0.880 | 0.968 | *0.747* | 0.870 | 0.805 | 0.936 | 0.902 | 0.984 | 0.579 | 0.759 | 0.712 | 0.867 | 0.724 | 0.924 |
| TDH | 0.145 | 0.291 | 0.117 | 0.273 | 0.174 | 0.337 | 0.086 | 0.223 | 0.138 | 0.281 | 0.164 | 0.330 | *0.965* | *0.986* |
| SC | 0.527 | 0.736 | 0.363 | 0.569 | 0.324 | 0.514 | 0.525 | 0.730 | 0.083 | 0.177 | 0.099 | 0.204 | 0.276 | 0.450 |
| *—filtered* | 0.689 | 0.859 | 0.434 | 0.647 | 0.409 | 0.588 | 0.720 | 0.875 | 0.093 | 0.183 | 0.112 | 0.212 | 0.304 | 0.475 |
| SC+$A10_{OZ}$ | 0.864 | 0.981 | 0.671 | 0.866 | 0.742 | 0.905 | 0.905 | 0.979 | 0.151 | 0.279 | 0.160 | 0.285 | 0.359 | 0.559 |
| *—filtered* | 0.899 | 0.981 | 0.665 | 0.865 | 0.771 | 0.906 | 0.911 | 0.974 | 0.162 | 0.282 | 0.173 | 0.291 | 0.397 | 0.581 |
| SC+$A36_{OZ}$ | 0.878 | 0.972 | 0.687 | 0.874 | 0.806 | 0.947 | 0.935 | 0.990 | 0.215 | 0.372 | 0.275 | 0.447 | 0.534 | 0.783 |
| *—filtered* | 0.891 | 0.970 | 0.675 | 0.862 | *0.823* | 0.950 | *0.942* | 0.988 | 0.221 | 0.369 | 0.288 | 0.451 | 0.556 | 0.781 |
| SC+$A10_{99}$ | 0.757 | 0.941 | 0.600 | 0.828 | 0.629 | 0.844 | 0.813 | 0.957 | 0.119 | 0.234 | 0.136 | 0.258 | 0.343 | 0.536 |
| *—filtered* | 0.863 | 0.960 | 0.621 | 0.830 | 0.677 | 0.856 | 0.876 | 0.962 | 0.128 | 0.238 | 0.150 | 0.265 | 0.378 | 0.560 |
| SpacNet+$A10_{99}$ | 0.796 | 0.931 | 0.636 | 0.778 | 0.624 | 0.814 | 0.743 | 0.925 | 0.190 | 0.353 | 0.149 | 0.296 | 0.289 | 0.565 |
| SC+$TDH_{99}$ | 0.564 | 0.773 | 0.385 | 0.603 | 0.345 | 0.545 | 0.547 | 0.731 | 0.136 | 0.268 | 0.172 | 0.324 | 0.678 | 0.884 |
| *—filtered* | 0.745 | 0.901 | 0.456 | 0.696 | 0.440 | 0.624 | 0.731 | 0.875 | 0.154 | 0.283 | 0.195 | 0.339 | 0.707 | 0.878 |
| SC+A10 | *0.906* | **0.984** | 0.728 | *0.900* | 0.782 | 0.916 | 0.932 | **0.993** | 0.670 | 0.842 | 0.657 | 0.819 | 0.732 | 0.922 |
| *—filtered* | **0.916** | *0.982* | 0.715 | **0.901** | 0.817 | 0.922 | **0.944** | *0.992* | *0.703* | *0.849* | 0.684 | 0.829 | 0.774 | 0.917 |
| SC+A36 | 0.890 | 0.976 | 0.724 | 0.897 | 0.820 | *0.953* | 0.934 | **0.993** | 0.575 | 0.757 | *0.719* | *0.871* | 0.760 | 0.936 |
| *—filtered* | 0.905 | 0.973 | 0.707 | 0.893 | **0.845** | **0.958** | 0.938 | *0.992* | 0.610 | 0.771 | **0.750** | **0.882** | 0.800 | 0.934 |
| SC+TDH | 0.586 | 0.800 | 0.412 | 0.624 | 0.384 | 0.598 | 0.606 | 0.801 | 0.166 | 0.321 | 0.221 | 0.417 | **0.967** | **0.989** |
| *—filtered* | 0.754 | 0.902 | 0.489 | 0.710 | 0.477 | 0.677 | 0.793 | 0.920 | 0.190 | 0.338 | 0.242 | 0.435 | 0.963 | *0.986* |

Table 5.9: PCK@0.05 and PCK@0.1 of the ViTPose+ models trained on the specified dataset (row) using a **MAE pre-trained backbone** and evaluated on the specified data (columns). We put in bold the best results, and in italics the second best.



(a) SC (b) A10$_{OZ}$ (c) SC+A10$_{99}$ (d) SC+A10

Figure 5.14: ViTPose+ trained on the specified dataset using a MAE pre-trained backbone and run on one of the images from the R123 (first row) and Zebra-zoo (second row) datasets, manually cropped around the zebra *after* inference.

Figure 5.15: ViTPose+ trained on the specified dataset using a randomly-initialized backbone (first row) and an MAE pre-trained backbone (second row), and run on one of the images from the TDH dataset shown as *processed* per dataset specifics.

## 5.7 Conclusions

In this chapter, we first demonstrated that the currently available datasets do not generalize well to the task of detecting zebras captured from an aerial point of view. To solve this, we generated a large-scale synthetic dataset of zebras by using GRADE, a state-of-the-art framework for synthetic data generation. The dataset, which is the first of its kind both in terms of size and visual realism, has been released for the benefit of the community. By using that, we performed extensive evaluations by training and testing YOLO with a wide range of combinations of real and synthetic data. This provides strong evidence that the visual realism of the data generated is very high since our models showed performances that are as good as the one obtained by a detector trained on real world labeled data. Using synthetic information, we can thus surpass the process of collecting and labeling data in controlled scenarios, avoiding the probable introduction of errors. Further testing by using combined synthetic and a small amount of real-world data showed that we can successfully generalize to a wide variety of scenarios. A known limitation that we need to address is the realism of the adopted environments and more precise placement strategies, which we believe could solve both the generalization problem and the usage of high-resolution images by the network.

In this chapter, we also show how *only* synthetic data can be used to successfully train *both* a detector and a 2D pose estimation network without the use of style transfer techniques and potentially without real world data. Besides the results in the detection task, we show remarkable performance in estimating the pose of zebras despite the low number of animations. Finally, the inclusion of just 99 real images improves performance and generalization capabilities up to comparable baseline methods. The results thus suggest that our generation strategy, which is focused on visual realism and variability, successfully addresses the problem even when not using any pre-trained network or complex animal model. Through our extensive validation, we also highlight how i) using the same image sizes for training and testing with YOLO is not always effective, despite being common practice, ii) synthetic datasets can overcome the need for pre-trained backbones, and iii) how extensive testing with diverse data is fundamental.

# Chapter 6

# Conclusion

## 6.1 Summary of Contributions

In this thesis, we focus on tightly integrating visual perception with action. Our contributions span the development of novel algorithms for Active Visual SLAM, the creation of a high-fidelity simulation framework, and the design of dynamic scene understanding methods—all of which contribute to more robust, flexible, and efficient autonomous systems.

We begin in Chapter 2 with the design and implementation of a new autonomous exploration approach, iRotate. Traditional methods often treat Active SLAM as a "static" optimization problem, focusing solely on either short-term maneuvering or long-term path planning. These approaches fail to account for the fact that available information *continuously* changes, analyzing it only at "fixed" intervals. Rather than employing a human-like approach of ever-switching focus, classical methods fail to adapt their behavior once a course of action is chosen until the next decision point. In contrast, iRotate predicts and adapts to information gain dynamically and continuously. As we explain in Section 2.5, we do so by optimizing camera viewpoints at three distinct levels: a long-term layer that pre-optimizes paths toward global frontiers, a mid-level that refines camera orientations based on local uncertainty while the robot reaches subsequent waypoints, and a real-time layer that enhances pose estimation using local visual features. This is the first multi-layered strategy that allows the robot to continuously adjust its view independently of its physical trajectory and that effectively integrates long-term, short-term, and real-time analysis of both its state, the map being built, and its surroundings. Furthermore, as we show in Section 2.6, iRotate can be applied to omnidirectional as well as non-holonomic robots thanks to the independent camera rotational joint we introduced. This joint allows the decoupled control of the robot and camera orientations and requires a new state estimation technique to jointly model the robot and camera states (Section 2.4.3). Extensive experiments, both in simulated and real worlds, demonstrate that iRotate reduces localization errors, explores larger environments with shorter trajectories, and minimizes overall energy consumption - all at the same time.

iRotate, like several other V-SLAM methods, was developed under the assumption of a static environment. However, static environments fail to capture the complexities of

real-world dynamics, hindering the widespread deployment of autonomous robots in everyday applications. Real-world scenarios involve moving objects, dynamic lighting, and changing scene conditions, all of which significantly impact the performance of robotic perception and decision-making systems. This assumption is widespread due to the dangers and costs associated with directly testing robots in real-world settings, the difficulties of collecting precise ground-truth data in dynamic environments, and the limitations of existing simulation software. To address these challenges and be able to expand our own work to dynamic environments, in Chapter 3 we introduce GRADE, a high-fidelity simulation framework designed to bridge the gap between simulation and reality. GRADE leverages a photorealistic rendering engine to create realistic, interactive, and fully customizable environments, offering fine-grained control over all components of the simulation, ranging from automatic placement of animated assets to fine-grained simulation control. Unlike traditional simulators, which often lack flexibility in modifying environments or fail to accurately simulate physical interactions, GRADE integrates dynamic elements such as moving humans, animated animals, and variable lighting conditions, enabling the realistic evaluation of perception and control strategies in diverse, real-world-inspired scenarios. Moreover, GRADE supports physically enabled environments, allowing robots to interact with objects and surfaces in ways that closely resemble real-world conditions. In Section 3.4.3, we also introduce a novel method within GRADE that enables not only the precise repetition of past experiments but also fine-grained modifications to them. This means that researchers can alter scene conditions, adjust sensor configurations, or introduce new obstacles while maintaining control over experimental variables—an essential capability for both robotics benchmarking and deep learning model training. These characteristics make GRADE a powerful tool not only for generating synthetic datasets for training deep learning approaches but also for rigorously evaluating autonomous robotics methods that must actively interact with their environment. Specifically, we demonstrate GRADE's syn-to-real capabilities by introducing a large-scale, automatically generated dataset of videos collected in indoor environments with animated humans and flying objects (Section 3.5). Using this dataset, we train YOLOv5 and Mask R-CNN models exclusively on synthetic data, achieving strong performance in real-world scenarios (Section 3.6.2). By supporting systematic benchmarking, iterative improvements, and large-scale data generation, GRADE represents a significant step toward the scalable development of intelligent robotic systems capable of operating in dynamic, real-world environments.

Following up on that, in Chapter 4 we employ GRADE to tackle a longstanding challenge in V-SLAM: dynamic environments. Conventional Dynamic V-SLAM methods suffer from the erroneous rejection of useful visual features as they rely on coarse object detection or optical flow mechanisms. These often misclassify moving visual features or suffer failures when evaluated in the few available benchmarking datasets. We show this in Section 4.3, by using GRADE to create different evaluation sequences and extensively test state-of-the-art methods in static and dynamic environments. There, we not only show how existing methods fail to completely or correctly track trajectories but also

how not always the best-performing detection/segmentation models yield the best SLAM performance (Section 4.3.2). Moreover, existing approaches fail to consider the varying motion states of objects — e.g. some objects move at different rates, some remain static temporarily, or some can move only parts of them. Also, they do not consider that they could still contribute to the motion estimation process if stationary. To solve this, in Section 4.4 we introduce DynaPix, a novel Dynamic V-SLAM method designed for indoor environments. DynaPix addresses these limitations by estimating motion probabilities at the pixel level, enabling the SLAM system to selectively incorporate features that, while moving, still provide valuable information for camera pose optimization. These probabilities are then integrated into the bundle adjustment procedures of the SLAM backend (Section 4.4.2), which dynamically adjust their contribution based on their current motion states. This fine-grained approach to dynamic scene understanding leads to longer tracking times and reduced trajectory errors, as validated on both synthetic data from GRADE and real-world datasets. The introduction of per-pixel motion probabilities represents a shift toward more human-like scene interpretation, where the system can distinguish between transient disturbances and reliable static features.

Finally, in Chapter 5, the thesis explores the use of synthetic data to address the challenges of training deep learning models in cases where real-world annotated data is scarce or expensive to acquire. While real-world datasets suffer from annotation errors, bias, and limited availability, synthetic data offers a controlled, scalable alternative. Based on the preliminary result obtained in Section 3.6.2, we employ a similar strategy in a more challenging and impactful application: aerial detection and pose estimation of wild zebras. Accurate monitoring of wildlife populations is critical for conservation efforts, helping track endangered species and study migration patterns, for example. However, data collection in remote environments is often impractical, requiring costly aerial surveys and manual annotation. Moreover, while aerial observation is necessary to avoid introducing disturbance to the animal, available datasets often do not represent this peculiar point of view. Therefore, in Section 5.3 we introduce ZebraPose, the first synthetic-to-real method for top-down zebra detection and 2D pose estimation. There, besides the new extensive synthetic dataset that we generate using our GRADE framework, we also introduce a new real-world dataset collected by us consisting of zebras observed from UAVs and precisely labeled with bounding-boxes. The results show that models trained solely on high-fidelity synthetic data can achieve competitive performance on real-world tasks, eliminating the need for costly and time-consuming real-world data collection. This contribution highlights the importance of data quantity, quality, and variability in bridging the sim-to-real gap. Notably, ZebraPose is the first approach to demonstrate that both detection and pose estimation of animals can be performed using exclusively synthetic data, setting a precedent for future applications in wildlife monitoring and conservation.

## 6.2 Future Research Directions

While this thesis has laid a robust foundation in integrating visual perception with action for autonomous systems, there remain several promising avenues for future research.

### 6.2.1 Toward Full (3D) Trajectory Planning in Active SLAM

The iRotate framework has demonstrated impressive performance in 2D scenarios, yet real-world applications (especially those involving aerial robots) demand exploration in three dimensions. Extending iRotate to 3D environments introduces several challenges. Aerial platforms often cannot afford the computational overhead associated with start-and-stop behaviors, and their continuous motion requires more sophisticated motion prediction and control strategies. Currently, approximation techniques are used in 3D environments to make the computation of optimal headings feasible [10, 47], possibly ignoring potential information gains. Future work should thus focus on developing deep learning models to predict optimal headings in 3D, utilizing synthetic data generated via the GRADE framework for training. Moreover, with the exception of exterior building reconstruction, often the UAVs are assumed to be at a fixed height. However, it might be advantageous to plan the full trajectory in 3D, including the height of the drone, although this would increase the computational complexity. In this, the integration of learned techniques with, for example, Reinforcement Learning could help. Finally, while the historical focus in Active SLAM has been on the control and optimization of the perception system along the shortest path, future work should also develop strategies that facilitate the discovery of paths that are more informative on their own. Instead of merely following the shortest route, future Active SLAM methods could incorporate information-based costmaps and uncertainty-aware mapping techniques to select trajectories that maximize long-term gains and minimize mapping errors. Such approaches would push the boundaries of Active SLAM by enabling robots to continuously explore and map complex environments while maintaining robust localization potentially using longer but more informative paths.

### 6.2.2 Dynamic-aware and Task-driven Active SLAM

Clearly, the combination of dynamic content and Active SLAM is of utmost importance as most real-world environments are inherently dynamic with moving objects, humans, and changing conditions that significantly impact SLAM performance. Traditional Active SLAM assumes a largely static world, while Dynamic SLAM focuses on filtering out moving objects to maintain localization. However, a more comprehensive approach would actively leverage dynamic elements rather than simply treating them as disturbances. For example, predicting the motion of people in a crowded environment would allow robots to optimize exploration strategies, navigate more safely, and build better long-term maps that account for changes over time [67, 86]. This is particularly relevant

for applications like assistive robotics, autonomous navigation in urban environments, and industrial automation. A key advancement in this direction is the explicit modeling of humans and their behavior, leveraging recent developments in articulated models such as SMPL for human pose estimation and motion prediction [201]. By integrating human-aware SLAM, robots can anticipate and adapt to human motion, improving interaction, safety, and efficiency in shared spaces. Beyond static representations, future forecasting models could enable continuous operation in dynamic environments by predicting the future states of moving agents. This would allow Active SLAM systems to proactively adjust exploration strategies, avoiding occlusions, improving long-term mapping, and enabling better decision-making in human-centric environments. Another promising direction is the combination of Active SLAM with focused tasks, such as object discoverability, potentially driven by Large Language Models (LLMs). This would enable robots to prioritize exploration based on high-level goals, such as searching for specific objects, identifying anomalies, or gathering relevant information in unknown environments. Such an approach has direct applications in disaster scenarios (e.g. search and rescue), autonomous inventory management, and adaptive navigation in unfamiliar spaces. By leveraging LLMs, a robot could receive high-level instructions (e.g. *"Find the nearest fire extinguisher"* in a disaster scene, or *"Locate all misplaced items in a warehouse"*) and dynamically adjust its Active SLAM strategy accordingly. While current Active SLAM techniques focus primarily on exploration, future work could thus explore the integration of task-driven objectives. For example, in scenarios involving robotic manipulation or human-robot interaction, SLAM systems could be tailored to prioritize mapping areas that are most relevant to the task at hand. Reinforcement learning-based policies could then balance exploration with task constraints, ensuring that robots are on the fly mapping their environment but doing so in a way that optimizes overall task performance.

### 6.2.3 Controllable and Continuous Simulations

The GRADE simulation framework has already taken significant steps in this direction, for example, by enabling the simulation of SMPL models in robotics-enabled environments, the testing of Active SLAM in dynamic photorealistic settings, or the low-level control of the simulation in its finest components. However, moving beyond its current capabilities, there is considerable potential to further automate it and increase its usability. Ideally, the setup and control of the simulation itself can be further automated. One promising direction is the integration of LLMs and other foundational models to assist in several aspects, from the generation of diverse environments, to the configuration of simulation, of the robots, or the real-time control of the simulated characters. Inspired by techniques used in [251] for personalized environment generation, [112] for simulation code generation, and approaches from [92, 219] for character motion generation and control, future iterations of GRADE could evolve into a more adaptive and intelligent simulation platform. By leveraging these advancements, GRADE could allow

researchers to automatically generate complex simulation scenarios from high-level textual descriptions, reducing manual effort and streamlining workflow setup. For example, instead of manually configuring environments, researchers could specify tasks in natural language (e.g. *"Simulate a warehouse with ten human workers and an aerial drone navigating through dynamic obstacles"*) and have the system generate the appropriate code, assets, and behaviors. This, while still living in the same physically enabled simulation integrated with the robotic software itself through SIL or ROS. Furthermore, enhanced automation could facilitate rapid prototyping and iterative testing of novel autonomous strategies, particularly in Active and Dynamic SLAM research. With LLM-driven simulation orchestration, researchers could dynamically modify environments on the fly — introducing new obstacles, altering human movement patterns, or generating adversarial conditions to test robustness in real-world-like settings. This would significantly accelerate experimentation cycles and enable more comprehensive evaluations of autonomous methods. By evolving into an adaptive, LLM-assisted simulation environment, GRADE could push the boundaries of data-driven robotics research, offering scalable, customizable, and intelligent simulation tools that keep pace with the rapid advancements in robot learning, perception, and planning.

### 6.2.4 Decomposing SLAM for Robust and Adaptive Systems

While building DynaPix, we noticed how current SLAM frameworks often attempt to address mapping, visual odometry, and control in a single, unified system. While this holistic approach has proven effective in certain scenarios, it also introduces inherent trade-offs — especially in dynamic environments where scene changes and unpredictable motion can degrade performance. We believe that a more modular approach, where each component is optimized for its specific function, could significantly enhance both accuracy and adaptability, as we did in DynaPix. Future research should explore decomposing SLAM into specialized modules, each designed to tackle a distinct aspect of the problem. One module could focus on maintaining a stable, high-fidelity representation of the static elements in the environment, ensuring long-term map reliability. Another could specialize in robust visual odometry, not by filtering out moving objects but by actively modeling and leveraging dynamic elements, such as pedestrians and vehicles, to improve motion estimation. A third module could integrate control and future prediction, enabling the system not only to react to changes but also to anticipate future dynamics, allowing robots to proactively adjust their trajectories based on observed motion patterns. Recent advances in deep learning have shown great promise in improving perception and mapping. However, they remain too computationally intensive for real-time deployment on resource-constrained robots. Many real-world scenarios prevent reliance on cloud computing due to connectivity limitations, latency concerns, or security restrictions. This highlights the need for approaches that do not depend on high-performance computing resources but can instead run efficiently on edge devices and onboard processors. This requires integrating lightweight neural networks for per-pixel motion estimation to re-

duce computational overhead, leveraging efficient GPU-based inpainting to reconstruct occluded or unreliable regions in SLAM maps, and enhancing feature selection strategies to ensure robust tracking even in crowded or fast-changing environments. These regions can then be used only for the reconstruction part, while more critical areas, such as dynamic obstacle avoidance, will not necessarily leverage them. Moreover, Neural Radiance Fields (NeRF) and Gaussian Splatting (GS) present promising avenues for enhanced map representation and visualization, but their application must be carefully considered. While high-quality visualizations can improve human interpretability and decision-making, they are not strictly necessary for autonomous robot operation. Future research should separate real-time robotic perception from offline or asynchronous processing, ensuring that robots can function autonomously in real-time while additional post-processing and visualization tasks run separately to support human operators when needed. Finally, layered approaches such as scene graphs [183] integrated with natural-language descriptions could facilitate a more seamless integration of robots into society, enabling more complex behaviors. By adopting a modular, learning-driven approach, future SLAM systems could better adapt to dynamic scenes, bridging the gap between map stability, real-time perception, and predictive control. This would allow autonomous systems to operate more reliably across diverse real-world applications, from urban navigation to assistive robotics and industrial automation. In this context, GRADE remains a valuable tool, providing a controlled environment for testing and refining perception and mapping strategies under dynamic conditions. However, several challenges still need to be addressed in SLAM before achieving a robust foundation for autonomous systems that can be seamlessly integrated into daily life. Advancing beyond restricted environments will require further progress in human-robot interaction and collaboration, ensuring adaptability and generalization across a wider range of real-world scenarios.

### 6.2.5 Cooperative (Active) SLAM at Scale for Animal Conservation

As demonstrated in ZebraPose, synthetic data is increasingly capable of supporting real-world deployment. However, conservation scenarios demand broader generalization, as species vary widely in appearance, behavior, and habitat. Future work could extend ZebraPose to other animals through domain adaptation and multi-species synthetic datasets, ensuring perception models remain robust across ecosystems. Also, multi-robot systems could enable fine-grained 3D animal understanding in the wild, such as estimating the 3D pose and shape of individual animals. In this direction, our prior work on AirPose and AirCapRL (not included in this thesis) offers a compelling starting point. By integrating such capabilities into a broader SLAM framework, we could also go beyond localization and mapping, enabling rich, behaviorally-informed models of wildlife that operate directly in the field. By combining Active SLAM, dynamic scene understanding, and synthetic data-driven adaptation, autonomous systems can make conservation efforts more scalable, intelligent, and resilient. With its ability to simulate heterogeneous multi-robot systems, GRADE provides an ideal platform to explore and validate these

research directions in realistic, yet controllable, scenarios. Indeed, beyond improving individual perception models, advancing SLAM for multi-robot coordination is crucial for scaling conservation efforts to large, unstructured environments. Thus, an exciting application for the advances presented in this thesis is their integration into multi-robot (Active) SLAM systems tailored for conservation. Autonomous agents could significantly enhance wildlife monitoring, habitat preservation, and anti-poaching efforts. For instance, aerial drones can provide real-time surveillance, track animal movements, and reposition based on detected activity or behavior. At the same time, ground robots can assist in terrain exploration and serve as communication relays in remote areas. A key research challenge remains adaptive coordination: enabling robots to dynamically switch roles—such as leading exploration, relaying data, or refining the map—based on real-time conditions and discoveries. This adaptability is especially vital in conservation, where environments and animal behaviors are highly dynamic, and broadband communication might not be available.

## 6.3 Closing Remarks

This thesis has explored the integration of visual perception and action to advance autonomous systems capable of operating in dynamic, real-world environments. We have taken significant steps toward bridging the gap between perception, decision-making, and interaction through contributions in Active SLAM, dynamic scene understanding, and synthetic data generation. The development of iRotate, GRADE, DynaPix, and ZebraPose demonstrates the power of learning-driven approaches to enhance exploration, adaptability, and efficiency in robotics.

However, the journey toward truly intelligent autonomous systems is far from complete. Future research must continue pushing the boundaries of SLAM in dynamic settings, focusing on scalability, efficiency, and real-time adaptability. The challenge of balancing computational constraints with high-level reasoning remains an open problem, particularly for resource-limited robots operating in the field. Developing modular SLAM architectures, integrating human-aware models, and leveraging adaptive multi-robot coordination will be essential in advancing real-world applications, from robotic exploration to conservation and beyond.

Ultimately, the seamless integration of perception, prediction, learning and reasoning, with finally actions, is what will enable autonomous systems to not only navigate their environments but also understand, anticipate, and interact with them and what is there intelligently. By combining human-like reasoning, scalable simulation, and efficient decision-making, we move closer to a future where robots can operate alongside humans and animals with awareness, autonomy, and purpose.

# Bibliography


[1] Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual SLAM. *Expert Systems with Applications*, 205:117734, 2022. 10, 71

[2] Bulat Abbyasov, Roman Lavrenov, Aufar Zakiev, Konstantin Yakovlev, Mikhail Svinin, and Evgeni Magid. Automatic tool for Gazebo world construction: from a grayscale image to a 3D solid model. In *2020 IEEE International Conference on Robotics and Automation*, pages 7226–7232. IEEE, 2020. 72

[3] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver, 10 2023. 15

[4] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. SCAPE: shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, 2005. 75

[5] Rana Azzam, Tarek Taha, Shoudong Huang, and Yahya Zweiri. Feature-based visual simultaneous localization and mapping: A survey. *SN Applied Sciences*, 2:1–24, 2020. 10

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. 109

[7] Sanghoon Baek, Tae-Kyeong Lee, OH Se-Young, and Kwangro Ju. Integrated on-line localization, mapping and coverage algorithm of unknown environments for robotic vacuum cleaners based on minimal sensing. *Advanced Robotics*, 25(13-14):1651–1673, 2011. 4

[8] Irene Ballester, Alejandro Fontán, Javier Civera, Klaus H. Strobl, and Rudolph Triebel. DOT: Dynamic Object Tracking for Visual SLAM. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11705–11711, 2021. 109

[9] Clemens G. Bartnik and Iris I. A. Groen. Visual Perception in the Human Brain: How the Brain Perceives and Understands Real-World Scenes, Nov. 2023. 2

[10] Ana Batinovic, Antun Ivanovic, Tamara Petrovic, and Stjepan Bogdan. A shadowcasting-based next-best-view planner for autonomous 3d exploration. *IEEE Robotics and Automation Letters*, 7(2):2969–2976, 2022. 172

[11] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 13

[12] Ertugrul Bayraktar, Cihat Bora Yigit, and Pinar Boyraz. A hybrid image dataset toward bridging the gap between real and simulation environments for robotics: Annotated desktop objects real and synthetic images dataset: ADORESet. *Machine Vision and Applications*, 30(1):23–40, Aug. 2018. 72, 103

[13] Sara Beery, Yang Liu, Dan Morris, Jim Piavis, Ashish Kapoor, Neel Joshi, Markus Meister, and Pietro Perona. Synthetic examples improve generalization for rare classes. In *Proceedings of the ieee/cvf winter conference on applications of computer vision*, pages 863–873, 2020. 138, 139

[14] Ayman Beghdadi and Malik Mallem. A comprehensive overview of dynamic visual SLAM and deep learning: concepts, methods and challenges. *Machine Vision and Applications*, 33(4), May 2022. 3, 6

[15] Hugo Bertiche, Meysam Madadi, and Sergio Escalera. CLOTH3D: Clothed 3D Humans. In *European Conference on Computer Vision*, pages 344–359. Springer, 2020. 76, 79, 91

[16] Berta Bescos, Carlos Campos, Juan D. Tardós, and José Neira. DynaSLAM II: Tightly-Coupled Multi-Object Tracking and SLAM. *IEEE Robotics and Automation Letters*, 6(3):5191–5198, 2021. 109

[17] Berta Bescos, JM. Fácil, Javier Civera, and José Neira. DynaSLAM: Tracking, Mapping and Inpainting in Dynamic Environments. *IEEE RA-L*, 3(4):4076–4083, 2018. 3, 6, 75, 107, 109, 111, 115, 121, 127, 129

[18] Pia Bideau and Erik Learned-Miller. It's Moving! A Probabilistic Model for Causal Motion Segmentation in Moving Camera Videos. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 433–449, Cham, 2016. Springer International Publishing. 109

[19] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding Horizon "Next-Best-View" Planner for 3D Exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468, 2016. 5, 21, 27

[20] Michael J. Black, Priyanka Patel, Joachim Tesch, and Jinlong Yang. BEDLAM: A Synthetic Dataset of Bodies Exhibiting Detailed Lifelike Animated Motion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 8726–8737. IEEE, 2023. 76, 103, 138, 141

[21] Adrian Bojko, Romain Dupont, Mohamed Tamaazousti, and Hervé Le Borgne. Learning to Segment Dynamic Objects using SLAM Outliers. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 9780–9787, 2021. 107

[22] RE Bray and MS Edwards. Body condition scoring of captive (zoo) equids. In *Proceedings of the Third Conference on Zoo and Wildlife Nutrition*, 1999. 136

[23] Dorothy Breed, Leith C R Meyer, Johan C A Steyl, Amelia Goddard, Richard Burroughs, and Tertius A Kohn. Conserving wildlife in a changing world: Understanding capture myopathy—a malignant outcome of stress during capture and translocation. *Conservation Physiology*, 7(1), Jan. 2019. 138

[24] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017. 2, 3, 107

[25] Alex Brooks and Tim Bailey. *HybridSLAM: Combining FastSLAM and EKF-SLAM for Reliable Mapping*, page 647–661. Springer Berlin Heidelberg, 2009. 10

[26] Mihai Bujanca, Xuesong Shi, Matthew Spear, Pengpeng Zhao, Barry Lennox, and Mikel Luján. Robust SLAM Systems: Are We There Yet? In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5320–5327. IEEE, 2021. 69, 71, 107, 109, 110, 112, 130

[27] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016. 69

[28] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. 2, 21, 27, 35, 107

[29] Weijia Cai, Lei Huang, and Zhengbo Zou. Actively-exploring thermography-enabled autonomous robotic system for detecting and registering HVAC thermal leaks. *Automation in Construction*, 152:104901, 2023. 4

[30] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021. 21

[31] Jinkun Cao, Hongyang Tang, Haoshu Fang, Xiaoyong Shen, Yu-Wing Tai, and Cewu Lu. Cross-Domain Adaptation for Animal Pose Estimation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9497–9506. IEEE, 2019. 137, 140, 141

[32] Peter Carr, Michael Mistry, and Iain Matthews. Hybrid robotic/virtual pan-tilt-zom cameras for autonomous event recording. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 193–202, 2013. 28

[33] Henry Carrillo, Philip Dames, Vijay Kumar, and José A. Castellanos. Autonomous robotic exploration using a utility function based on Rényi's general theory of entropy. *Autonomous Robots*, 42(2):235–256, 2018. 25, 27, 35, 48, 66

[34] David Caruso, Jakob Engel, and Daniel Cremers. Large-scale direct SLAM for omnidirectional cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 141–148, 2015. 29

[35] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D Data in Indoor Environments. *International Conference on 3D Vision (3DV)*, 2017. 74

[36] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository, 2015. 78

[37] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning To Explore Using Active Neural SLAM. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 30

[38] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping. In *Robotics: Science and Systems*, volume 11, pages 3–12. Rome, 2015. 21

[39] Guanxiong Chen, Haoyu Yang, and Ian M. Mitchell. ROS-X-Habitat: Bridging the ROS Ecosystem with Embodied AI. In *2022 19th Conference on Robots and Vision (CRV)*, pages 24–31, 2022. 73

[40] Y. Chen, S. Huang, and R. Fitch. Active SLAM for Mobile Robots With Area Coverage and Obstacle Avoidance. *IEEE/ASME Transactions on Mechatronics*, 25(3):1182–1192, 2020. 3, 25, 27

[41] Jiyu Cheng, Hong Zhang, and Max Q.-H. Meng. Improving Visual Localization Accuracy in Dynamic Environments Based on Dynamic Region Removal. *IEEE Transactions on Automation Science and Engineering*, 17(3):1585–1596, 2020. 109

[42] Shuhong Cheng, Changhe Sun, Shijun Zhang, and Dianfan Zhang. SG-SLAM: A Real-Time RGB-D Visual SLAM Toward Dynamic Scenes With Semantic and Geometric Information. *IEEE Transactions on Instrumentation and Measurement*, 72:1–12, 2023. 107

[43] HeeSun Choi, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory Hager, David Han, Frank Hearl, Jessica Hodgins, Abhinandan Jain, Frederick Leve, Chen Li, Franziska Meier, Dan Negrut, Ludovic Righetti, Alberto Rodriguez, Jie Tan, and Jeff Trinkle. On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, 118(1), Dec. 2020. 3

[44] Jack Collins, David Howard, and Jurgen Leitner. Quantifying the Reality Gap in Robotic Manipulation Tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6706–6712, 2019. 3

[45] Peter Corke. *Robotics, Vision and Control*. Springer International Publishing, 2017. 2

[46] Gabriela Csurka, Christopher R. Dance, Florent Perronnin, and Jutta Willamowski. *Generic Visual Categorization Using Weak Geometry*, page 207–224. Springer Berlin Heidelberg, 2006. 16

[47] A. Dai, S. Papatheodorou, N. Funk, D. Tzoumanikas, and S. Leutenegger. Fast Frontier-based Information-driven Autonomous Exploration with an MAV. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9570–9576, 2020. 4, 22, 25, 26, 27, 28, 35, 48, 172

[48] F. Daum and J. Huang. Curse of dimensionality and particle filters. In *2003 IEEE Aerospace Conference Proceedings (Cat. No.03TH8652)*, volume 4, pages 1979–1993, 2003. 10

[49] Andrew J. Davison and David W. Murray. Mobile robot localisation using active vision. In Hans Burkhardt and Bernd Neumann, editors, *European conference on computer vision*, pages 809–825, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. 26, 29

[50] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007. 10, 21

[51] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 74

[52] A. Del Bimbo, G. Lisanti, I. Masi, and F. Pernici. Continuous recovery for real time pan tilt zoom localization and mapping. In *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 160–165, 2011. 28

[53] L. Del Pero, S. Ricco, R. Sukthankar, and V. Ferrari. Behavior Discovery and Alignment of Articulated Object Classes from Unstructured Video. *International Journal of Computer Vision (IJCV)*, 2016. 152

[54] Maximilian Denninger, Dominik Winkelbauer, Martin Sundermeyer, Wout Boerdijk, Markus Knauer, Klaus H. Strobl, Matthias Humt, and Rudolph Triebel. BlenderProc2: A Procedural Pipeline for Photorealistic Rendering. *Journal of Open Source Software*, 8(82):4901, 2023. 78

[55] Soumyabrata Dey, Vladimir Reilly, Imran Saleemi, and Mubarak Shah. Detection of independently moving objects in non-planar scenes via multi-frame monocular epipolar constraint. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V 12*, pages 860–873. Springer, 2012. 109

[56] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning Optical Flow with Convolutional Networks. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2758–2766. IEEE Computer Society, 2015. 109, 120

[57] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B. McHugh, and Vincent Vanhoucke. Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2553–2560. IEEE, 2022. 78

[58] Kamak Ebadi, Lukas Bernreiter, Harel Biggie, Gavin Catt, Yun Chang, Arghya Chatterjee, Christopher E Denniston, Simon-Pierre Deschênes, Kyle Harlow, Shehryar Khattak, et al. Present and future of slam in extreme environments: The darpa subt challenge. *IEEE Transactions on Robotics*, 40:936–959, 2023. 4

[59] Salehe Erfanian Ebadi, Saurav Dhakad, Sanjay Vishwakarma, Chunpu Wang, You-Cyuan Jhang, Maciek Chociej, Adam Crespi, Alex Thaman, and Sujoy Ganguly. PSP-HDRI+: A Synthetic Dataset Generator for Pre-Training of Human-Centric Computer Vision Models. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022. 3, 6, 7, 76, 99, 101, 138, 139

[60] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 14

[61] James T Enns and Alejandro Lleras. What's next? New evidence for prediction in human vision. *Trends in cognitive sciences*, 12(9):327–333, 2008. 3

[62] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. 137

[63] Maurice F Fallon, John Folkesson, Hunter McClelland, and John J Leonard. Relocating underwater features autonomously using sonar-based SLAM. *IEEE Journal of Oceanic Engineering*, 38(3):500–513, 2013. 2

[64] Andrew Farley, Jie Wang, and Joshua A. Marshall. How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion. *Simulation Modelling Practice and Theory*, 120:102629, 2022. 72

[65] Olivier D. Faugeras. *What can be seen in three dimensions with an uncalibrated stereo rig?*, page 563–578. Springer Berlin Heidelberg, 1992. 13

[66] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014. 46

[67] Mark Nicholas Finean, Luka Petrović, Wolfgang Merkt, Ivan Marković, and Ioannis Havoutis. Motion planning in dynamic environments using context-aware human trajectory prediction. *Robotics and Autonomous Systems*, 166:104450, 2023. 172

[68] S. Frintrop and P. Jensfelt. Attentional Landmarks and Active Gaze Control for Visual SLAM. *IEEE Transactions on Robotics*, 24(5):1054–1065, 2008. 26, 28, 29

[69] Huan Fu, Bowen Cai, Lin Gao, Lingxiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, and Hao Zhang. 3D-FRONT: 3D Furnished Rooms with layOuts and semaNTics. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 10913–10922. IEEE, 2021. 75, 78, 91

[70] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3D-FUTURE: 3D furniture shape with texture. *International Journal of Computer Vision*, 129(12):1–25, 2021. 75

[71] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016. 83, 85, 111

[72] Marianna Bergamaschi Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jonathan Lenchner, Andrea Loreggia, Nicholas Mattei, Francesca Rossi, Biplav Srivastava, Kristen Brent Venable, et al. Combining Fast and Slow Thinking for Human-like and Efficient Decisions in Constrained Environments. In *NeSy*, pages 171–185, 2022. 2

[73] Hector Geffner. Model-free, model-based, and general intelligence. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 10–17, 2018. 2

[74] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013. 69

[75] Patrick Geneva, James Maley, and Guoquan Huang. An efficient schmidt-ekf for 3D visual-inertial SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12105–12115, 2019. 10

[76] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., USA, 2006. 2, 11

[77] Jacob M Graving, Daniel Chae, Hemal Naik, Liang Li, Benjamin Koger, Blair R Costelloe, and Iain D Couzin. DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning. *eLife*, 8:e47994, 2019. xxii, 140, 141, 152

[78] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J. Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam H. Laradji, Hsueh-Ti Derek Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, A. Cengiz Öztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: A scalable dataset generator. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 3739–3751. IEEE, 2022. 72

[79] Anders Grunnet-Jepsen, John N. Sweetser, and John Woodfill. BKMs Tuning RealSense D4xx Cam. https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/BKMs_Tuning_RealSense_D4xx_Cam.pdf. [Accessed 23-Feb-2023]. 45, 111

[80] Abhishek Gupta, Alagan Anpalagan, Ling Guan, and Ahmed Shaharyar Khwaja. Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, 10:100057, 2021. 2

[81] David Hall, Ben Talbot, Suman Raj Bista, Haoyang Zhang, Rohan Smith, Feras Dayoub, and Niko Sünderhauf. BenchBot environments for active robotics (BEAR): Simulated data for active scene understanding research. *The International Journal of Robotics Research*, 41(3):259–269, 2022. 73, 75

[82] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. 37

[83] Jiaming He, Mingrui Li, Yangyang Wang, and Hongyu Wang. OVD-SLAM: An Online Visual SLAM for Dynamic Environments. *IEEE Sensors Journal*, 23(12):13210–13219, 2023. 6, 110, 128

[84] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2980–2988. IEEE Computer Society, 2017. 2, 71, 98, 107, 109

[85] Zhu Heming, Cao Yu, Jin Hang, Chen Weikai, Du Dong, Wang Zhangye, Cui Shuguang, and Han Xiaoguang. Deep Fashion3D: A Dataset and Benchmark for 3D Garment Reconstruction from Single Images. In *Computer Vision – ECCV 2020*, pages 512–530. Springer International Publishing, 2020. 76

[86] Dorian F Henning, Tristan Laidlow, and Stefan Leutenegger. Bodyslam: joint camera localisation, mapping, and human motion tracking. In *European Conference on Computer Vision*, pages 656–673. Springer, 2022. 172

[87] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77, 2013. 15

[88] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013. 14, 78

[89] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit—An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011. 40, 46

[90] Xinggang Hu, Yunzhou Zhang, Zhenzhong Cao, Rong Ma, Yanmin Wu, Zhiqiang Deng, and Wenkai Sun. CFP-SLAM: A Real-time Visual SLAM Based on Coarse-to-Fine Probability in Dynamic Environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4399–4406, 2022. 107, 109, 110, 128

[91] Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. SceneNN: A Scene Meshes Dataset with aNNotations. In *International Conference on 3D Vision (3DV)*, 2016. 74

[92] Yiming Huang, Weilin Wan, Yue Yang, Chris Callison-Burch, Mark Yatskar, and Lingjie Liu. CoMo: Controllable Motion Generation Through Language Guided Pose Code Editing. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XXIX*, page 180–196, Berlin, Heidelberg, 2024. Springer-Verlag. 173

[93] Zhaoyang Huang, Xiaoyu Shi, Chao Zhang, Qiang Wang, Ka Chun Cheung, Hongwei Qin, Jifeng Dai, and Hongsheng Li. FlowFormer: A Transformer Architecture for Optical Flow. *ECCV*, 2022. 124

[94] Hyukseong Kwon, Youngrock Yoon, Jae Byung Park, and A. C. Kak. Person Tracking with a Mobile Robot using Two Uncalibrated Independently Moving Cameras. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2877–2883, 2005. 28

[95] M. Irani and P. Anandan. A unified approach to moving object detection in 2D and 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):577–589, 1998. 2, 109

[96] Marcelo Jacinto, João Pinto, Jay Patrikar, John Keller, Rita Cunha, Sebastian Scherer, and António Pascoal. Pegasus simulator: An isaac sim framework for multiple aerial vehicles simulation. In *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 917–922, 2024. 82

[97] Tete Ji, Chen Wang, and Lihua Xie. Towards Real-time Semantic RGB-D SLAM in Dynamic Environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11175–11181, 2021. 127

[98] Le Jiang, Caleb Lee, Divyang Teotia, and Sarah Ostadabbas. Animal pose estimation: A closer look at the state-of-the-art, existing gaps and opportunities. *Computer Vision and Image Understanding*, 222:103483, 2022. 139, 141, 151

[99] Le Jiang, Shuangjun Liu, Xiangyu Bai, and Sarah Ostadabbas. Prior-Aware Synthetic Data to the Rescue: Animal Pose Estimation with Very Limited Real Data. In *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*, page 868. BMVA Press, 2022. 138, 139, 141, 151, 152

[100] Le Jiang and Sarah Ostadabbas. SPAC-Net: Synthetic Pose-aware Animal ControlNet for Enhanced Pose Estimation, 2023. 7, 138, 139, 141, 149, 150, 151

[101] Li Jinyu, Yang Bangbang, Chen Danpeng, Wang Nan, Zhang Guofeng, and Bao Hujun. Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality. *Virtual Reality & Intelligent Hardware*, 1(4):386–410, 2019. 107

[102] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, Zeng Yifu, Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, 2022. 7, 71, 98, 107

[103] Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. RViz: a toolkit for real domain data visualization. *Telecommun. Syst.*, 60(2):337–345, 2015. xvii, 38

[104] Mina Kamel, Thomas Stastny, Kostas Alexis, and Roland Siegwart. Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System. In Anis Koubaa, editor, *Robot Operating System (ROS) The Complete Reference, Volume 2*. Springer, 2017. 87

[105] Ravpreet Kaur and Sarbjeet Singh. A comprehensive review of object detection with deep learning. *Digital Signal Processing*, 132:103812, 2023. 2

[106] Mukul Khanna, Yongsen Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Shacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X Chang, and Manolis Savva. Habitat synthetic scenes dataset (hssd-200): An analysis of 3d scene scale and realism tradeoffs for objectgoal navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16384–16393, 2024. 75

[107] Ayoung Kim and Ryan M Eustice. Next-best-view visual SLAM for bounded-error area coverage. In *IROS workshop on active semantic perception*, pages 1–6, 2012. 21

[108] Jaeseok Kim, Anand Kumar Mishra, Raffaele Limosani, Marco Scafuro, Nino Cauli, Jose Santos-Victor, Barbara Mazzolai, and Filippo Cavallo. Control strategies for cleaning robots in domestic applications: A comprehensive review. *International Journal of Advanced Robotic Systems*, 16(4):172988141985743, July 2019. 2

[109] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007. 2

[110] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and*

*Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3. IEEE, 2004. 44, 69, 72

[111] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *ArXiv preprint*, abs/1712.05474, 2017. 72

[112] Peter Kulits, Haiwen Feng, Weiyang Liu, Victoria Fernandez Abrevaya, and Michael J. Black. Re-Thinking Inverse Graphics With Large Language Models. *Transactions on Machine Learning Research*, 2024. 173

[113] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. 15

[114] 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, 2019. 2, 3, 5, 21, 46, 111

[115] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010. 137

[116] C. Leung, S. Huang, and G. Dissanayake. Active SLAM using Model Predictive Control and Attractor based Exploration. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5031, 2006. 3, 25, 26, 27, 35

[117] Ao Li, Jikai Wang, Meng Xu, and Zonghai Chen. DP-SLAM: A visual SLAM with moving probability towards dynamic environments. *Information Sciences*, 556:128–142, 2021. 109

[118] Borui Li, Fuchun Sun, Huaping Liu, and Bin Fang. Path Planning for Active V-Slam Based on Reinforcement Learning. In Fuchun Sun, Huaping Liu, and Dewen Hu, editors, *Cognitive Systems and Signal Processing*, pages 477–487, Singapore, 2019. Springer Singapore. 30

[119] Chen Li and Gim Hee Lee. From Synthetic to Real: Unsupervised Domain Adaptation for Animal Pose Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 1482–1491. Computer Vision Foundation / IEEE, 2021. 3, 7, 139, 140, 141, 151, 152

[120] Chen Li and Gim Hee Lee. ScarceNet: Animal Pose Estimation with Scarce Annotations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 17174–17183. IEEE, 2023. 141

[121] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A Benchmark for Embodied AI with 1,000 Everyday Activities and Realistic Simulation. In *6th Annual Conference on Robot Learning*, 2022. 73

[122] Jie Li, Michael Kaess, Ryan M Eustice, and Matthew Johnson-Roberson. Pose-graph SLAM using forward-looking sonar. *IEEE Robotics and Automation Letters*, 3(3):2330–2337, 2018. 2

[123] Jiefeng Li, Chao Xu, Zhicun Chen, Siyuan Bian, Lixin Yang, and Cewu Lu. HybrIK: A Hybrid Analytical-Neural Inverse Kinematics Solution for 3D Human Pose and Shape Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 3383–3393. Computer Vision Foundation / IEEE, 2021. 75

[124] Wenbin Li, Sajad Saeedi, John McCormac, Ronald Clark, Dimos Tzoumanikas, Qing Ye, Yuzhong Huang, Rui Tang, and Stefan Leutenegger. InteriorNet: Mega-scale Multi-sensor Photo-realistic Indoor Scenes Dataset. In *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*, page 77. BMVA Press, 2018. 74

[125] Yang Li, Hikari Takehara, Takafumi Taketomi, Bo Zheng, and Matthias Nießner. 4DComplete: Non-Rigid Motion Estimation Beyond the Observable Surface. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12686–12696. IEEE, 2021. 140

[126] Zhen Li, Chengze Lu, Jianhua Qin, Chun-Le Guo, and Ming-Ming Cheng. Towards An End-to-End Framework for Flow-Guided Video Inpainting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 17541–17550. IEEE, 2022. 121, 128

[127] Zhengqin Li, Ting-Wei Yu, Shen Sang, Sarah Wang, Meng Song, Yuhan Liu, Yu-Ying Yeh, Rui Zhu, Nitesh B. Gundavarapu, Jia Shi, Sai Bi, Hong-Xing Yu, Zexiang Xu, Kalyan Sunkavalli, Milos Hasan, Ravi Ramamoorthi, and Manmohan Chandraker. OpenRooms: An Open Framework for Photorealistic Indoor Scene Datasets. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 7190–7199. Computer Vision Foundation / IEEE, 2021. 74

[128] G. Lidoris, K. Kuhnlenz, D. Wollherr, and M. Buss. Combined Trajectory Planning and Gaze Direction Control for Robotic Exploration. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4044–4049, 2007. 29

[129] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. *CoRR*, abs/1405.0312:740–755, 2014. xxii, 3, 69, 72, 137, 143, 152

[130] Hailin Liu, Liangfang Tian, Qiliang Du, and Wenjie Xu. Robust RGB: D-SLAM in highly dynamic environments based on probability observations and clustering optimization. *Measurement Science and Technology*, 35(3):035405, 2023. 110

[131] Jianheng Liu, Xuanfu Li, Yueqian Liu, and Haoyao Chen. RGB-D Inertial Odometry for a Resource-Restricted Robot in Dynamic Environments. *IEEE Robotics and Automation Letters*, 7(4):9573–9580, 2022. 6, 75, 109, 111, 118, 129

[132] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. pages 38–55, 2024. 141

[133] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer, Springer International Publishing. 150

[134] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. FlowNet3D: Learning Scene Flow in 3D Point Clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 529–537. Computer Vision Foundation / IEEE, 2019. 109

[135] Yu Liu and Zhiyu Zhou. Optical Flow-Based Stereo Visual Odometry With Dynamic Object Detection. *IEEE Transactions on Computational Social Systems*, pages 1–13, 2022. 107, 109

[136] Iker Lluvia, Elena Lazkano, and Ander Ansuategi. Active Mapping and Robot Exploration: A Survey. *Sensors*, 21:2445, 2021. 21, 25, 26, 35

[137] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, Sept. 1981. 13

[138] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graph.*, 34(6), 2015. 75, 140

[139] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004. 13

[140] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of Motion Capture As Surface Shapes. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 5441–5450. IEEE, 2019. 75, 91, 137

[141] Dario Mammolo. Active SLAM in Crowded Environments, 2019. 34

[142] Travis Manderson, Florian Shkurti, and Gregory Dudek. Texture-aware SLAM using stereo imagery and inertial information. In *2016 13th Conference on Computer and Robot Vision (CRV)*, pages 456–463. IEEE, 2016. 29

[143] Alexander Mathis, Thomas Biasi, Steffen Schneider, Mert Yuksekgonul, Byron Rogers, Matthias Bethge, and Mackenzie W. Mathis. Pretraining Boosts Out-of-Domain Robustness for Pose Estimation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1859–1868, 2021. 140

[144] Matías Mattamala, Constanza Villegas, José Miguel Yáñez, Pablo Cano, and Javier Ruiz-del Solar. A Dynamic and Efficient Active Vision System for Humanoid Soccer Robots. In Luis Almeida, Jianmin Ji, Gerald Steinbauer, and Sean Luke, editors, *RoboCup 2015: Robot World Cup XIX*, pages 316–327, Cham, 2015. Springer International Publishing. 29

[145] Etienne Meunier, Anaïs Badoual, and Patrick Bouthemy. EM-Driven Unsupervised Learning for Efficient Motion Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4462–4473, 2023. 109

[146] O. Michel. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004. 69

[147] Ashkan Mirzaei, Tristan Aumentado-Armstrong, Konstantinos G. Derpanis, Jonathan Kelly, Marcus A. Brubaker, Igor Gilitschenski, and Alex Levinshtein. SPIn-NeRF: Multi-view Segmentation and Perceptual Inpainting with Neural Radiance Fields. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 20669–20679. IEEE, 2023. 121

[148] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598:593–598, 2002. 10, 21

[149] T. Moore and D. Stouch. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, 2014. 33

[150] Jiteng Mu, Weichao Qiu, Gregory D. Hager, and Alan L. Yuille. Learning From Synthetic Animals. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 12383–12392. IEEE, 2020. 140, 141, 149, 151

[151] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications. *International Journal of Computer Vision*, 126(9):902–919, 2018. 72

[152] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. 2, 3, 7, 10, 108, 109, 111, 127

[153] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011. 10

[154] Simon Niklaus and Feng Liu. Softmax Splatting for Video Frame Interpolation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 5436–5445. IEEE, 2020. 122, 123

[155] Farzan M. Noori, David Portugal, Rui P. Rocha, and Micael S. Couceiro. On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo? In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 19–24, 2017. 72

[156] Linus Nwankwo and Elmar Rueckert. Understanding Why SLAM Algorithms Fail in Modern Indoor Environments. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 186–194. Springer, 2023. 3

[157] C. Papachristos, S. Khattak, and K. Alexis. Uncertainty-aware receding horizon exploration and mapping using aerial robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4568–4575, 2017. 3, 21, 23, 25, 26, 27, 28, 35, 39

[158] Seonwook Park, Thomas Schöps, and Marc Pollefeys. Illumination change robustness in direct visual slam. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 4523–4530. IEEE, 2017. 10

[159] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. Expressive Body Capture: 3D Hands, Face, and Body From a Single Image. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 10975–10985. Computer Vision Foundation / IEEE, 2019. 73, 75

[160] Marius V. Peelen, Eva Berlot, and Floris P. de Lange. Predictive processing of scenes and objects. *Nature Reviews Psychology*, 3(1):13–26, Nov. 2023. 2, 3

[161] Kai Pervolz, A Nuchter, Hartmut Surmann, Joachim Hertzberg, and S Birlinghoven. Automatic reconstruction of colored 3d models. *VDI BERICHTE*, 1841:215–222, 2004. 28

[162] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos. A survey on active simultaneous localization and mapping: State of the art and new frontiers. *IEEE Transactions on Robotics (T-RO)*, 2023. 2, 3, 4, 21, 27, 107

[163] Jonathan Platt and Kenneth Ricks. Comparative Analysis of ROS-Unity3D and ROS-Gazebo for Mobile Ground Robot Simulation. *Journal of Intelligent & Robotic Systems*, 106(4):80, 2022. 72

[164] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3):133–148, 2013. 33

[165] Eric Price and Aamir Ahmad. Accelerated Video Annotation driven by Deep Detector and Tracker. In *International Conference on Intelligent Autonomous Systems*, pages 141–153. Springer, 2023. 150

[166] Eric Price, Pranav C Khandelwal, Daniel I Rubenstein, and Aamir Ahmad. A Framework for Fast, Large-scale, Semi-Automatic Inference of Animal Behavior from Monocular Videos. *Methods in Ecology and Evolution*, 16(9):1966–1982, 2025. xxii, 159

[167] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondruš, Theophile Gervet, Vincent-Pierre Berges, John M Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A Co-Habitat for Humans, Avatars and Robots. In *The Twelfth International Conference on Learning Representations*, 2024. 73

[168] Albert Pumarola, Jordi Sanchez, Gary P. T. Choi, Alberto Sanfeliu, and Francesc Moreno. 3DPeople: Modeling the Geometry of Dressed Humans. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 2242–2251. IEEE, 2019. 76

[169] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018. 109

[170] Tong Qin, Jie Pan, Shaozu Cao, and Shaojie Shen. A General Optimization-based Framework for Local Odometry Estimation with Multiple Sensors. *ArXiv preprint*, abs/1901.03638, 2019. 112

[171] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009. 72

[172] Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen Indoors: Photorealistic Indoor Scenes using Procedural Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21783–21794, 2024. 75

[173] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. 74

[174] Ananth Ranganathan, Michael Kaess, and Frank Dellaert. Fast 3D pose estimation with out-of-sequence measurements. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2486–2493, 2007. 10

[175] Anurag Ranjan, Varun Jampani, Lukas Balles, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J. Black. Competitive Collaboration: Joint Unsupervised Learning of Depth, Camera Motion, Optical Flow and Motion Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12240–12249. Computer Vision Foundation / IEEE, 2019. 109

[176] Amir Rasouli and John K Tsotsos. The effect of color space selection on detectability and discriminability of colored objects. *ArXiv preprint*, abs/1702.05421, 2017. 45

[177] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788. IEEE Computer Society, 2016. 2

[178] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Ángel Bautista, Nathan Paczan, Russ Webb, and Joshua M. Susskind. Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 10892–10902. IEEE, 2021. 74

[179] AWS Robotics. GitHub - aws-robotics/aws-robomaker-small-house-world: A house world with multiple rooms and furniture for AWS RoboMaker and Gazebo simulations. — github.com. https://github.com/aws-robotics/aws-robomaker-small-house-world. [Accessed 11-04-2024]. 44

[180] Luis Roldão, Raoul de Charette, and Anne Verroust-Blondet. 3D Semantic Scene Completion: A Survey. *International Journal of Computer Vision*, 130(8):1978–2005, 2022. 74

[181] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied Hands: Modeling

and Capturing Hands and Bodies Together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6), 2017. 75

[182] A. Rosinol, M. Abate, Y. Chang, and L. Carlone. Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1689–1696, 2020. 21

[183] Antoni Rosinol, Arjun Gupta, Marcus Abate, Jingnan Shi, and Luca Carlone. 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans. *Robotics: Science and Systems XVI*, 2020. 3, 175

[184] Matthias Rottmann and Marco Reese. Automated Detection of Label Errors in Semantic Segmentation Datasets via Deep Learning and Uncertainty Quantification. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3214–3223, 2023. 137

[185] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. 13

[186] Martin Runz, Maud Buffier, and Lourdes Agapito. MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 10–20, 2018. 112

[187] Nitin Saini, Elia Bonetto, Eric Price, Aamir Ahmad, and Michael J. Black. AirPose: Multi-View Fusion Network for Aerial 3D Human Pose and Shape Estimation. *IEEE Robotics and Automation Letters*, 7(2):4805–4812, 2022. 2, 3, 6, 69, 75, 76, 103, 137, 138

[188] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2304–2314, 2019. 75

[189] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser-Nam Lim, and Rama Chellappa. Learning From Synthetic Data: Addressing Domain Shift for Semantic Segmentation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3752–3761. IEEE Computer Society, 2018. 138

[190] Mirella Santos Pessoa de Melo, José Gomes da Silva Neto, Pedro Jorge Lima da Silva, João Marcelo Xavier Natario Teixeira, and Veronica Teichrieb. Analysis and Comparison of Robotics 3D Simulators. In *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, pages 242–251, 2019. 72

[191] Muhamad Risqi U. Saputra, Andrew Markham, and Niki Trigoni. Visual SLAM and Structure from Motion in Dynamic Environments: A Survey. *ACM Comput. Surv.*, 51(2):1–36, 2018. 3, 6, 71

[192] Manolis Savva, Jitendra Malik, Devi Parikh, Dhruv Batra, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, and Vladlen Koltun. Habitat: A Platform for Embodied AI Research. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9338–9346. IEEE, 2019. 72

[193] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart, and J. Nieto. An Efficient Sampling-Based Method for Online Informative Path Planning in Unknown Environments. *IEEE Robotics and Automation Letters*, 5(2):1500–1507, 2020. 22, 25, 26, 27, 28

[194] Raluca Scona, Mariano Jaimez, Yvan R. Petillot, Maurice Fallon, and Daniel Cremers. StaticFusion: Background Reconstruction for Dense RGB-D SLAM in Dynamic Environments. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3849–3856. IEEE, 2018. 107, 111

[195] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments. *IEEE Robotics and Automation Letters*, 4(2):1699–1706, 2019. 26, 27, 35

[196] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles, 2017. 72

[197] Yifei Simon Shao, Yuwei Wu, Laura Jarin-Lipschitz, Pratik Chaudhari, and Vijay Kumar. Design and Evaluation of Motion Planners for Quadrotors in Environments with Varying Complexities. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10033–10039, 2024. 71

[198] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne P. Tchapmi, Micael E. Tchapmi, Kent Vainio, Josiah Wong, Li Fei-Fei, and Silvio Savarese. iGibson 1.0: a Simulation Environment for Interactive Tasks in Large Realistic Scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page accepted. IEEE, IEEE, 2021. 72, 73

[199] Shihao Shen, Yilin Cai, Wenshan Wang, and Sebastian Scherer. Dytanvo: Joint refinement of visual odometry and motion segmentation in dynamic environments. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4048–4055. IEEE, 2023. 107, 108

[200] Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, et al. Are we ready for service robots? the openloris-scene datasets for lifelong slam. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 3139–3145. IEEE, 2020. 107

[201] Soyong Shin, Juyong Kim, Eni Halilaj, and Michael J. Black. WHAM: Reconstructing World-grounded Humans with Accurate 3D Motion. In *Computer Vision and Pattern Recognition (CVPR)*, 2024. 173

[202] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition, 2011. 2, 10, 14, 21

[203] R. Sim and N. Roy. Global A-Optimal Robot Exploration in SLAM. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 661–666, 2005. 21, 25

[204] Chelsea V. Smith, Tania C. Gilbert, Tim Woodfine, Alex Kraaijeveld, Geoffrey Chege, David Kimiti, Belinda Low-Mackey, Mathew Mutinda, Shadrack Ngene, Dan Rubenstein, Anthony Wandera, and Philip Riordan. Population and habitat connectivity of Grevy's

zebra Equus grevyi, a threatened large herbivore in degraded rangelands. *Biological Conservation*, 274:109711, Oct. 2022. 139

[205] Randall Smith, Matthew Self, and Peter Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics*, page 435–461. Elsevier, 1988. 10

[206] Randall C. Smith and Peter Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986. 10

[207] Jose Sosa and David Hogg. A Horse with no Labels: Self-Supervised Horse Pose Estimation from Unlabelled Images and Synthetic Prior. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1049–1056, 2023. 149

[208] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *Robotics: Science and Systems*, volume 2, pages 65–72, 2005. 27, 35

[209] Cyrill Stachniss, John J Leonard, and Sebastian Thrun. Simultaneous localization and mapping. *Springer handbook of robotics*, pages 1153–1176, 2016. 10

[210] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica Dataset: A Digital Replica of Indoor Spaces. *ArXiv preprint*, abs/1906.05797, 2019. 74

[211] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. IEEE, 2012. 5, 69, 72, 99, 108, 115, 128

[212] Zhaoqi Su, Tao Yu, Yangang Wang, and Yebin Liu. DeepCloth: Neural Garment Representation for Shape and Style Editing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1581–1593, 2023. 76

[213] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017. 3

[214] Deqing Sun, Stefan Roth, John P Lewis, and Michael J Black. Learning optical flow. In *European Conference on Computer Vision*, pages 83–97. Springer, 2008. 2

[215] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 8934–8943. IEEE Computer Society, 2018. 109, 120

[216] Yuxiang Sun, Ming Liu, and Max Q-H Meng. Motion removal for reliable RGB-D SLAM in dynamic environments. *Robotics and Autonomous Systems*, 108:115–128, 2018. 121

[217] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010. 2, 11

[218] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel X. Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 251–266, 2021. 73

[219] Omid Taheri, Vasileios Choutas, Michael J. Black, and Dimitrios Tzionas. GOAL: Generating 4D Whole-Body Motion for Hand-Object Grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13263–13273, June 2022. 173

[220] Ben Talbot, David Hall, Haoyang Zhang, Suman Raj Bista, Rohan Smith, Feras Dayoub, and Niko Sünderhauf. BenchBot: Evaluating Robotics Research in Photorealistic 3D Simulation and on Real Robots, 2020. 72, 73, 82

[221] Zachary Teed and Jia Deng. DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras. *Advances in neural information processing systems*, 2021. 2, 3

[222] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. 10, 14, 21

[223] Sebastian Thrun and John J. Leonard. *Simultaneous Localization and Mapping*, page 871–889. Springer Berlin Heidelberg, 2008. 2

[224] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. 3

[225] Varun Tolani, Somil Bansal, Aleksandra Faust, and Claire Tomlin. Visual Navigation Among Humans With Optimal Control as a Supervisor. *IEEE Robotics and Automation Letters*, 6(2):2288–2295, 2021. 2, 21

[226] L. Monica Trondrud, Cassandra Ugland, Erik Ropstad, Leif Egil Loe, Steve Albon, Audun Stien, Alina L. Evans, Per Medbøe Thorsby, Vebjørn Veiberg, R. Justin Irvine, and Gabriel Pigeon. Stress responses to repeated captures in a wild ungulate. *Scientific Reports*, 12(1), Sept. 2022. 138

[227] H. Umari and S. Mukhopadhyay. Autonomous robotic exploration based on multiple rapidly-exploring randomized trees. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1396–1402, 2017. 25, 28

[228] J. Vallvé and J. Andrade-Cetto. Active pose SLAM with RRT*. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2167–2173, 2015. 25, 27, 28

[229] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from Synthetic Humans. In *2017 IEEE Conference*

*on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4627–4635. IEEE Computer Society, 2017. 69, 76, 91

[230] Chenjie Wang, Bin Luo, Yun Zhang, Qing Zhao, Lu Yin, Wei Wang, Xin Su, Yajun Wang, and Chengyuan Li. DymSLAM: 4D Dynamic Scene Reconstruction Based on Geometrical Motion Segmentation. *IEEE Robotics and Automation Letters*, 6(2):550–557, 2021. 109

[231] Jingbo Wang, Ye Yuan, Zhengyi Luo, Kevin Xie, Dahua Lin, Umar Iqbal, Sanja Fidler, and Sameh Khamis. Learning Human Dynamics in Autonomous Driving Scenarios. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 20739–20749. IEEE, 2023. 75

[232] Maozhen Wang, Xianchao Long, Peng Chang, and Taşkln Padlr. Autonomous robot navigation with rich information mapping in nuclear storage environments. In *2018 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, pages 1–6. IEEE, 2018. 4

[233] Wenbo Wang, Hsuan-I Ho, Chen Guo, Boxiang Rong, Artur Grigorev, Jie Song, Juan Jose Zarate, and Otmar Hilliges. 4D-DRESS: A 4D Dataset of Real-world Human Clothing with Semantic Annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 76

[234] Wenshan Wang, Yaoyu Hu, and Sebastian Scherer. TartanVO: A Generalizable Learning-based VO. In *Conference on Robot Learning (CoRL)*, 2020. 75, 107, 111

[235] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. TartanAir: A Dataset to Push the Limits of Visual SLAM. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4909–4916. IEEE, 2020. 3, 5, 108

[236] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: towards unleashing infinite data for automated robot learning via generative simulation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. 73

[237] Yanan Wang, Kun Xu, Yaobin Tian, and Xilun Ding. DRG-SLAM: A Semantic RGB-D SLAM using Geometric Features for Indoor Dynamic Scene. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1352–1359, 2022. 109

[238] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 98

[239] Shuhuan Wen, Xiao Chen, Chunli Ma, H.K. Lam, and Shaoyang Hua. The Q-learning obstacle avoidance algorithm based on EKF-SLAM for NAO autonomous walking under unknown environments. *Robotics and Autonomous Systems*, 72:29–36, 2015. 30

[240] Peter Whaite and Frank P Ferrie. Autonomous exploration: Driven by uncertainty. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):193–205, 1997. 3, 4

[241] Siyuan Wu, Gang Chen, Moji Shi, and Javier Alonso-Mora. Decentralized Multi-Agent Trajectory Planning in Dynamic Environments with Spatiotemporal Occupancy Grid

Maps. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7208–7214, 2024. 71

[242] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 101

[243] Fei Xia, Amir Roshan Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: Real-World Perception for Embodied Agents. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9068–9079. IEEE Computer Society, 2018. 5, 74

[244] Zhiwei Xing, Xiaorui Zhu, and Dingcheng Dong. DE-SLAM: SLAM for highly dynamic environment. *Journal of Field Robotics*, 39(5):528–542, 2022. 110

[245] Yuliang Xiu, Jinlong Yang, Xu Cao, Dimitrios Tzionas, and Michael J. Black. ECON: Explicit Clothed humans Optimized via Normal integration. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 512–523. IEEE, 2023. 75

[246] Hongyi Xu, Eduard Gabriel Bazavan, Andrei Zanfir, William T. Freeman, Rahul Sukthankar, and Cristian Sminchisescu. GHUM & GHUML: Generative 3D Human Shape and Articulated Pose Models. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 6183–6192. IEEE, 2020. 75

[247] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. ViTPose: Simple Vision Transformer Baselines for Human Pose Estimation. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. 2

[248] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. ViTPose++: Vision Transformer Foundation Model for Generic Body Pose Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(2):1212–1230, 2023. 3, 7, 139, 141, 161

[249] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997. 25, 35

[250] Gengshan Yang and Deva Ramanan. Learning To Segment Rigid Motions From Two Frames. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 1266–1275. Computer Vision Foundation / IEEE, 2021. 109

[251] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024. 173

[252] Yuxiang Yang, Junjie Yang, Yufei Xu, Jing Zhang, Long Lan, and Dacheng Tao. APT-36K: A Large-scale Benchmark for Animal Pose Estimation and Tracking. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural*

*Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. 151, 152

[253] Zhitao Yang, Zhongang Cai, Haiyi Mei, Shuai Liu, Zhaoxi Chen, Weiye Xiao, Yukun Wei, Zhongfei Qing, Chen Wei, Bo Dai, Wayne Wu, Chen Qian, Dahua Lin, Ziwei Liu, and Lei Yang. SynBody: Synthetic Dataset with Layered Human Models for 3D Human Perception and Modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 20282–20292, October 2023. 76

[254] Shaokai Ye, Alexander Mathis, and Mackenzie Weygandt Mathis. Panoptic animal pose estimators are zero-shot performers. *ArXiv preprint*, abs/2203.07436, 2022. 137, 140

[255] Chao Yu, Zuxin Liu, Xin-Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1168–1174, 2018. 107, 109, 127

[256] Hang Yu, Yufei Xu, Jing Zhang, Wei Zhao, Ziyu Guan, and Dacheng Tao. AP-10K: A Benchmark for Animal Pose Estimation in the Wild. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. 137, 151, 152

[257] Andrej Zdešar, G. Klančar, and I. Škrjanc. Vision-based Localization of a Wheeled Mobile Robot with a Stereo Camera on a Pan-tilt Unit. In *ICINCO*, 2019. 28

[258] Jiacheng Zhang, Mingyu Gao, Zhiwei He, and Yuxiang Yang. DCS-SLAM: A Semantic SLAM with Moving Cluster towards Dynamic Environments. In *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1923–1928, 2022. 107, 109

[259] Ji Zhang, Sanjiv Singh, et al. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9. Berkeley, CA, 2014. 2

[260] Shuang Zhang, Ada Zhen, and Robert L. Stevenson. A Dataset for Deep Image Deblurring Aided by Inertial Sensor Data. *Fast track article for IS&T International Symposium on Electronic Imaging 2020: Computational Imaging XVIII proceedings.*, pages 379–1–379–6(6), 2020. 83, 98, 111

[261] Shishun Zhang, Longyu Zheng, and Wenbing Tao. Survey and Evaluation of RGB-D SLAM. *IEEE Access*, 9:21367–21387, 2021. 107

[262] Tianwei Zhang, Huayan Zhang, Yang Li, Yoshihiko Nakamura, and Lei Zhang. FlowFusion: Dynamic Dense RGB-D SLAM Based on Optical Flow. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7322–7328, 2020. 107

[263] Xu Zhang, Wen Wang, Zhe Chen, Yufei Xu, Jing Zhang, and Dacheng Tao. CLAMP: Prompt-based Contrastive Learning for Connecting Language and Animal Pose. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pages 23272–23281. IEEE, 2023. 141

[264] Jianhao Zheng, Daniel Barath, Marc Pollefeys, and Iro Armeni. Map-adapt: real-time quality-adaptive semantic 3D maps. In *European Conference on Computer Vision*, pages 220–237. Springer, 2024. 14

[265] Jia Zheng, Junfei Zhang, Jing Li, Rui Tang, Shenghua Gao, and Zihan Zhou. Structured3D: A Large Photo-realistic Dataset for Structured 3D Modeling. In *Proceedings of The European Conference on Computer Vision (ECCV)*, 2020. 74

[266] Fangwei Zhong, Sheng Wang, Ziqi Zhang, China Chen, and Yizhou Wang. Detect-SLAM: Making Object Detection and SLAM Mutually Beneficial. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1001–1010, 2018. 107, 109

[267] Yuanhong Zhong, Shuangshuang Hu, Guan Huang, Long Bai, and Qimin Li. WF-SLAM: A Robust VSLAM for Dynamic Scenarios via Weighted Features. *IEEE Sensors Journal*, 22(11):10818–10827, 2022. 3, 110, 128, 129

[268] Boyu Zhou, Yichen Zhang, Xinyi Chen, and Shaojie Shen. FUEL: Fast UAV Exploration Using Incremental Frontier Structure and Hierarchical Planning. *IEEE Robotics and Automation Letters*, 6(2):779–786, 2021. 87

[269] Silvia Zuffi, Angjoo Kanazawa, T. Berger-Wolf, and Michael J. Black. Three-D Safari: Learning to Estimate Zebra Pose, Shape, and Texture From Images "In the Wild". *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5358–5367, 2019. 138

[270] Silvia Zuffi, Angjoo Kanazawa, and Michael J. Black. Lions and Tigers and Bears: Capturing Non-Rigid, 3D, Articulated Shape From Images. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3955–3963. IEEE Computer Society, 2018. 137, 140

[271] Silvia Zuffi, Angjoo Kanazawa, David W. Jacobs, and Michael J. Black. 3D Menagerie: Modeling the 3D Shape and Pose of Animals. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5524–5532. IEEE Computer Society, 2017. 103, 137, 139, 140