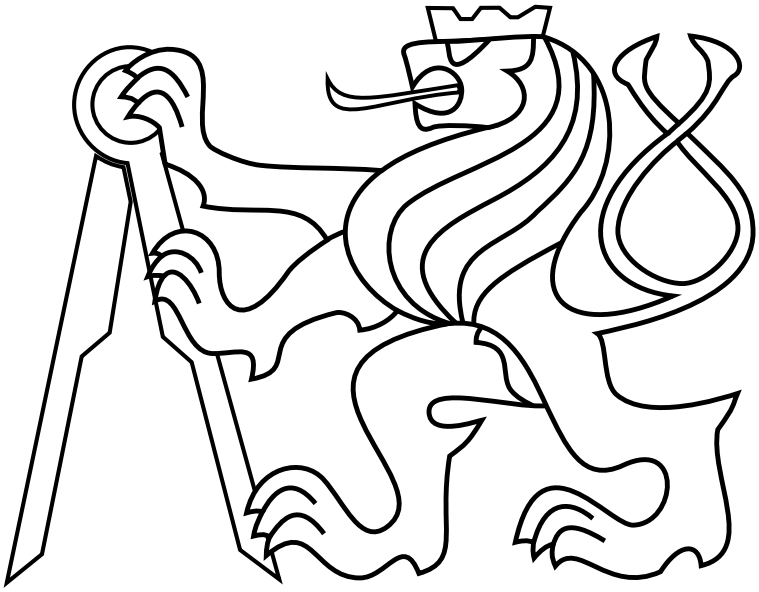
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Robust Vision-Based Navigation in Extreme Environments Inspired by the Hippocamal-Entorhinal System

Tomáš Musil

Thesis supervisor: **Ing. Matěj Petrlík**

Department of Cybernetics

January 2024

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Musil Tomáš** Personal ID number: **483430**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Robust Vision-Based Navigation in Extreme Environments Inspired by the Hippocamal-Entorhinal System

Master’s thesis title in Czech:

Robustní vizuální navigace v extrémních prostředích inspirovaná hippokampem a entorhinálním kortexem

Guidelines:

1. Review the state of SLAM and its connection to biological spatial cognition, cognitive maps and current AI trends [1]. Discuss the open problems not solved by current SLAM algorithms.
2. Construct a simulator for benchmarking multi-session vision-based navigation [2] of autonomous agents in large-scale, 3D, changing environments with high perceptual aliasing, dynamic objects and weather effects. Take inspiration from existing simulators [3].
3. Develop a robust SLAM-inspired vision-based spatial perception and action system that can navigate in the extreme environments of the simulator.
4. Discuss the performance of the designed system and compare it to existing spatial perception systems [4].

Bibliography / sources:

[1] Safron, Adam, Ozan Çatal and Tim Verbelen. “Generalized Simultaneous Localization and Mapping (G-SLAM) as unification framework for natural and artificial intelligences: towards reverse engineering the hippocampal/entorhinal system and principles of high-level cognition.” Frontiers in Systems Neuroscience 16 (2021).
[2] Anderson, Peter, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva and Amir Roshan Zamir. “On Evaluation of Embodied Navigation Agents.” ArXiv abs/1807.06757 (2018).
[3] CARLA - Dosovitskiy, Alexey, Germán Ros, Felipe Codevilla, Antonio M. López and Vladlen Koltun. “CARLA: An Open Urban Driving Simulator.” Conference on Robot Learning (2017).
[4] Campos, Carlos, Richard Elvira, Juan J. G'omez Rodr'iguez, 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 (2020): 1874-1890.

Name and workplace of master’s thesis supervisor:

Ing. Matěj Petrlík Multi-robot Systems FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **18.09.2023** Deadline for master's thesis submission: **09.01.2024**

Assignment valid until: **16.02.2025**

Ing. Matěj Petrlík	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

I would like to thank my family and friends for supporting me through the hard times of my studies and while writing this thesis. They have always been there for me and I cannot thank them enough for that.

I will also be forever grateful to all the amazing members of the Multi-Robot Systems Group, as without them, I this thesis would by no means be possible. They made me truly fall in love with robotics. At MRS, I was given the ability and the support to work amongst brilliant roboticists on the hard problems in robotics, and they have always been supportive of me and helped me learn. They are truly great people, and I want to specifically thank Ing. Matěj Petrlík for guiding me while writing this thesis and our previous works. He has been the best research advisor anyone could ever hope for.

Abstract

This thesis deals with bio-inspired monocular visual navigation and is divided into three main parts. First, we survey the research in cognitive neuroscience on human and animal navigation, compare it with the currently best robotic vision-based navigation systems, and identify core strengths of biological navigation which could be worth replicating in robotics. Second, we present a novel simulator that we purpose-build to simulate challenging scenarios and find the limits of existing vision-based systems, which often assume small-scale, static, and unambiguous environments. We also propose a method of evaluating navigation holistically, using the simulator. Third, we present a new vision-based autonomous navigation, map-building and exploration approach. We also show that by using large-scale spatial geometry instead of visual appearance, one can achieve robust multi-session localization even in a highly perceptually aliased environment. Finally, we demonstrate the exploration and safety-aware planning of the designed system both in simulation, and in the real world on an inexpensive unmanned aerial vehicle with limited sensing capabilities.

Keywords: navigation, vision-based navigation, exploration, cognitive science, hippocampus, entorhinal cortex, unmanned aerial vehicle, SLAM

Abstrakt

Tato diplomová práce se zabývá bio-inspirovanou robotickou vizuální navigací a je rozdělena na tři hlavní části. První částí je porovnání navigace zvířat a lidí oproti navigaci v aktuálních robotických systémech a krátké shrnutí výzkumu na téma navigace lidí a zvířat. Druhá část je o novém simulátoru, který jsme navrhli tak, aby v něm výzkumníci mohli rozbíjet aktuální robotické systémy založené na počátačovém vidění. Dále také představujeme novou metodu vyhodnocování navigace jako celku, pomocí navrženého simulátoru. Poslední částí je systém pro vizuální navigaci, mapování a prozkoumávání, spolu s metodou hledání korespondencí mezi více mapami, která pro tyto účely používá geometrii velkých úseků prostředí. Funkcionalitu tohoto systému demonstrujeme na experimentech v simulovaném i reálném prostředí.

Klíčová slova: navigace, vizuální navigace, prozkoumávání, kognitivní vědy, hipokampus, entorinální kortex, bezpilotní helikoptéra, SLAM

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

Humans and animals can navigate both known and unknown physical space — a city, a forest, or the open sea, efficiently and reliably over many sessions, without relying on any global positioning systems. That, however, cannot be said for today's mobile robots. Through decades of research, SLAM and autonomous navigation have come a long way towards systems that are as good at navigating and spatial reasoning as biological ones, but there still remain many unsolved fundamental questions

The potential impact of building inexpensive systems that can reliably navigate any given environment is immense. From agriculture robots that would work sprawling fields to swarms of low-cost drones that would search for survivors during a natural disaster, all robots that operate in the real world require reliable and efficient systems for knowing where they are, where they have been, and how to reach any given relevant place.

The main question of this thesis is thus **"How can we build artificial systems that could navigate real-world, large-scale, dynamic environments as reliably and effortlessly as animals?"** As an example, we propose to take the egyptian fruit bat, described in the following section, as the target for vision-based navigation systems.

1.2 100km wide Cognitive Maps and Repeatable Navigation in Bats

The authors of [1] documented that the Egyptian fruit bat is capable of returning to its nest after being translocated by up to 80km in arbitrary directions from it. After a thorough dicsussion in the paper, the authors conclude that the bat most likely relies primarily on visual information in such navigation, likely using faraway mountain ranges as its primary visual cue. This is an incredible feat of navigation, considering that single-robot missions without global position systems have so far reached only up to a few kilometers, and that is most often using powerful depth sensors such LIDARs [24].

Additionally, the fruit bats are capable of repeatedly navigating to one particular fruit tree, 15km away from their nest, night after night [1]. Such repeatability is also a very impressive and inspiring achievement, compared to how difficult it is to realize repeatable robotic navigation even in a very controlled small-scale setting.

Bats possess superb sensory systems [2] – vision; somatosensation which helps with flight control and path integration; echolocation which helps them sense distance, relative velocity, shape and texture of objects up to approx. 17 meters; magnetoperception which helps them obtain their rotation relative to Earth's magnetic field; and sensitive olfaction which helps them distinguish many different smells. However, aside from olfaction and the sensitivity of their echolocation system, this sensory equipment is not that different from what one can achieve on a robot with an IMU, global shutter cameras, and compass sensors. Thus, it might be possible to replicate such large-scale navigation and map building even with existing sensory and computational technology.

1.3 Contributions and Outline

In this thesis, we focus on searching for answers as to how we could achieve such incredible navigation in robotic systems. The thesis is divided into three main parts:

In chapter 2, we analyze the fields related to navigation used in robotics today, make a brief survey of the current research on mammalian navigation, and then attempt to find comparisons and potential inspiration that can be taken from biology. We also briefly touch on how navigation might be a core component of higher-level cognition in general and why that makes it that much more interesting to study.

In chapter 3, we present a new simulator that we built to assist researchers in developing robust navigation systems, and describe a new replicable benchmarking method that can be used to evaluate navigation as a whole, using the simulator.

In chapter 4, we present a novel visual-inertial navigation system that takes heavy inspiration from biology, combines it with the best available modules for robotic navigation,

and thus achieves multi-session navigation and exploration on a UAV platform in the new simulator and in the real world.

Chapter 2

Artificial versus Biological - A Navigation and Mapping Survey

Contents

2.1 Biological Navigation Types and their Robotic Counterparts

In cognitive science, navigation is commonly divided into the below mentioned categories, as summarized for example in [2]. Animals are thought to use all of these behaviors and to switch between them according to need. The categories are:

- **Beaconing** is the simplest navigation type where an animal navigates directly towards a visual feature that it can currently see. This is reflected in **visual servoing** [3], where a system detects, tracks and then aligns itself to move towards a particular goal object or area. This is a fairly mature field and applications exists for example in military settings.

- **Route Following** is when an animal remembers a route to a given place and then follows that route. This is best reflected in **teach-and-repeat navigation** [4, 5],

which is today also a mature field with real-world robotic applications where it is sufficient to follow unchanging paths between places.

- **Path Integration** is the ability of animals to integrate self-motion cues (e.g. acceleration sensing, counting steps, rotations etc.) and visual measurements to estimate its traveled path in an environment. This corellates to **odometry estimation** [6, 7] methods, which are a critical part of any mobile system and vary for different sensory setups. An example of this are ants, which when kidnapped and moved away from their position, will execute the same route they traveled in an attempt to return to their nest and thus not reach the nest.

- **Map-based navigation** is the most advanced level of navigation, where an animal constructs a *cognitive map* and uses it to navigate. What differentiates this strategy from others is the ability of animals to navigate to a goal even when their preferred path is blocked or to use shortcuts in an environment to use paths that haven't been traveled before. The observation of this behavior in rats led Tolman in 1948 to propose [8] the existence of cognitive maps for navigation, and since then, researchers have discovered many systems in mammalian brains that support this hypothesis, which we discuss more in subsection 2.3.1. In robotics, this corresponds to systems that perform **Simulatenous Localization and Mapping (SLAM)** and employ **path planning** and **path following** methods to navigate in their created maps. We will focus on this type of navigation in this thesis.

2.2 Navigation and Mapping in Robots

2.2.1 Simulatenous Localization and Mapping (SLAM)

SLAM is a wide research field that focuses on the chicken-and-egg problem of simultaneously building a map and localizing in it. To localize within a map, one needs a map, and to update a map with new measurements, one needs to localize oneself relative to the map. Thus arises the difficulty in this task. The authors of [9] thoroughly survey the history, current state and open questions of SLAM, and the text in this section is our attempt at summarizing the parts that are relevant for comparison with biological spatial intelligence.

The most common way of thinking about SLAM is the graph-based, probabilistic approach [10, 11]. In that paradigm, the problem is generally formulated as finding robots' trajectories and locations of observed spatial objects (points, lines, surfaces etc.) in a single coordinate frame from a number of sensory measurements (cameras, range sensors, IMUs, magnetometers, barometers etc.).

Several algorithms can solve (in a sense that the output is useful for spatial navigation tasks) the SLAM problem in specific environments in real time. The two most common

ones are *smoothing* and *filtering*. Smoothing approaches, the most popular being iSAM [12, 13], represent measurements, observations, and prior knowledge as factors that constrain the unknown states in a factor graph. The smoothing process finds the Maximum A Posteriori (MAP) estimate of the probabilistically constrained states based on the uncertainties of the factors using nonlinear optimization methods such as the Gauss-Newton or Levenberg-Marquardt algorithms. Because with increasing time the factor graph grows and optimizing the whole graph would become intractable in real time, smoothing approaches often *marginalize* states and measurements, meaning that they only optimize variables in some sliding window and previous variables outside the window become fixed. Such approaches are called fixed-lag smoothing. When the size of the window accomodates only a single state, then we speak about filtering. Filtering approaches most often describe the full robot state and measurements using a Kalman filter [14]. These are generally more computationally efficient, but less accurate.

Localization without global pose sensors (such as GNSS, compass or uniquely identifiable landmarks with known positions) will always encounter odometry drift – the estimated pose relative to the specified coordinate frame will accumulate errors and become increasingly different from the true pose. To solve this, SLAM systems use *place recognition methods*, which are well summarized by the authors in [15]. This usually means that there is a thread that checks whether the current context (camera image, lidar scan, ...) is similar enough to some previously seen context. If it is similar enough based on some metric, the existing trajectory and map are deformed (often using pose-graph optimization) so that the current pose aligns with the pose of the previously visited context.

Visual place recognition is a large research field in itself, and many advances have been done in the last decades to make place recognition efficient, accurate and thus useful for SLAM and other applications [15]. Deep learning approaches have been strongly researched in the recent years and offer good solutions to this problem [16, 17]. The majority of visual SLAM algorithms [18, 19] however use a different method – describing features in a given image using descriptors such as SURF, SIFT or ORB descriptors without the need of a GPU, and then converting each image into a bag of visual words [20]. This bag-of-words database is extremely fast to query for similar images, and usually the SLAM algorithms keep the positions and descriptors of each image, so that when a word-level match is found, they also compute a homography between the images using the features and see if they reproject well between the images, often called geometrical verification.

There is only a handful of SLAM systems that do not use the paradigm of estimating a metrically accurate trajectory and landmark positions. The most noteworthy are those using different variants of continuous attractor networks, inspired by the cells of the hippocamal-entorhinal system, described more in subsection 2.3.1. A detailed survey of these systems is available in [21], but the most influential one is RatSLAM [22, 23]. RatSLAM takes heavy inspiration from biological cognitive maps, using a pose cell attractor network instead of traditional odometry. They describe spaces through a "local view network" that injects activity into the pose cell network. The authors demonstrate a remarkable loop closing ability of this method on the KITTI dataset, and the ability of a

system using RatSLAM to navigate repeatedly in an office environment. One disadvantage of that system, however, is that in it too, wrong loop closures can lead to the map being deformed, and lead to navigation failures.

2.2.2 Open Questions in SLAM relevant to Navigation

SLAM is usually the backbone of many real-world mobile robot applications [24, 25] and we rely on it for building other higher-level spatial representations such as for occupancy [26] or semantics [27]. Most of the above mentioned approaches work quite well in enabling mobile robots to navigate environments in short-term missions. However, there are still many open problems, which need to be addressed if we want to have systems that can navigate as reliably and effortlessly as humans or fruit bats described in section 1.2.

The authors of the survey [9] in 2016 have already covered the open questions in SLAM, the vast majority of which are still relevant today [28]. Thus, we analyze the following open problems related to navigation specifically:

- **Handling SLAM failure** – Nearly all navigation software stacks make one very strong assumption – that SLAM works. If SLAM loses tracking (in case of LiDAR odometry, for example in a smooth corridor) or performs a wrong loop closure, such systems break down completely (e.g. the robot will explore an area forever, thinking it is always a new part of the environment, or crash into obstacles due to map misalignment). We hypothesize that this is the main reason why reliable vision-based navigation systems are still not commonplace in the world, because in visual SLAM especially, it is very easy to lose tracking due to fast motions, textureless areas, low illumination and other problems. For LIDAR-based SLAM, some recent approaches [29] model environment degeneracy and can avoid such breakdown situations to some extent, but there is still much more work to be done, especially in vision.

- **When to gather more information and when to act on the most likely hypothesis?** – This is a question important to answer in any kind of action systems where there is perceptual aliasing in the environment, which happens very often in the real world. Active SLAM methods [30] are a modern class of methods that deal precisely with that question in the context of creating a good map. Most notable is ARAS [31], where the authors present a system which explicitly models ambiguity and actively revisits mapped areas to reduce possible hypotheses. For the context of navigating, we can take inspiration from such methods, and the methods we design for this problem will be shaped by the specific navigation task at hand.

- **Scalable Spatial Representations** – The navigation of bats through both 100km wide areas and tight cave systems is still unthinkable today, as there is no existing system that could navigate *both* such large scale and small scale environments. For that, we need spatial representations that can adapt well to widely different scales and

describe space in a compact and actionable way, as the authors of [9] put it. A point cloud or a voxel occupancy grid is almost certainly an unsuitable way to describe the entire world for the purpose of navigation. In our previous work [26], we have made a small contribution in this direction, by describing free space with spheres, so that tight spaces are described with a high density of spatial elements (spheres) and large open areas can be described with a few spheres, and the representation does not suffer from the many problems of grids. It remains an open question worth pursuing how to give the navigation systems the ability to describe space efficiently for navigation. Additionally, scalability in time – i.e. handling large-scale maps over many sessions and doing computations such as detecting changes between sessions – is also still not solved.

- **Using semantic knowledge in navigation** – Humans use semantic knowledge when navigating, along with episodic knowledge and maps [32]. For example, one can detect that some patch of ground is muddy and avoid it, or know that a door is a door, and thus it can be opened to reach some area. Such knowledge is usually hardcoded into the spatial perception and planning systems for each such chunk of semantic knowledge. It remains a very difficult question how such knowledge could be used in navigation, for example for clustering of large parts of space, so that robots can for example describe a forest not as a multiple-gigabyte point-cloud, but as for example "a forest with approx. such density, of approx. circular shape, with notable large-scale ridges and hills here and here" or "a room with many books relative to other rooms". Modern spatial perception systems [27] for example label all surfaces with their labels given by a visual semantic segmentation system, but how to use such knowledge remains largely unexplored. Additionally, in [27], the authors propose a way of clustering space in real time into rooms, hallways, floors and buildings, which can definitely be a useful method of abstraction for fast high-level planning. However, the way these abstractions are constructed is hard-coded, while animals and humans seem to create these spatial clusterings very flexibly both in physical space and in conceptual spaces as well [33]. It is thus another interesting problem how to engineer these efficient abstractions, and a more difficult question, how could such efficient abstractions be created autonomously without needing to be hard-coded.

2.3 Navigation and Cognitive Maps in Mammals

As we discussed in section 1.2, animals possess remarkable navigational capabilities. In the recent decades, researchers have provided high amounts of insight into the behavior of regions in mammalian brains (mostly of rodents and bats navigating in small arenas, but recently also of long-range flying bats [1, 2, 34]) related to navigation and spatial reasoning.

The most important of these regions is the hippocamal-entorhinal system (HES) [CITE-HES OVERVIEW], found to function similarly in mammals such as humans, mon-

keys, bats, and rats. In brief, it consists of the hippocampus, entorhinal cortex and surrounding regions, and damage to this part of the brain results in severe deficiencies in navigation and memory abilities. In humans, the importance of this region to memory-related functions was best documented thanks to the patient known as H.M [35],

2.3.1 HES Cells Identified for Spatial Cognition

The most commonly investigated cells in the HES related to spatial cognition are the place cells, grid cells, border cells and head direction cells, illustrated in Figure 2.1. In this section, we briefly explain these cells' behaviors and discuss the current understanding of their interactions and significance to spatial navigation, and also mention other cells encoding spatiotemporal concepts in the HES.

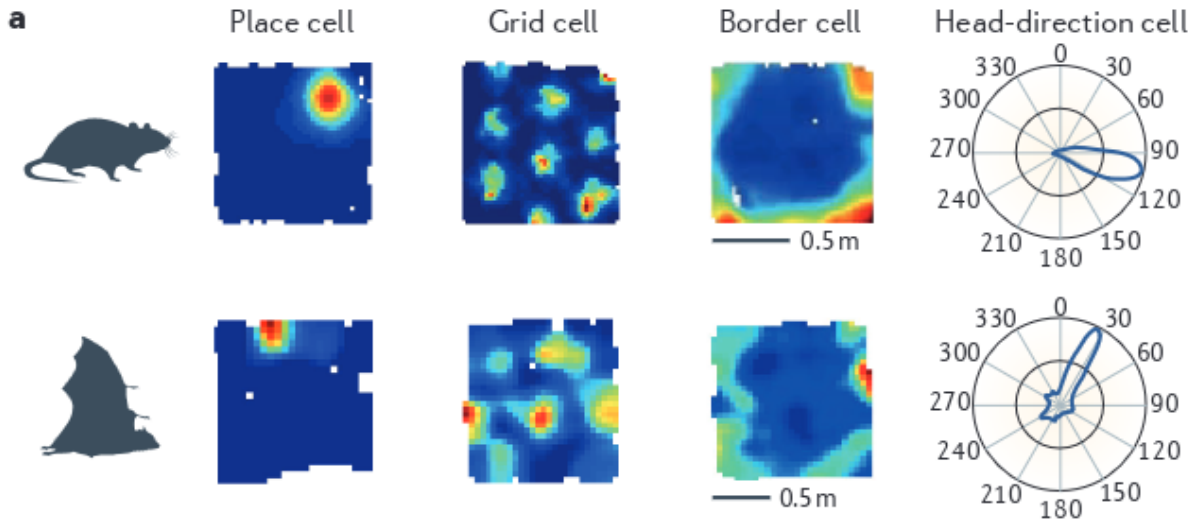


Figure 2.1: The primary entorhinal cell types for spatial cognition found in the HES, found both in bats and in rats. Source: [2]

Place Cells

In 1971, place cells were discovered in rat's hippocampi [36], which served as a key piece of evidence towards confirming Tolman's hypothesis that the hippocampus may serve for construction of a cognitive map [8]. A place cell is a neuron that fires only in a specific area of an environment, called a place field of that particular neuron. These firing fields can vary in size, and any particular place cells can also fire in other places in different environments, but in one given environment, it always fires in the same place. In some situations, they are heading-agnostic (meaning that they fire regardless of which way the rodent is oriented), mostly in open areas, but in some situations, they are heading-specific,

meaning that the cell fires only if the rodent is in a given place *and* aligned in some direction, which mostly happens in long corridors, where it is important for navigation to know which direction the rodent is coming from.

Grid Cells

Another major discovery was that of grid cells in 2005 [37]. A grid cell is a neuron in the entorhinal cortex that fires in a regular triangular pattern across the entire environment, best described by the firing field in Figure 2.1. Rodents have many of these cells, each with a slightly different offset and scale. If you imagine many of these hexagonal patterns with different offsets and scale for each cell, and know which of the cells are firing at a given time, then by seeing the overlaps of the active cells, you could determine where in the environment a rodent is located. Thus, it is mostly agreed that the main purpose of grid cells is that they are somehow important in path integration and provide a coordinate system for rodents (and of bats, even though in bats, a firing field does not form one perfect grid, but is composed of many smaller grids [38]). They are interlinked with place cells and head-direction cells, and continue to fire even in total darkness, which likely means that rodents are integrating their self-motion sensory inputs, as a robot might do using wheel encoders. As with place cells, the firing field of a grid cell remaps when entering a topologically distinct part of the environment, e.g. going through a doorway, but is stable inside any given environment.

Border Cells

In 2005, border cells (also known as boundary vector cells) were discovered [39]. A given border cell fires when there is an environmental boundary at a specific distance and direction from the animal, and as with the place and grid cells, an animal usually has many of them with different direction and distance sensitivities. Border cells are sensitive to boundaries that prohibit motion – walls, cliffs alike, but not for example to colored lines on the ground.

Head-direction Cells

A head-direction cell is a neuron find in multiple brain areas, which fires when the animal's head is pointing in a specific direction, not relative to the body, but relative to the environment, similar to a compass. They continue firing even in total darkness, suggesting that their firing is determined by a combination of self-motion integration and of visual cues.

Other Important Cells of HES and their Interplay

The above mentioned cells have been studied the most in existing literature, but the way they affect one another and form their firing fields is not yet fully understood, even though many models that describe their behavior to some extent have been proposed. We will now mention a few more cells (there are many more) to illustrate how many varied types of cells have been discovered so far, which will be important for the concluding point in subsection 2.4.3.

Object vector cells [40] are a fascinating type of cell that fires when an object is present near the rodent, and each cell has a firing field relative to the rodent's position, regardless of the rodent's orientation, suggesting that the objects are anchored to the rodent's cognitive map. Interestingly, when an object is removed, another type of cell fires with the same place field as a corresponding object vector cell, and these cells are called trace cells. These fire for a vast array of objects, suggesting that the animal represents the presence of *any* distinct object relative to the environment. Such representation of general object presence or removal could be useful to a wide array of behaviors that the rodent might need to perform. This is mentioned due to the fact that many artificial intelligence systems, which deal with object detection or object-based tasks, do not detect presence of *any* object but rather specific objects (e.g. YOLO [41] which is popular in robotics [24]). Thus, it could be useful, for example for reinforcement-learning models, to have preprocessing modules that provide *separate* representations of an object (even if unidentifiable) being present at a location and the type of the object.

Lastly, in bats, when a bat is observing the flight of another bat, cells representing the location of another bat in the environment have been identified [42], which suggests that bats, being social animals, are predicting the trajectory of one another. Such representation can be useful for many types of social interactions.

2.4 Opportunities for HES-Inspired Mapping, Navigation and Intelligence

The cells described in subsection 2.3.1 are just a small part of existing neuroscience research, and there is much more that could serve as inspiration to robotics and artificial intelligence research. In this section, we attempt to find some take-home messages from the research surveyed in this thesis and propose ideas that work well in biology and might be worth exploring more in robotics and artificial intelligence research.

2.4.1 Geometry or Visual Appearance as Stable Descriptors?

A mostly unexplored idea in visual navigation systems is using large-scale spatial geometry for place recognition instead of visual apperance. Yes, indeed, the method of

geometric verification is commonplace in place recognition systems, but that usually means reprojecting visually distinct features between images and seeing if they reproject near other features given some putative transformation between images. To the author's best knowledge, current vision systems do not utilize information about a more general shape of the environment as might be the case with 3D scan matching, often used in place recognition in LiDAR-based systems.

In biology, there is significant evidence that animals utilize the overall geometry of an environment to reorient and might not use information about textures at all. The most prominent experiment is [43] where researchers were putting a rodent into a rectangular box, and putting food in one of two opposing corners. Of course, if the walls were the same, and the animal was rotated randomly, it had a 50% chance of going into the correct corner to get the food. But strikingly, if one of the shorter walls was painted with a pattern and the food was always put to the corner left of the pattern, the rodent still continued to randomly pick one of the corners, so it did not use the information about the patterned wall at all. In the same study, human babies were found to perform the same inefficient search behavior as the mature rats, not using the non-geometric cues until the age of approx. 21 months.

A possible hypothesis as to why geometry is so important is that large-scale spatial geometry is *stable*. Intuitively, and with evidence from neuroscience [44], the larger an object or contour is, the more stable it usually is, and thus it is more useful and trusted as a landmark. For example, for large-scale navigation, even if a person is lost while hiking in for example a valley or mountain range, they can always walk for some time to gauge the shape of the valleys and mountains around them to reorient and navigate.

An interesting direction to research is thus finding ways of encoding and using this large-scale geometry, likely from time sequences rather than single camera or LiDAR measurements, to enable more efficient and reliable place recognition. The closest research to this is the iterative closest point (ICP) algorithm [45, 46] or newer scan-matching algorithms [47], which are often used in LiDAR-based SLAM. Another reasonable step in this direction is adapting these scan-matching algorithms to maps built on vision-based systems. A possible reason this has not been done is that most Visual SLAM systems build *sparse* maps, which are not very suitable for scan matching algorithms.

2.4.2 Multiple Maps and Reference Frames

The firing fields of the cells mentioned in subsection 2.3.1 are not globally stable. They *remap* in different environments or parts of the environment. This strongly suggests that **mammals do not describe the world by one metrically accurate map in a single coordinate frame**, but instead they construct (and potentially reuse) smaller submaps, where the metrically accurate SLAM problem is computationally easier to solve. The authors of [48] go into detail of this phenomenon, make arguments about the potential reasons of such fragmentation and demonstrate on a robot in 2D with a LiDAR how such

fragmentation can speed up path planning times. We find this phenomenon as a strong argument towards building and matching submaps instead of having one big map when constructing spatial navigation systems and build the system described in chapter 4 based on this idea.

2.4.3 Task-Specific Cognitive Maps and Higher Level Cognition

With the immense amount of discoveries of cells that encode different spatial concepts in HES, researchers are proposing interesting theories of how exactly the HES might be a crucial element in not just spatial map making, but also higher level cognition and abstract space reasoning. For detailed summaries of evidence and existing research in exploring this idea, we refer the reader to the section on the HES in [49] and the final section of [50]. We will now briefly summarize the arguments in these works, which have also been discussed in recent talks from researchers working at the neuroscience-AI intersection.

The most striking point seems to be that there has been recent evidence that the cells of HES do not encode only physical space, but abstract spaces as well. For example in [51], authors found signals similar to grid-cell signals in humans in a task where they had to make decisions about pictures of birds whose shapes were encoded by 2 variables – their neck length and legs length. The researchers found signals that correlate with these 2 dimensions, and they found them in the area related to spatial navigation. Theories that try to explain this kind of reasoning are proposed for example in [52]. In addition, in the area which houses place and grid cells, researchers have discovered many other cell types, which encode abstract, nonspatial contexts relevant to a particular given task – e.g. splitter cells [53, 54] which encode whether the rat is on an even or odd experiment when the researchers switched the food's location between two places on each experiment, time cells [55] which represent time when the task requires the rat to track time, or frequency cells [56], which describe chunks of frequencies similarly to place cells and were found to emerge in a task where rodents have to push levers to move the frequency coming from a speaker up or down so that it matches a previously heard frequency. This suggests that the HES is capable of building different types of cognitive maps that efficiently and flexibly encode important contexts and their relationships relevant to any given task.

In this line of research, several models have been developed in which there is emergence of cells that encode space and time similarly as cells in the HES, which is important because the emergence of such cells in biological systems is not yet fully understood. The most notable is the Tolman-Eichenbaum Machine (TEM) [57]. It is also very important to note that TEM is mathematically equivalent to the currently very popular transformer network [58].

We conclude this chapter by proposing that this navigation-and-mapping-based view of artificial intelligence might be very interesting to explore given the recent advances of SLAM, autonomous systems, artificial intelligence and cognitive science research. This is of course entirely outside the scope of a master's thesis, and thus the rest of this thesis

is focused more on implementing the insights described in subsection 2.4.2 and subsection 2.4.1.

Chapter 3

Simulating and Benchmarking Vision-based Navigation in Extreme Environments

Contents

3.1 Why another Simulator?

To test *navigation* holistically, one needs to evaluate the result of a system acting upon an environment, not just to passively process datasets. To evaluate robustness or reliability, one needs to test the system in multiple sessions under many different conditions, e.g. lighting conditions, environment object, appearance or topology changes. It is, however, virtually impossible to create repeatable conditions for such multi-session navigation experiments in the real world, and thus a simulator is the logical choice for such evaluations. For this reason, we constructed and open-sourced the HARDNAV [1] simulator. We have presented this simulator at a poster session of a workshop at IROS2023 [2] and currently, it is being used [3] by at least one PhD student at CTU for researching visual

[1] https://github.com/MrTomzor/navigation_unity_testbed
[2] https://mrs.felk.cvut.cz/hardnav
[3] https://github.com/Zdeeno/vtr-sim-ws

teach-and-repeat navigation.

There are many simulators available for robotics research, but most of them are strongly specialized for a particular domain (e.g. for cars only [59], for UAVs only [60] indoor environments only [61] etc.) and often are difficult to modify for other domains/tasks. Furthermore, in most of them, it would be difficult to set up systems for automatically resetting the simulated worlds to configurations defined by a particular benchmark and evaluating the reaching of goals specified by the benchmark.

For these reasons, we formulated a *new method of benchmarking multi-session visual navigation* and constructed a simulator that supports such benchmarking. The simulator allows user-friendly world state modification and resetting through a ROS service call (or, in the future, through an AI Gym interface), and also features minimalistic general-purpose robotics infrastructure, so that *any* robots or world behaviors can be easily added into the simulator.

3.2 HARDNAV Simulator Description

3.2.1 Unity Engine for Robotics Simulation

The core of HARDNAV is the Unity game engine, which offers physics, rendering and a GUI called the Unity Editor for editing scenes and objects in them. As of now, Unity is free of charge for users who earn less than some arguably high amount of money from their use of Unity, and has been used for many robotics projects in the past [60]. HARDNAV itself is then an open-source collection of Unity scripts, worlds, release builds, and accompanying ROS packages for connecting Unity and ROS, and for the robust navigation benchmarks. Users can interact with HARDNAV either by downloading a compiled build (which allows customizing the world through the world config service described in subsection 3.2.3) or by opening the project in the Unity Editor, which allows full modification of worlds, robots etc.

To briefly introduce how Unity is used - each Unity *project* is made up of *scenes* and *GameObjects* within them. Each *GameObject* has a position and rotation in the world, and can contain any number of *Component* instances. A *Component* is for example a mesh rendering component, collider component, rigidbody component, or a user-defined component specified by a C# file. For example, to implement a simple velocity controller, a user would create a new C# file in the project's filesystem and attach an instance of it to the objects they want to move. They would define parameters such as movement force as variables which can be modified in the GUI for each component's instance, and then write the calls for reading user keyboard input and applying forces to a rigidbody component based on the input into the FixedUpdate() function in the script, which gets automatically called on every physics update of the engine. This is the main structuring of Unity projects. Note that there are no explicit game-related definitions, because Unity

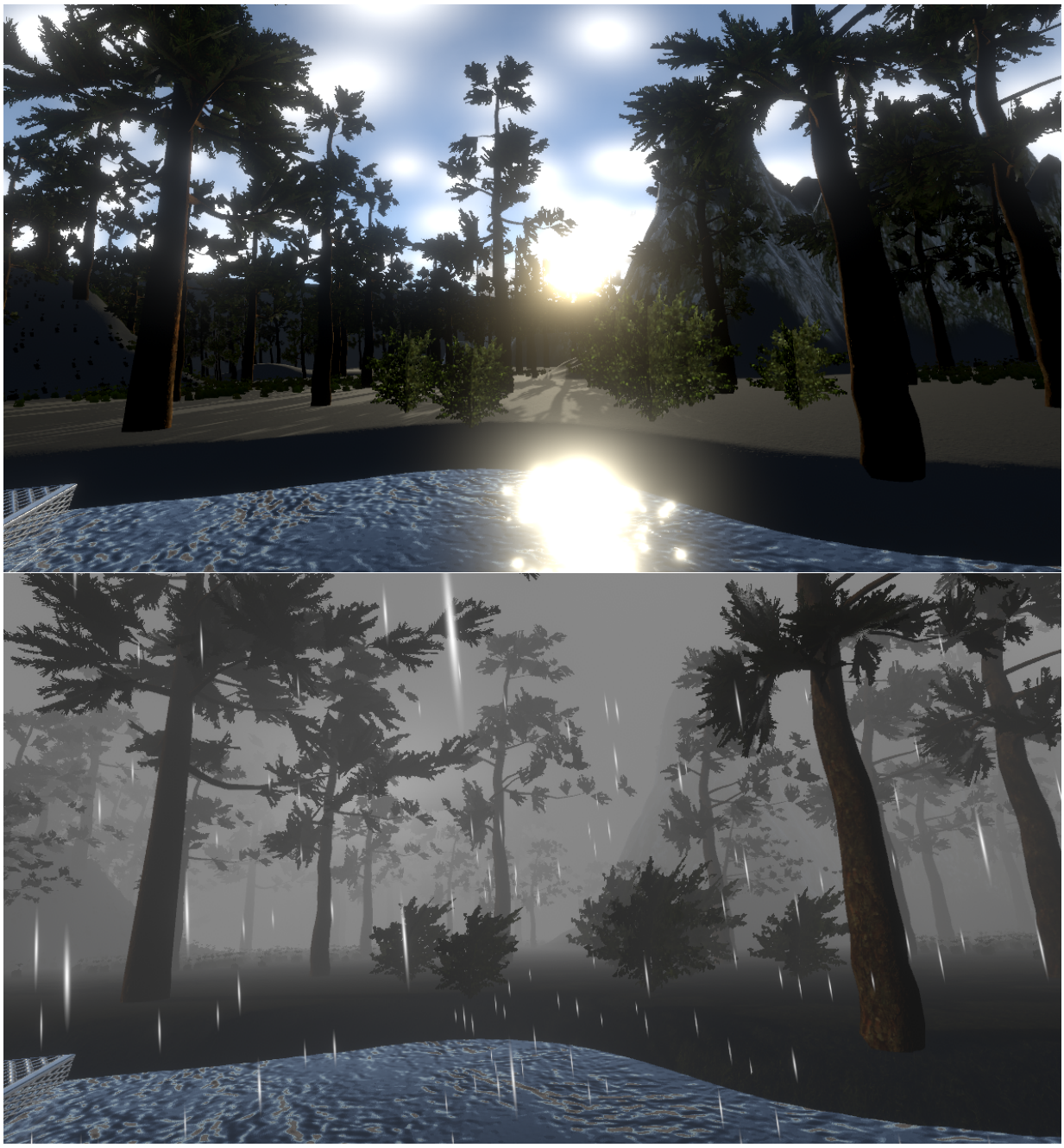


Figure 3.1: An illustration of the possible changes in the environmental conditions, modifiable through calling the World Config ROS service described in subsection 3.2.3. Note that in the upper image, the sun is shining very bright and reflecting on the water. In the second image, fog and rain are turned on, sunlight is limited and the large wall and mountain in the back are nearly invisible. Also notice how in the second image, the trees and bushes are bending to the right due to the simulated wind.

is more of a "sandbox" than only a game engine compared to its main competitor, Unreal engine, which for example has an explicitly defined "Player" object at all times in each scene, and in that regard, one may argue that Unity is more suitable for robotics simulation purposes.

Unity itself offers many great features as a game engine, and the Unity development team has released several official packages that link Unity and Robot Operating System (ROS), and even a package for machine learning agents which turns any Unity scene into an AI Gym environment. There is also an immense amount of game development tutorials for Unity on the internet, so it is very easy to implement features seen often in games, such as visual effects, particle effects or world generation into Unity by following those tutorials. This is, for example, how we implemented rain, moving water and Compared to this, traditional robotics simulators such as Gazebo do not have such wide community support.

Here are the following main packages provided by Unity that we used in constructing the simulator:

- **ROS-Unity Connection** – is facilitated through the Unity Robotics [4] package on the side of Unity After importing this package in Unity, users can write functionality for ROS subscribers, publishers an messages with virtually the same syntax as in C++ or Python ROS nodes. From Unity, data is sent outside of the Unity process through TCP. This is the main bottleneck in data transfer, but as we demonstrate later, it is sufficiently fast for our purposes. On the other side, the communication is handled by the ROS TCP-Connector node, also provided by the Unity team. All subscribers, publishers and services are then registered under the TCP connector node under topics specified in Unity. This integration is quite easy to work with and even allows even custom message types.

- **URDF Importer Package** – Another great package provided by the Unity team allows importing any robot specified in the popular Universal Robot Description Format (URDF) into Unity. As with the Robotics package, users install this inside Unity, copy their URDF into the project's filesystem, click a button, and the package constructs an array of GameObjects corresponding to robot links in the URDF, with matching physical, visual and collision properties. One disadvantage is that this is not usable on runtime, so when a user wants to add a new robot, they need to open the project in the editor and import their robot with the package. Additionally, the importer package does not support any sensors, because there are no standardized sensor implementations in Unity. Thus, we needed to add the camera and IMU sensor scripts to the robot GameObject manually, but this only needs about 5 minutes of drag-and-dropping the sensor scripts onto the object and setting up the ROS topic names.

[4] https://github.com/Unity-Technologies/Unity-Robotics-Hub

- **Terrain Simulation and Sculpting** – To truly test robots in a variety of scenarios, it would be beneficial to have a way of easily crafting large-scale terrains with different textures, trees, rocks etc. Fortunately, this need is shared in game development, and the Unity Editor has built-in tools for that. Specifically, Unity allows users to sculpt terrain meshes, select tree types, how they are randomized and rendered at different distances (which dramatically speeds up rendering and allows simulation of enormous worlds), and paint trees, vegetation and textures onto the terrain mesh. The Terrain engine also supports wind simulation, meaning that trees and vegetation sway in the wind with different direction and intensity, which could certainly break some spatial perception methods. All the worlds in the simulator were created using these tools in just a few hours.

- **NavMesh Navigation for Dynamic Objects** – Pathfinding and moving agents towards some goal is a universally needed feature in game development. Unity has built-in packages for constructing, even during runtime, navigation meshes. Thus, one can easily implement for example pedestrians or other "distracting" robots that move around the world intelligently, by just specifying several parameters and calling a single function that makes the NavAgent component navigate a GameObject to a given location on the navmesh.

- **Asset Store and Free Assets** – The last feature we want to mention is Unity's asset store. It is not a unique thing, the Unreal engine has one too, and Gazebo has its own library of plugins, but it is important to mention. There are thousands of assets that are free of charge on the asset store and importing them is very simple and well documented. This gives researchers the ability to quickly add new scenes, textures or models into their simulation setups (e.g. the humans package, One downside is that the assets cannot legally be redistributed, and thus when someone clones a Unity project, they need to redownload the assets from the asset store. For this reason, HARDNAV is dependent on only three assets from the store - one for trees and natural textures, one for metal textures and one for working with YAML files.

Despite Unity's great features and support for robotics research, we encountered several difficulties in using it for simulation, which were not documented anywhere, and we needed to find solutions to them in other existing Unity projects:

- **Time step control is primarily for games** – Likely the hardest implementation problem in this chapter was controlling Unity's time steps. Unity has different update loops (rendering, primary update and physics update, among others) and it is difficult to force it to render images at a target rate. However, we have achieved this and the simulator contains a script that overrides a camera object's base functionality and forces it to renderer at a fixed rate relative to Unity's physics time. We chose to use the physics time of Unity as the simulation time and that works reasonably well. With these edits, the simulation of all sensor data is now synchronized, and the current real-time-factor is visualized in the game screen when the simulator is running.

- **No standardized sensor implementations** – Unity does not provide any implementation of a camera, IMU or any other sensor in its base robotics package. They do have an example project with a C# script for simulating a 2D LiDAR, but nothing else. Thus, to implement an IMU and a camera that would publish at a fixed resolution and rate, we needed to implement our own scripts, in which we took inspiration from several different open-source Unity Robotics projects. We also made several improvements to make the rendering and image data transfer more efficient. There are other projects that also feature for example a 3D lidar or depth camera [60] so in future work, these could be assimilated into HARDNAV too. For the purposes of benchmarking robust visual navigation however, IMUs and cameras are sufficient and biologically-mirrored sensors.

- **Different coordinate systems and Transform publishing** – One big inconvenience is that Unity uses a left-handed coordinate system where the 'y' axis points up, whereas in robotics and ROS a right-handed system with 'z' pointing up is used. The Robotics package does contain functions for converting between these systems but has no documentation, and so to implement anything custom, we needed to guess the functionality from the code of the package. The Unity packages also do not offer any way of automatically publishing transforms as in Gazebo. Unity has a project with a publisher of transforms between a robot imported through the URDF importer, but not any two general GameObjects. Thus, for calibration, evaluation and debugging, we wrote scripts for publishing a transform between any two objects and use it to output ground truth pose and poses between robot sensors into ROS.

With all these difficulties handled, Unity works very well as a simulator. We hope that this in-depth description of the simulator's workings and the challenges identified in working with Unity can help other researchers in expanding HARDNAV for their purposes or building their own simulation environments.

3.2.2 Supported Robots, Sensors, Actuators

As of now, HARDNAV features two robots - a super-simplified underwater/space box robot with disabled gravity and a Clearpath Husky robot, both shown in Figure 3.2. For simplicity, and to focus on large-scale, long-term navigation, we created scripts in Unity for "perfect" velocity and force control, which are used on both the flying robot and the Husky robot. These scripts have access to the robots' true velocities in Unity. We believe that controlling velocity is generally attainable in most robotic platforms, and thus it seems like a reasonable place where to draw the simplification line, so that researchers can more easily focus on perception and navigation. Unity does supports precise simulation of wheel dynamics (after all, it is a game engine), and it is possible to implement many forms of dynamics, so if anyone needs such features for their spatial navigation experiments, they can spend the time to implement them.

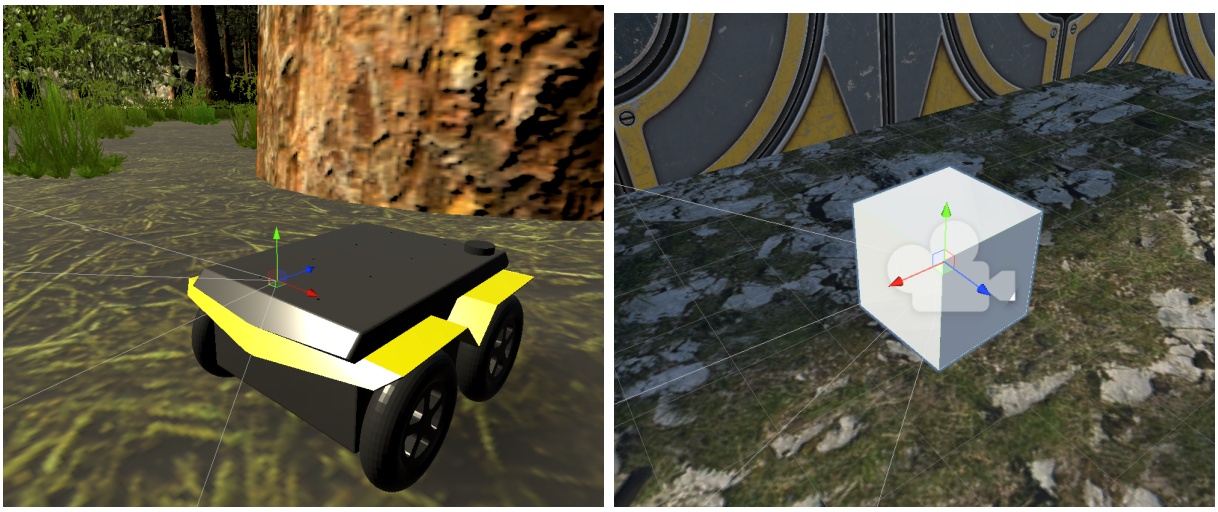


Figure 3.2: The two so-far implemented robots in HARDNAV — Left: Jackal robot with a camera with the field of view visualized and IMU. Right: idealized gravity-disabled flying robot with the same sensory setup. Both have a simple built-in velocity controller.

Both these robot types feature an IMU and a monocular pinhole camera, and their parameters and publishing rates can be changed in the Unity Editor. Importantly, there is no limit to the number of robots in the simulator. One can simply drag-and-drop the blueprint of the robot from the project's filesystem anywhere in the world, rewrite the topic names and thus create an instance of one of the robots. In future work, we intend to implement spawning and despawning robots and automatically through a ROS subscriber or service server. One other student is also working on implementing UAVs of the Multi-Robot Systems Group [62] into Unity and this will be merged into HARDNAV in the near future.

3.2.3 World Configuration Resetting

In the simulator, we built a software pipeline that allows the user to change pre-defined world states (such as enabling/disabling rain, dynamic objects, clouds, changing ambient light intensity and angle, changing spawn pose of robot and robot type) through a ROS service. The configuration is specified as a YAML file, and we provide an script that loads a default world state from the file and calls the service to reset the world with that config. The contents of the service message are simply the entire YAML file as a string, which is parsed inside Unity.

3.2.4 Worlds, Objects, Textures and Extreme Conditions

The core idea behind the simulator is that by taking navigation *ad absurdum*, we could find solutions that cover all corner cases where SLAM or exploration or navigation

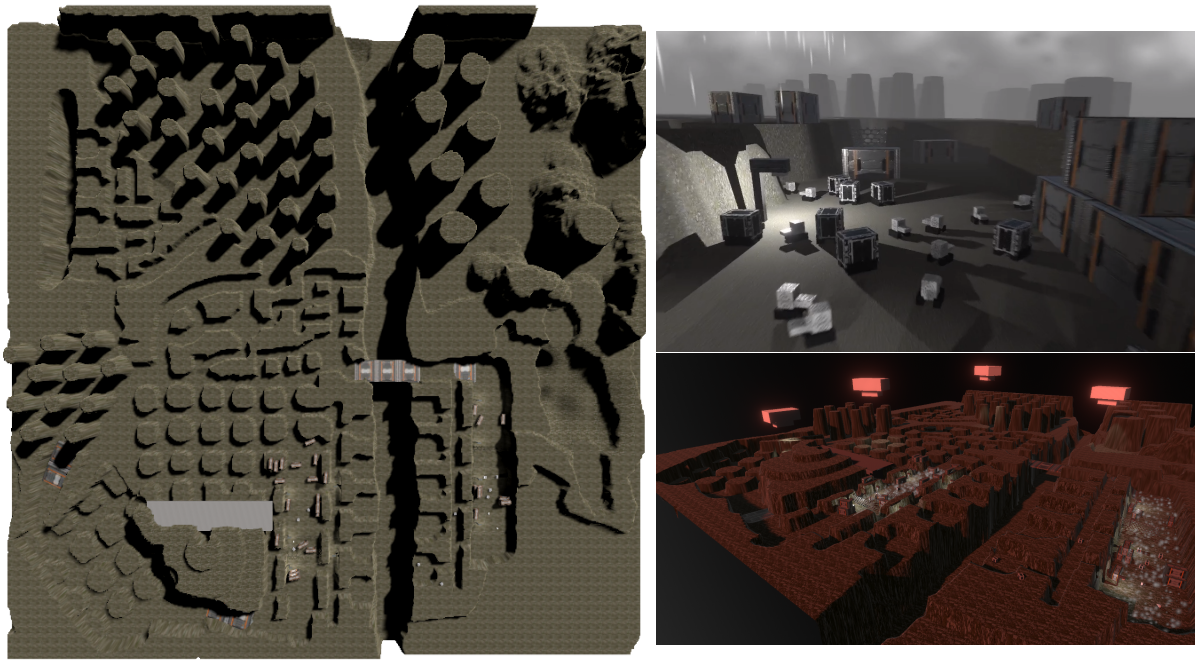


Figure 3.3: **Left** - The *Station* world custom-built for HARDNAV. This world was purposefully designed to have *many* areas that are nearly identical, such as the grid-like protuberances in the central left part (inspired by silos or water treatment plants in the real world). Not only does the environment look very similar at each crossroads in area environment, but this same environmental structure is also present in the bottom left portion of the map. **Top-Right** - In this image, you can see the "workspace" area with dynamically moving robots that avoid the user-controlled robot, many crates (whose position can be randomized through the world config service) and flickering light sources. This area is also mirrored on the other side of the valley. **Bottom-Right** - Another view of the world, with the world config tuned so that the lighting hue is red. Both of the two "workspace" areas are visible, alonng with the dust particles that are present in those areas. Additionally, there are the large white flying dynamic objects above the surface - these are meant to be moving "airship-like" objects that shine strong spotlights onto the map, which will pose difficulties to many static-world-assuming visual spatial intelligence methods.

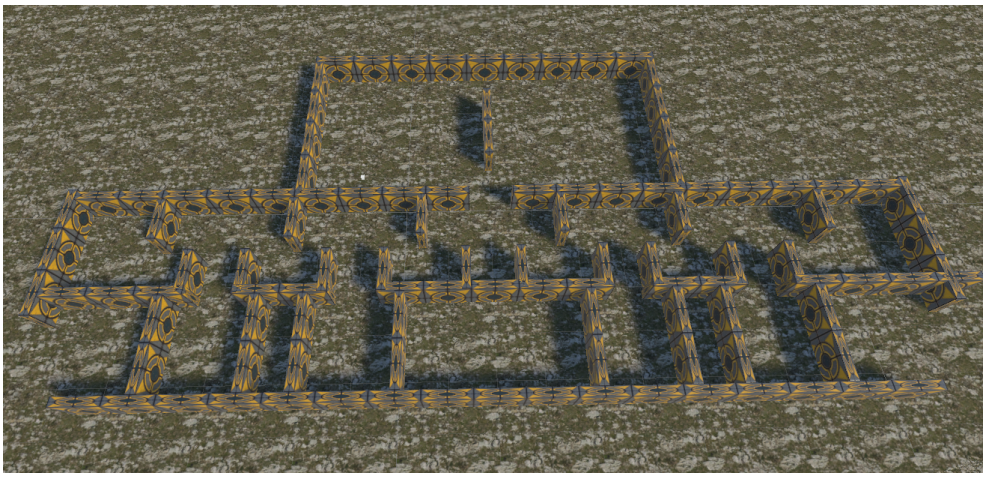


Figure 3.4: A top-down view of the "Symmetric1" world in HARDNAV used for the submap matching and exploration experiments in section 4.4 and section 4.6. The world has many levels of symmetry. First, it is symmetric along the plane cutting it in the middle of the image. Second, the 2 small rooms on the right and 2 on the left are completely identical. We set the tiling of the ground texture so that pairwise, the small rooms have even the same ground texture. It is designed primarily to test the limits of appearance-based and small-scale (e.g. from a single LiDAR scan) geometry-based place recognition and to show how our method is able to achieve place recognition using large-scale spatial geometry.

break down today. For this reason, we have implemented a wide array of objects, particle effects, environmental behaviors and worlds that are designed specifically to break today's navigation systems. A full list of these features can be found in Table 3.1. One of the worlds, desgined to be large-scale and highly ambiguous, and with dynamic objects in the air and on the ground, is described in Figure 3.3.

Many even more exciting situations could be made possible thanks to using a game engine – moving platforms such as ships or trains, completely abstract worlds made of objects and visual effects that are not encountered in the real world, and perhaps even non-euclidean worlds that humans can still navigate, as in the videogame "Antichamber", which could lead us towards generalized navigation principles, as discussed in subsection 2.4.3. This is set as future work.

3.3 Proposed Benchmarking of Robust Navigation

In our proposed method of evaluating navigation as a whole, we build on the work of [63]. In [63], the authors define several possible task specification for evaluating navigation capabilities of embodied agents. Our task can be categorized as 'area navigation', as the goal is always to reach a given area of arbitrary shape.

Challenge	Examples	Expected negative effect on SLAM, VPR and VBN
Visual corruption	Dust particles, *leaves*, rain, motion blur, *lens dirt*, big exposure changes	all of vision
Dynamic objects	Ground and air robots randomly moving around environment; *grass and trees swaying in the wind*, clouds	odometry drifting if many objects nearby, faulty object-based place recognition
Dynamic lights	Flickering lights, *fires*, many light sources on dynamic objects	purely visual odometry drifting due to many outliers or perceived motion
Illumination and visibility changes	Global directional light of varying intensity, color, angle; fog of varying color and density	faulty visual place recognition, odometry having low amount of features
Small structural change across sessions	Object positions being randomized, *trees falling down between sessions*	wrong place recognition, relocalization
Drastic structural change across sessions	Passages being blocked, *buildings being built*, distant landmarks disappearing, *several meters of snow*	wrong place recognition, relocalization, teach-and-repeat navigation
Robot affecting scene	*Leaving footprints/tracks*, moving objects on collision	place recognition, relocalization
Featureless areas, transparent objects	textureless corridors (lack of visual features), straight smooth corridor/wide open area (depth features), fences, window	odometry drift, depth estimation failures in case of vision
Perceptual aliasing / self-similarity	Multiple areas having the same appearance - appearance-based, topology-based or both. Medium in *Forest1*, heavy in *ScifiBase1*	relocalization, loop closure
Scale variation	Having both small areas (buildings, small corridors) and comparably larger areas (forest, big rooms) in one environment	fixed-resolution mapping/navigation
Large environment scale	The environments are approx. 2km wide with robots being approx. 1m wide	SLAM memory problems, difficulty learning with deep learning methods

Table 3.1: The challenges for place recognition and navigation implemented (and planned, in italics) in HARDNAV. The upper portion of the challenges in the table can all be individually enabled or disabled in the session config YAML string for each session. The lower half challenges are caused by the design of the individual environments and are thus not toggleable.

We name our task definition as **multi-session area navigation** (MSAN). Because navigation requires an agent interacting with an environment, a navigation task can be tested only in the real world or in simulation [61], and thus the benchmark is tied to the simulator or real-world environment used. A specific MSAN benchmark is defined by a robot R, an environment E, a set of datasets D taken in E, and a set of trials T performed in E. Because we intend to design very difficult benchmarks, we select the evaluated metric as simply the percentage of successful trials.

Each dataset D is a sequence of sensory inputs at variable rates and a mapping $A(t)$ of each timestep to a set of area labels for all pre-defined areas that the robot is intersecting with at time t. Pragmatically, we realize $A(t)$ by using Unity's built-in collider system — we define areas as oriented bounding boxes in the Unity Editor (although more complicated shapes are also possible) with unique labels, and in HARDNAV, we implement a script that finds all intersecting areas at each timestep and publishes a ROS message containing the corresponding labels. The labels are not meant to carry any semantic meaning, which

differentiates our task from vision-language navigation [64]. In the real world, the label setting can be implemented for example as a button that defines a new area in a given radius around the robot's current position when a person is driving the robot around a work area.

Each trial is given by a world configuration W_T (its implementation is discussed in subsection 3.2.3), a target area label A_T and a duration τ_T. In each trial, the simulated world is reset to a state specified by W_T, which also contains the robot type (should be same across all trials) and robot pose. Then, the trial is marked as successful if the robot arrives within the area A_T and sends a "area reached" signal within time τ_T. In other words, the trial does not end when the robot reaches A_T, but only when the robot sends the signal or if the time runs out. If the robot does not send the "area reached" signal in time or if it sends it but is not within A_T, the trial is unsuccessful.

Our reasoning for having the "area reached" signal is inspired by the authors' arguments in [63]. For a wide array of tasks that we might want robots to perform and which require them to be in a specific area (e.g. sowing seeds at a specific part of a field, searching for objects in a given room, patrolling a given area), it can be too expensive or even impossible to implement any infrastructure, such as RFID tags, that would inform a robot that it has reached the target area This adds an interesting layer of difficulty, as the robot needs to weigh searching the environment for more information against navigating to the most likely location in the current session, which is a hard and still mostly unsolved problem in robotics, as we discuss in subsection 2.2.2.

This task definition is quite general, and can encompass both small-scale, static-environment navigation with no appearance changes between sessions to kilometer-scale navigation trials with dynamic objects and heavy appearance change. Most of the software for evaluating the benchmarks (the world configuration system, area definition and automatic checking) is already in place, but the design of specific benchmark instances is out of scope of this thesis and we leave it for future work.

3.4 Comparison with other Simulators

There are many robotics simulators available for development of spatial navigation, so why construct a new one? Most of the popular simulators already allow changing environmental conditions [59], [65].

The most traditional and commonly used, general purpose simulator for robotics is the Gazebo simulator. Its main advantage is that it does not rely on any external physics or rendering engine and is *completely* open-source. Additionally, the simulator runs as a ROS node, and thus sensor data can be obtained from it directly as ROS messages. Compared to this, simulators based on the Unity engine [59, 60] need to compress and send the sensor data out of the simulator through a net socket, and then convert it to a ROS message which slows down simulation time. However, Unity has been shown by the authors in [66] to have

plentiful advantages over Gazebo, mainly the ability to render large-scale scenes. It is, therefore, better suited for the purposes of benchmarking robust autonomy in large-scale, changing and realistic environments.

Then there is the FlightMare Simulator [60] for UAVs, also based on the Unity engine. This simulator is unique in the fact that it has a separate high-fidelity physics simulator for UAV dynamics. This makes it very useful for reinforcement learning of UAV behavior that can then be better transferred to the real world than if the Unity's base physics engine were used. However, it makes it specialized for UAVs, and thus not usable for simulating navigation of varied robots in varied environments.

The most similar simulator to HARDNAV is the AirSim [65] simulator. AirSim offers simulation of UAVs in outdoor environments with weather effects, and is extremely popular among robotics researchers. It has been used to demonstrate AirSim is using the Unreal game engine, which is the competing game engine to Unity. Same as Unity, Unreal features an asset store, so developers can easily take premade assets (e.g. a hyperrealistic, hand-crafted forest environment and its accompanying 3D models and textures). Its benefit over Unity is definitely the rendering – Unreal has much more easily accessible photorealistic rendering capabilities. Its downside compared to Unity is that Unreal is more of a *game* engine than Unity, meaning that a large part of the UI in the editor and many structures are organized specifically for game development (for example there must be a "Player" object in the scene). Unity is a game engine, but is more of a "sandbox" than Unreal. For this reason, we chose Unity as the base engine for HARDNAV. One other major issue of AirSim is that in 2017, its development was taken over by Microsoft and is no longer open-source. HARDNAV, compared to AirSim, is fully open-source, offers multiple robot types (and simple tutorials that show how to import new robots), and is built on top of the Unity engine which allows easier setup and modifications of worlds and interactive objects and task-specific logic.

Another popular robotics simulator is CARLA [59], built on top of the Unreal engine. This simulator offers large-scale environments, changing weather, but is only for cars.

In the machine learning community, there are also many simulators available for embodied AI aresearch - for example Habitat [67] and RoboTHOR [61]. These are mostly aimed on the task of "vision-language navigation" in which deep learning models are trained to perform tasks specified in human language [64]. The Habitat and RoboTHOR simulators support changing the textures of objects and environmental surfaces, and they have been used for research of vision-based navigation, for example navigation with damaged wheels or cracked camera lens [68]. These simulators however only offer indoor environments and not much in the terms of dynamic objects or lighting.

None of these simulators can be directly used or easily modified for benchmarking robust navigation on varied robotic platforms in varied large-scale, realistic, dynamic environments in the way described in section 3.3. HARDNAV is thus different and it offers ways of developing and testing robust general vision-based navigation. Furthermore, its modular nature allows other simulation developers to easily take the scripts or prefabs for

things such as camera data publishing, organizing random ground and air objects, flickering lights, closing gates etc., and copy them to their Unity robotics projects. Thus, it can also help the robotics community by simply being another open-source project of Unity-based robotics simulation, as not many such projects (more so well documented) are available.

3.5 Discussion and Future Work

In the future, we envision having many of these benchmarks available within the simulator, so that anyone can download a dataset, a compiled build of the simulator, and start solving these challenges using their vision-based autonomy systems. This could serve as a new way to evaluate spatial intelligence systems (place recognition, SLAM, ...) *in the context of navigation*, which passive dataset-based evaluation cannot do.

Chapter 4

Multi-Map Visual-Inertial Navigation and Exploration System

Contents

4.1 System Overview and Design Choices

Navigating and mapping any environment with only a single camera and inertial measurement unit (IMU) is a challenging task for many reasons. First, visual and visual-inertial SLAM will inevitably accumulate large amounts of drift with respect to the starting pose. Usually, this is corrected through place recognition methods that deform the pose graph of the traveled trajectory. However, taking inspiration from the findings that animals use many fragmented spatial maps, we attempt to move away from the notion of storing and optimizing the poses of all spatial objects in a single coordinate frame.

For this reason, we decide that **in our system, there is no one fixed frame used in any way, the whole map is composed of submaps, some of which might**

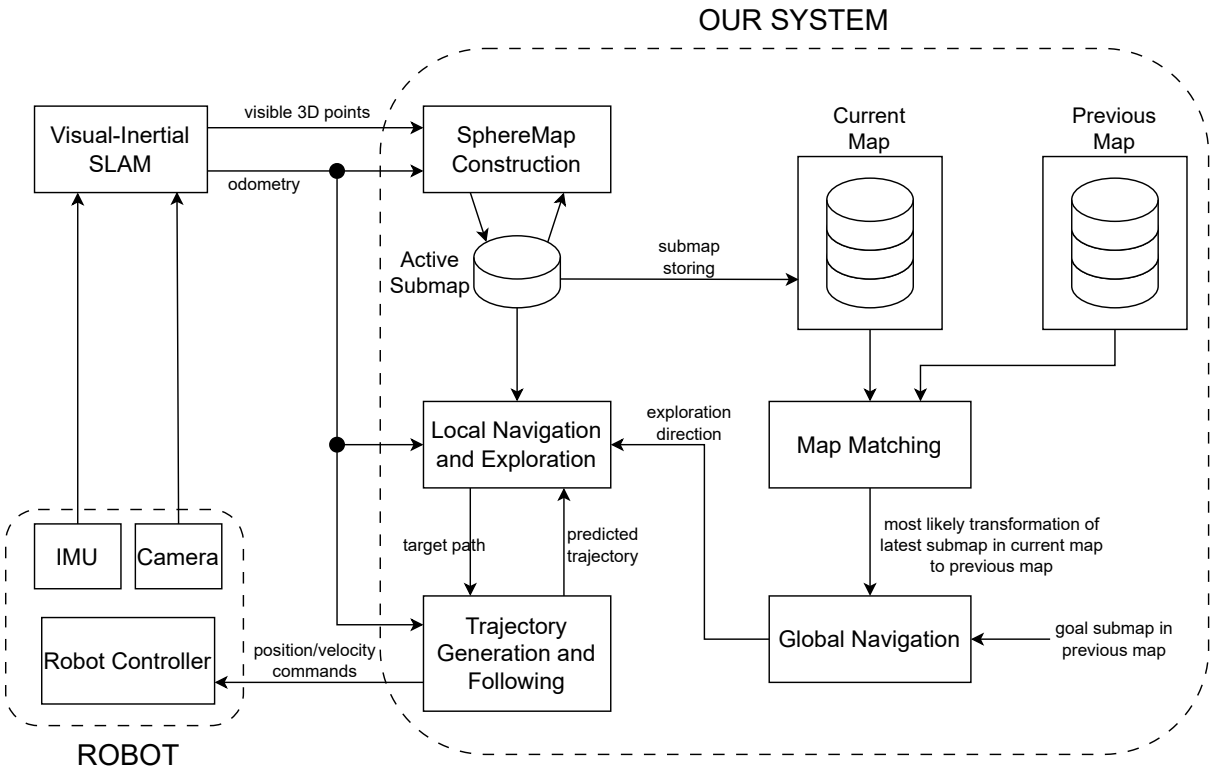


Figure 4.1: Diagram of the designed system

have extremely noisy data, and all computations are using relative poses. This has made the implementation problematic, as this opens many problems concerning data association (such as "How to decide that one submap is the same as another?" or "When to merge submaps?"), but we believe this difficulty must be embraced to enable systems that do not break down due to erroneous loop closures, which is always a possibility, and can navigate long-term in many sets of environments. This direction could hypothetically also lead to more generalizable navigation principles, which could be applied to abstract space mapping and navigation, as we discuss in subsection 2.4.3.

Furthermore, a large variety of motions (fast motions, pure rotations, etc.) can cause visual and visual-inertial odometry to lose tracking of visual features (and on aerial systems, will force the robot to physically crash). To tackle this, we designed the planning modules and methods so that **local motion planning is odometry-aware, safety-aware, sampling-based, replanned at a high rate and only uses a few of the most recent submaps in which the odometry drift is negligible**.

The system's modules are illustrated in Figure 4.1 and the modules are described in detail in the following sections. Implementation-wise, the system is written in python, uses ROS, and runs in real time on a standard laptop computer. The "Visual-Inertial SLAM" and "Trajectory Generation and Following" modules vary in implementation between our

simulation and real-world variants of the system due to the properties of the used Tello UAV platform, and are described in more detail in the following sections.

4.2 SphereMap Builder

The input of this module is odometry, a point-cloud specifying points and their positions relative to the robot, and inverse depth covariance for each point. We assume the points are estimated from a camera in the current implementation, but as the reader will see, the map update function can easily be changed to use different field-of-vision models than just for a camera.

The local map is composed of obstacle points, free space spheres and frontier points, as visualized for example in Figure 4.7. This is a major extension of our previous work [26] which relied on a preexisting occupancy grid built from LIDAR data. The main advantage of this representation is that it can easily be deformed and scaled in different axes. Furthermore, in search-based planning methods such as A*, planning speeds can be much faster than using an occupancy grid (3 orders of magnitude in our previous work [26]), because the spheres can describe free space in fewer nodes than grids, and they also explicitly encode obstacle distances at each sphere's center, which is often important for planning.

The local map is updated at 10 Hz, and one update can be described thus: As in FLAME [69], we first perform Delaunay triangulation on the input points, projected to the camera's image plane. In FLAME, the authors tear apart the triangles where the points are too far from one another, but for simplicity, we do not do that. We then construct an obstacle mesh M_o from the connected 3D points, and an FOV mesh M_{fov} from a point in the camera's position and points that form a convex hull of the points in projected in the image plane. Together, they form a visible freespace mesh M_{vis}, which is watertight.

A fixed amount of points inside M_{vis} are sampled during each update, and we check their distance to M_{vis}. If the potential free-space radius of a new sphere at a given point is larger than some minimum radius r_min, we add it.

Updating existing spheres is trickier. Some previously added sphere might be seen in the current frame only partially and thus be much closer to the current M_{fov}, and it would not make sense to decrease its radius r because of that. Thus, for existing spheres intersecting or inside M_{vis}, we compute the distances d_{fov} from M_{fov} and d_o from M_o. If $\max(d_o, d_{fov}) < r$, we increase the radius r to $\max(d_o, d_{fov}) < r$. If $d_o < r$, we decrease the radius r to d_o, and delete it if $d_o < r_{min}$. To keep the number of the spheres not exploding to infinity, we perform a check for redundancy whenever we update the radius of an existing sphere or add a new sphere. The redundancy check is identical to the one in our previous work [26].

Adding obstacle points to the map is simple – if some input 3D point is less than r_{detail} away from any existing points in the map, it is added. Updating them robustly is again slightly harder, because in our system, we assume that the depth estimates can be

very inaccurate. For the purpose of this thesis, we simply delete points that fall at least some pre-defined distance into free space defined by the spheres.

We also update frontiers [70] (this can be turned off and is used only for planning) as the boundary of free and unknown space – frontier points are simply sampled at the edges of M_{fov}, added if not too close to other frontier or obstacle points, their normals with respect to freespace are computed using distances to nearby spheres, and if they are found to be inside existing spheres, they are deleted.

Taking inspiration from how fragmentation seems to play a very important role in mammalian navigation and spatial cognition, as described in subsection 2.4.2, we do not build one large map, but rather fragment the world into many submaps. As of now, the fragmentation occurs whenever the UAV has traveled a pre-defined "context" distance, which is a weighted sum of traveled euclidean distance and integrated heading changes (rotation around the gravity axes), which is tuned according to the severity of drift in the used odometry.

When a map fragmentation occurs, a new map is initialized and the previous one is stored into a long-term storage. Currently, this could pose some difficulties for the planning module, but in future work, we intend to use close stored submaps, if they align with the current submap and are not too old. This becomes largely more challenging if we consider dynamic environments, where the robot should likely always look with its cameras in the direction it is moving, but this is completely out of scope of this thesis.

4.3 Local Perception-Aware Navigation and Exploration

For the action pipeline, we choose the standard approach of planning a path and then sending it to some module that converts the path to a feasible trajectory and follows that trajectory by giving lower-level control inputs to the robot. We chose path planning to be sampling-based and we conceptually build on RRT* [71], first because of how easy it is to modify the sampling strategy and rules for pruning or adding new nodes, and second, because the quality of path founds corresponds to the time spent on pathfinding, which is similar to path planning in biology. The action pipeline of course depends on the robot used, and for the purposes of this thesis, we chose to work with a real-world Tello UAV connected to the MRS system [62], and in simulation with the idealized velocity-controlled flying robot described in subsection 3.2.2.

On both platforms, the pipeline can be described in short as follows: Whenever the UAV has no target goal, we allow the planner to search for paths via RRT through free space and away from obstacle points, with some user-defined minimal distance to the edge of free space. We check the leaves of the RRT tree and allow selecting only the ones, in which at least one frontier point would fall into the FOV of the robot. To not lose odometry, we also only allow adding an RRT node if at least 3 obstacle points (which correspond to

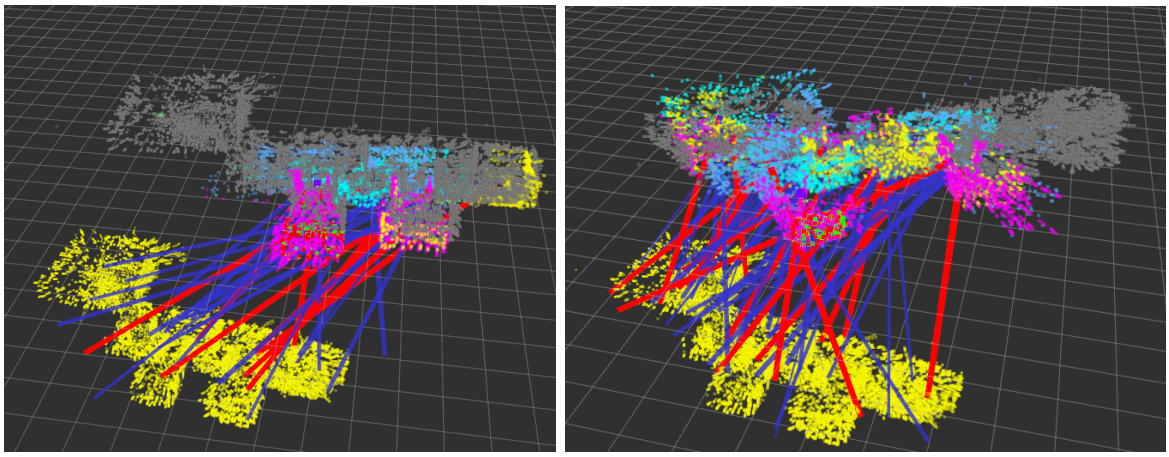


Figure 4.2: A visualization of the relative map transformation estimation described in section 4.4. Red lines represent the most likely match between a given submap in the current map (multi-colored submaps) to submaps in the previous map (yellow). One can see that many of the red matches are wrong, but since we take the top 3 most similar submaps as possible correspondences, the RANSAC algorithm is capable of converging to a correct solution, visualized as the grey map transformed by the most likely transformation between the maps. In both images, the algorithm is able to compute an approximately correct transformation (surely useful for navigation) with 3 (left) and 5 (right) inlier correspondencies, with inlier distance set to 15 meters. Grid cell size is 10 meters and the grid is at the same height as the current map.

triangulated visual keypoints) are visible from that node. Additionally, we limit the rotation per meter of translation, so that the robot does not perform purely rotating motions, which can break visual SLAM. We then rank the RRT leaf nodes based on the cost of the path found to them, and weigh path length and safety in the same way as in our previous work [26]. We also allow directed exploration by adding the distance of the leaf position from the robot's current position in the exploration direction to the cost function.

When a goal is selected, the UAV then sends the path to it to a module that converts the path into a trajectory and track that trajectory (on Tello, we use the trajectory generator of the MRS system and in HARDNAV, we implement a simple path follower that sends velocity commands to the robot's controller in Unity). Because the distance measurements can be very noisy, we also check the predicted trajectory of the UAV, for whether it would get too close to obstacles. If that happens, we trigger a replanning procedure.

4.4 Large-Scale Geometry-Based Map Matching

We designed this module as a proof-of-concept that even in vision-based systems which have sever odometry drift and depth measurement uncertainty, it is possible to

extract rough environmental 3D geometry and match it between parts of the two maps for place recognition. We intended to use a particle filter algorithm to obtain the most likely place on the previous map from these outputs, then to plan a path from there to a target submap in the old map, and to follow that path by setting the exploration direction of the local navigation and exploration module described in section 4.3. Unfortunately, due to time constraints, we did not manage this in a reliable manner, so we only showcase this as a proof of concept of using large-scale geometry as an alternative, or even better – as an addition to apperance-based place recognition.

Given a current map A and a previous map B, the map matching works as follows: at a fixed rate, we select one random submap $X \in A$ and a random submap $Y \in B$ We take up to N (tested with $N = 4$) adjacent submaps (because we do not use any submap-submap association yet, these are the submaps that were explored before and after the given submap, but when we do place association, we will use the associated maps as well). The obstacle points from the adjacent submaps to X are all assimilated into one pointcloud and the same is done for Y and its adjacent submaps. We then perform point-to-plane ICP matching, initialized with random relative rotation, and slight random offset. We store the result of the ICP matching, and set its inlier root mean square error (RMSE) as the similarity score $S(X, Y)$ and the resulting transformation.

This is done around 3 times per second and not computed in larger batches, because the system needs to have other threads running to plan and build its current new submap. Ideally, we should have some way of describing each submap by a vector descriptor and then take distances of the descriptors in the feature space as $S(X, Y)$, but we have not encountered an applicable descriptor computation.

To compute an estimated transformation between two sets of submaps in A and B, we essentially do the same thing as in classical homography computation between two camera images. We establish correspondence of each submap X to another submap Y' in B at random, but with choice probability proportional to the score $S(X, Y')$ with some score cutoff (currently we take only the 3 most similar submaps, but this can cause problems in larger scenes). Then, we perform around 1000 iterations of selecting one $X- > Y'$ correspondence, retrieving their transformation T, which is the candidate transformation, project all submaps in A to B using T and with some pre-defined distance, determine which correspondences are inliers or outliers. We then rank the hypotheses first by numer of inliers, and when tied, take the one with the smallest summed inlier reprojection error. An illustration of this matching can be seen in Figure 4.2.

4.5 Learning-Free Visual Inverse-Depth Estimator

Estimating distances using a monocular camera is a difficult task with extreme potential for real-world applications, thanks to the low cost of cameras compared to RGB-D or LiDAR sensors. With a single moving camera and without prior knowledge on object

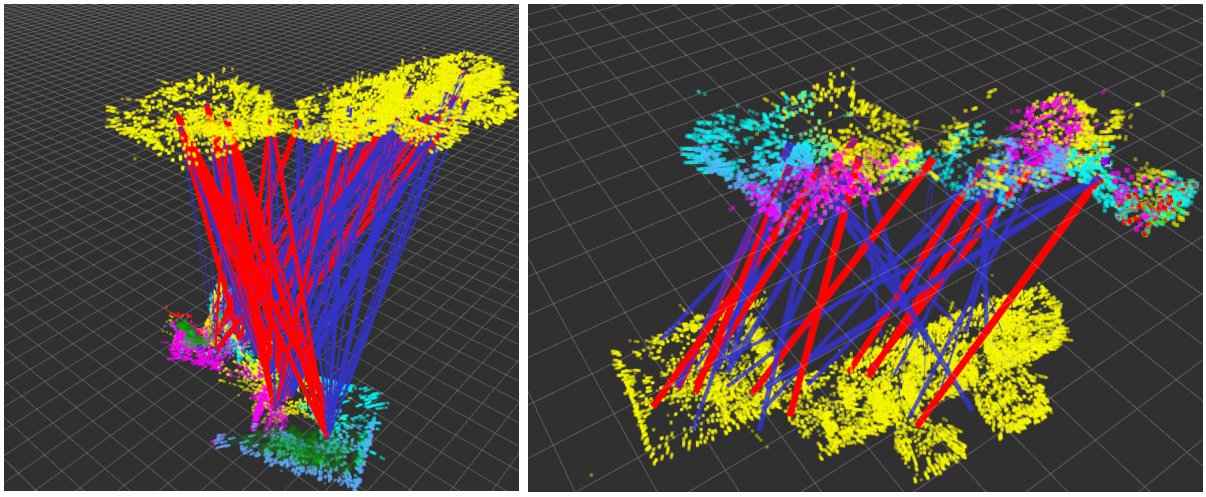


Figure 4.3: A visualization of the matching described in section 4.4. The previous map is yellow (up in left image, down in right image) and the multi-colored map is the current map, with colors representing submaps. In the left image, the red lines signify the 3 most likely matches, and blue lines are the other matches for each submap, with line width signifying the match score. In that image, the matching is shown after stopping motion for about 1 minute and letting the randomized ICP matching selection gather enough matches between all possible pairs, and one can see that the correspondencies are corrrect. In the right image, the first most likely match is visualized in red. The small room on the right-hand side of the image is not yet matched to the second small room in the original map, as this image is taken shortly after exploring the small room and the matches have not yet converged.

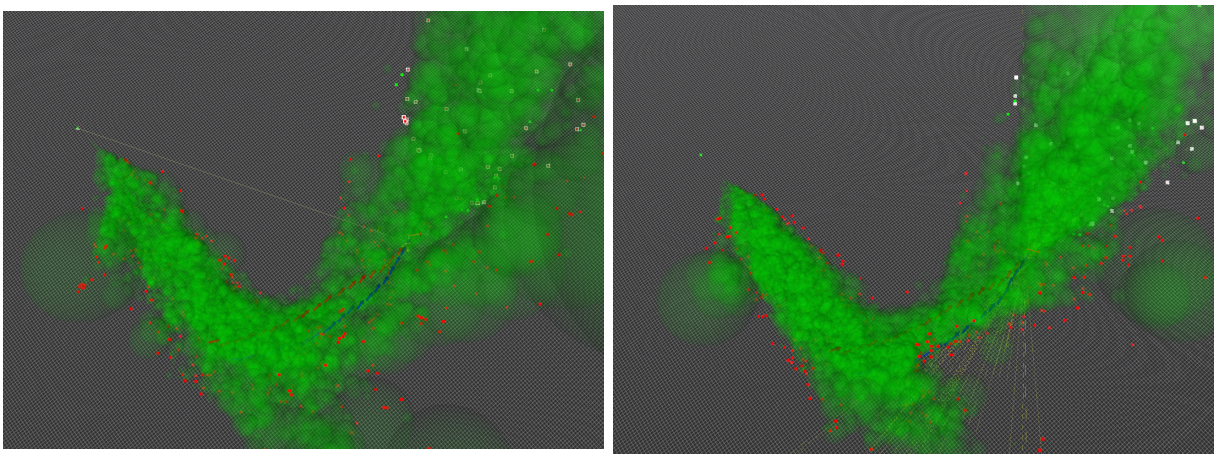


Figure 4.4: The effect of using Bayesian filtering of feature inverse depth - disabled on the left, enabled on the right. The map is a corridor on the left, with is completely smooth with textured walls, which opens up to a large open area on the right side of each image. Green: spheres of the single submap in this example, Red: map points Red arrows: OpenVINS odometry used as the base for the triangulation, Blue arrows: purely monocular SLAM of the FIRE module (see section 4.5), scaled to the OpenVINS odometry scale.

sizes, it is only possible to recover distances of features in space up to scale. Imagine seeing a video from a UAV flying through a city – it is impossible to estimate from the camera data alone, without any prior world information, whether the video is of a normal UAV flying through a normal-size city, or from a tiny UAV flying through a small tabletop model of a city.

Deep learning approaches have made progress in this regard, and some models are able to estimate scaled depth by learning the objects and spatial relations in their datasets, for example in [72]. However, theoretically, if a camera frame contains very unfamiliar objects and environments, that approach will fail to give a reliable depth estimate. Models for estimating unscaled depth also exist, and it would be interesting as future work to try using them in combination with metric scale sensors (such as an IMU). However, for the purpose of this thesis, we chose a different approach.

Because GPUs have high power requirements, are more expensive and can add weight to robots, it is of particular utility for roboticists to use methods that do not require GPUs to run in real time. For depth estimation, the authors of FLAME [69] presented a method of estimating scaled depth from images and known camera positions. Their method track visual keypoints in the camera data, which is similar to any other indirect visual SLAM [18], but taking just the tracked points would result in a very sparse depth map, which would not cover a large part of the visual field. In FLAME, the authors perform Delaunay triangulation on the tracked features in 2D, and thus interpolate the depth between any 3 connected points, which gives a more rich depth estimate.

We have tried using OpenVINS [73] in combination with FLAME, but encountered

an issue with FLAME, most likely cause is that the inverse depth updates of visual features happens at every frame, even if the robot is stationary. Thus, if the robot is stationary for a longer time, the inverse depth coming from FLAME gets corrupted after a few seconds. Another option we considered was that OpenVINS, which estimates robot trajectory and positions of SLAM points in space (with no loop closures), outputs its SLAM points in the form of a pointcloud, so they could be used in a similar way as in FLAME. Unfortunately, OpenVINS uses SLAM points only at very close ranges to the robot, and thus the range sensing of this method is limited.

For both of these problems, we could engineer a workaround, but to learn about depth estimation and visual SLAM, the author of this thesis has decided to implement a custom pipeline similar to FLAME for inverse depth estimation (for benefits of using inverse depth opposed to depth, see [74]). The pipeline is functionally very similar to FLAME — it also requires odometry, but allows it to be noisy, which is the case for the Tello UAV platform in the real-world experiments. As in FLAME, FIRE tracks visual keypoints, performs some triangulation between keyframes, and fuses inverse depth measurements in a Bayesian manner in the same way as the authors of FLAME. However, FLAME makes the assumption that the odometry is very precise. In our system, we perform a purely visual SLAM, by estimating the unscaled transformation between the current and previous keyframe, and then scaling it so that the distance traveled in the purely monocular FIRE SLAM matches the metric distance traveled according to the odometry, up to some time in the past (approx. 5 seconds in the experiments.) Scaling the visual SLAM over a large window of time allows us to deal with image-odometry delays and small-scale odometry errors, as we demonstrate in the real-world experiments. The tracked 2D points are projected to 3D based on their inverse depth estimate and given to the SphereMap builder module in real-world experiments instead of the OpenVINS 3D points used in simulation.

4.6 Vision-based Exploration in HARDNAV

The demonstration of mapping and exploration capabilities of our system in simulation is rather simple compared to the real world experiment, and is depicted in Figure 4.6. In this experiment, compared to the real world, we used OpenVINS as the main odometry source, and the map is built using the triangulated points provided by OpenVINS, which it uses internally for SLAM. In the terms of *directed* exploration, the system did what we wanted it to do. Larger-scale area-specified exploration requires handling the drift induced in the submaps, which is severe even in simulation and can be seen in Figure 4.5, which is left for future work.

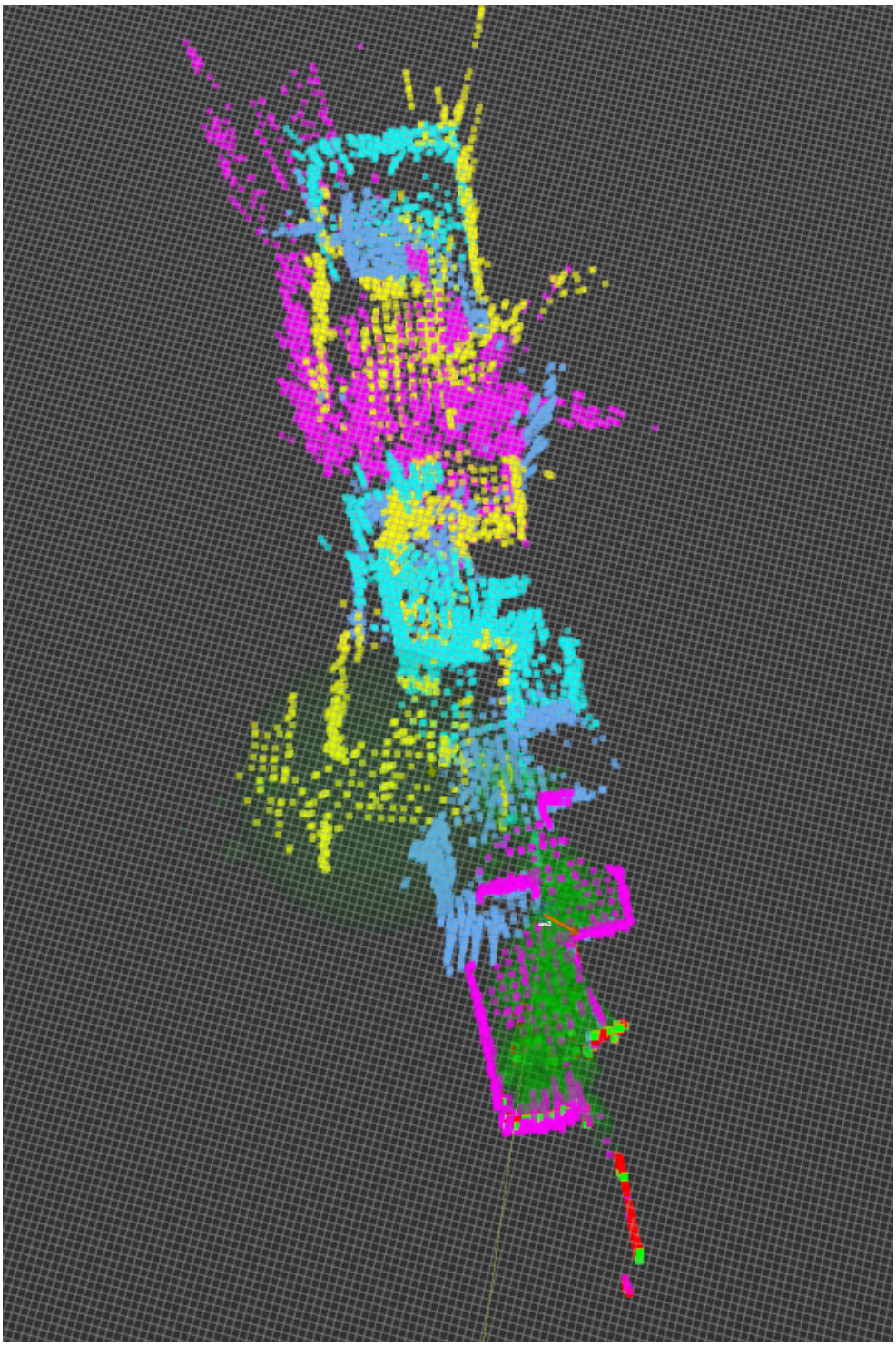


Figure 4.5: Visualization of drift encountered during a manual flight through the environment depicted in Figure 3.4 while running the mapping pipeline using OpenVINS. In this example, we flew the robot to the rightmost end of the map, rotated it, and moved back towards the middle of the map. One can see that the overall map is severely shifted relative to the world's layout.

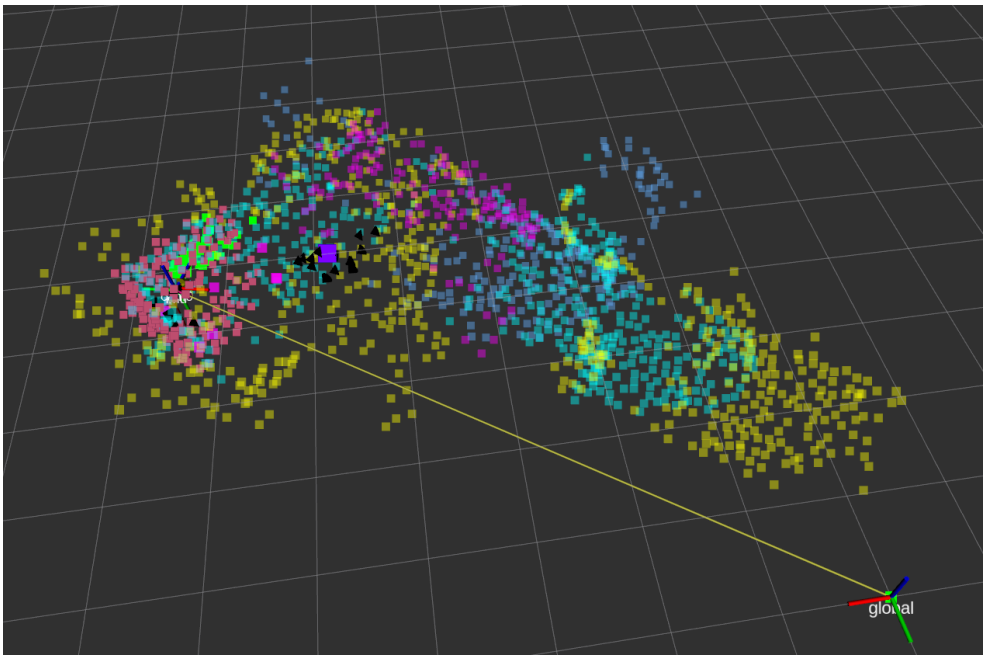


Figure 4.6: Visualization of the directed exploratint experiment on the flying robot in HARDNAV, in the world visualized in Figure 3.4. The UAV system was iteratively planning next best views that should uncover some frontiers and reached them. We set the exploration direction as left relative to the starting pose (in the direction of the red axis of the global frame visualization in the bottom-right corner). The UAV explored, did not crash into any walls, including the wall in the middle of the large room (center of the image), and we ended the experiment when the UAV started oscillating between exploration goals in the corner in the left part of the image, as that was the local maximum of that exploration direction.

4.7 Vision-based Exploration in the Real World

For the real-world demonstration, we used the DJI Telo UAV platform, pictured in Figure 4.7. The Tello UAV sends its camera data stream from its frontal camera and its internal velocity measurements (computed onboard the UAV using DJI's internal odometry software that supposedly uses the UAV's downward-facing camera and IMU) over WiFi to a connected computer. On the computer (in the experiments a standard laptop), we use an open-source MRS wrapper [1] that attaches the Tello API to the MRS UAV System [62] and integrates the velocity measurements to provide odometry ROS messages. With the image and odometry, we use our custom inverse depth estimator (see section section 4.5) for triangulating points of environmental surfaces, which we normally get from OpenVINS in simulation, as we weren't able to get OpenVINS running with Tello data).

We ran several experiments at the Charles' Square campus of CTU. In them, our

[1] https://github.com/ctu-mrs/mrs_uav_dji_tello_api

goal was for the Tello to explore as far as possible, and we set the preferred exploration direction to face forward from the origin of the latest submap, to simply "explore forward and avoid obstacles". Many of them resulted in a crash or emergency landing due to the problem we encountered that Tello requires relatively high amount of ambient lighting to work, and will often give incorrect estimates or land unexpectedly or even refuse to take off, and even at noon, there is not much light in the campus's courtyard. A good example of how bad the odometry is, can be seen in Figure 4.5.

We have achieved three experiments where the UAV flew more than approx. 10 meters by itself and the crash was a result of severe odometry failure. In the most notable experiment, visualized in figure Figure 4.7, the UAV explored for roughly 30 meters forward, dodged a large obstacle, and experienced a forced landing due to high control errors, likely due to the odometry being noisy due to low illumination. In another experiment at the same location, the system explored the obstacle from another side, but crashed in a tight spot, most likely due to a combination of the odometry being incorrect and the MRS controller not handling such errors For the experiments, we used the default control settings of the MRS Tello wrapper, where the control is very "floaty", with large oscillations from the target position, and this could also be tuned in future experiments. In the last notable experiment, the UAV correctly detected obstacle points on two very thin threes, and thus (arguably) correctly estimated obstacles and free space and only crashed, again, due to high control errors triggering a safety landing. Videos from the experiments showcase the system's workings best, and are available at [2].

The scale of these experiments might not seem that impressive at first, as robotic systems have achieved navigation of kilometer scales for example in the DARPA SubT Challenge [25, 24, 75]. However, the reader should note that in SubT, the long-distance navigating systems were equipped with LiDARs or global-shutter RGB or RGBD cameras, and were built by teams counting dozens of engineers. The system presented in this thesis achieves small-scale exploration and volumetric mapping **on a flying robot with only a single camera with variable image-to-odometry delays and very noisy visual-inertial odometry, with no prior map**, which, to the author's best knowledge, has not yet been explored in existing literature (see the 2023 survey [76] of vision-based navigation methods for UAVs). From the related works that are the closest to our system, in [77] the authors present a UAV system that performs monocular-inertial SLAM, but makes assumption that the world is composed of 3D corners, lines and corridors, and only demonstrate flights indoors, albeit longer ones. Our method makes no such spatial structure assumptions. In newer approaches, many camera-based exploration systems also utilize depth cameras [78], while our system has no depth camera sensors. Monocular (e.g. [79]) and visual-inertial SLAM (e.g. [73]) have been researched for a long time, but not many systems exist that can perform autonomous exploration, on a UAV, without depth sensors.

[2] https://mrs.felk.cvut.cz/musil2024thesis

4.8 Limitations and Future Work

The system has many avenues for improvement of all the components — the planning, collision avoidance, exploration logic, but most work remains to be done in connecting the multi-session submap-to-submap place recognition back to the planning modules, so that the system can navigate to a previously visited submap in a previous map. The local-map navigation and map-to-map matching is in a working state as of now, but we have to leave it for future work to connect them in a robust manner.

Other promising future work is to combine the geometry-based place recognition with appearance-based place recognition. In this thesis, we purposefully did not use visual appearance for matching at all, and instead used the shapes of the submaps, capturing larger-scale geometry, as a proof of concept that it can be done. Combining it with appearance-based methods could lead to place recognition systems, that would quickly generate potential matches from apperance, and filter them out after seeing a larger part of the environment, because matching geometry on its own requires a large portion of the scene to be seen, but leads to few outliers after that, which is useful in visually ambiguous environments.

Figure 4.7: The main Tello UAV exploration experiment, described in section 4.7. Purple points in camera picture: triangulated points at that given frame corresponding to white map points with yellow depth covariance. Red points in map: accumulated 3D points. Gold arrow in map: current exploration viewpoint. Purple points in map: frontier points.

Chapter 5

Conclusion

In this thesis we tackled 3 main tasks leading to a robust bio-inspired navigation system using only a monocular camera with inertial measurements All parts of the thesis assignment have been completed.

In chapter 2, we made an introductory-level summary of cognitive science research on animal navigation and map-making, compared it to SLAM and identified the open problems that today's SLAM-using vision-based robotic navigation systems struggle with, but animals handle with ease. We also identified potential ways of taking inspiration from natural intelligence, which have not been replicated in embodied artificial intelligence research, but seem worthy of doing so.

In chapter 3, we presented a new simulator dubbed HARDNAV, which we built as a tool to help robotics researchers build robust systems that can be transferred well to the real world with all of its often ignored difficulties. Additionally, we proposed a specific way of evaluating navigation as a whole using the simulator, and provided software tools for this type of evaluation. We have presented the simulator at a workshop of the IROS2023 conference, advertised it to researchers, and it is currently being used by one doctoral student, that we know of, for researching visual teach-and-repeat navigation. We plan to make a few more improvements, design several benchmarks of the proposed type, and then submit a publication on the new simulator and benchmarks, in the following months.

In chapter 4, we presented a system that can build volumetric maps and navigate in them using only a monocular camera and IMU on a flying robot, which is a mostly unexplored area in literature. As of now, the system is capable of autonomously exploring for approx. 100 meters on a simulated flying robot and approx. 30 meters in the real world on an inexpensive UAV platform without crashing into obstacles. Its current main point of failure is losing odometry tracking and this can be solved by improving the odometry-aware planning, which is very rough at this stage. We introduced a module for matching large-scale geometry of a set of submaps to submaps in another such map built in a previous navigation session, but due to time constraints, we have not yet connected the output of

this map matching to the local navigation module to navigate to areas seen in previous maps. Despite its challenges, we plan to keep the submap-based structure of the system, implement a prototype weak loop closing method similar to [22] and particle-filtering, where the pose graph is not deformed by loop closing, and to submit a conference paper on this system for IROS2024.

In conclusion, all of the thesis assignments have been completed and any of the three main contributions of this thesis — the artificial and biological navigation survey, the new challenging-scenario simulator and the submap-based visual-inertial navigation system can be easily expanded upon to be ready for publication in robotics journals.

Bibliography

[1] A. Tsoar, R. Nathan, Y. Bartan, A. L. Vyssotski, G. dell'Omo, and N. Ulanovsky, "Large-scale navigational map in a mammal," *Proceedings of the National Academy of Sciences*, vol. 108, pp. E718 – E724, 2011.

[2] M. Petrlik, P. Petráček, V. Krátký, T. Musil, Y. Stasinchuk, M. Vrba, T. Báča, D. Heřt, M. Pecka, T. Svoboda, and M. Saska, "Uavs beneath the surface: Cooperative autonomy for subterranean search and rescue in darpa subt," *Field Robotics*, vol. 3, no. 1, p. 1–68, Jan. 2023.

[3] M. Geva-Sagiv, L. Las, Y. Yovel, and N. Ulanovsky, "Spatial cognition in bats and rats: from sensory acquisition to multiscale maps and navigation," *Nature Reviews Neuroscience*, vol. 16, pp. 94–108, 2015.

[4] T. Báča, P. Štěpán, V. Spurný, D. Hert, R. Pěnička, M. Saska, J. Thomas, G. Loianno, and V. Kumar, "Autonomous landing on a moving vehicle with an unmanned aerial vehicle," *Journal of Field Robotics*, vol. 36, 01 2019.

[5] T. Krajník, J. Faigl, V. Vonasek, K. Košnar, M. Kulich, and L. Preucil, "Simple yet stable bearing-only navigation," *Journal of Field Robotics*, vol. 27, pp. 511 – 533, 09 2010.

[6] T. Krajník, P. De Cristóforis, K. Kusumam, P. Neubert, and T. Duckett, "Image features for visual teach-and-repeat navigation in changing environments," *Robotics and Autonomous Systems*, vol. 88, 11 2016.

[7] K. S. Chong and L. Kleeman, "Accurate odometry and error modelling for a mobile robot," *Proceedings of International Conference on Robotics and Automation*, vol. 4, pp. 2783–2788 vol.4, 1997.

[8] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, pp. 80–92, 2011.

[9] E. C. Tolman, "Cognitive maps in rats and men." *Psychological review*, vol. 55 4, pp. 189–208, 1948.

[10] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, pp. 1309–1332, 2016.

[11] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, pp. 403 – 429, 2006.

[12] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, pp. 52–57, 2002.

[13] M. Kaess, A. Ranganathan, and F. Dellaert, "isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, pp. 1365–1378, 2008.

[14] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, pp. 216 – 235, 2012.

[15] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[16] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2016.

[17] R. Arandjelović, P. Gronát, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5297–5307, 2015.

[18] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, and T. Sattler, "D2-net: A trainable cnn for joint description and detection of local features," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8084–8093, 2019.

[19] C. Campos, R. Elvira, J. J. G. Rodr'iguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, pp. 1874–1890, 2020.

[20] Y. Tian, Y. Chang, F. H. Arias, C. Nieto-Granda, J. P. How, and L. Carlone, "Kimera-multi: Robust, distributed, dense metric-semantic slam for multi-robot systems," *IEEE Transactions on Robotics*, vol. 38, pp. 2022–2038, 2021.

[21] J. Sivic and A. Zisserman, "Video google: a text retrieval approach to object matching in videos," *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 1470–1477 vol.2, 2003.

[22] F. Yu, J. Shang, Y. Hu, and M. Milford, "Neuroslam: a brain-inspired slam system for 3d environments," *Biological Cybernetics*, vol. 113, pp. 515 – 545, 2019.

[23] M. Milford, G. Wyeth, and D. Prasser, "Ratslam: a hippocampal model for simultaneous localization and mapping," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 1, 2004, pp. 403–408 Vol.1.

[24] M. Milford and G. F. Wyeth, "Mapping a suburb with a single camera using a biologically inspired slam system," *IEEE Transactions on Robotics*, vol. 24, pp. 1038–1053, 2008.

[25] M. Tranzatto, M. Dharmadhikari, L. Bernreiter, M. Camurri, S. Khattak, F. Mascarich, P. Pfreundschuh, D. Wisth, S. Zimmermann, M. Kulkarni, V. Reijgwart, B. Casseau, T. Homberger, P. D. Petris, L. Ott, W. Tubby, G. Waibel, H. Nguyen, C. Cadena, R. Buchanan, L. Wellhausen, N. Khedekar, O. Andersson, L. Zhang, T. Miki, T. Dang, M. Mattamala, M. Montenegro, K. Meyer, X. Wu, A. Briod, M. W. Mueller, M. F. Fallon, R. Y. Siegwart, M. Hutter, and K. Alexis, "Team cerberus wins the darpa subterranean challenge: Technical overview and lessons learned," *ArXiv*, vol. abs/2207.04914, 2022.

[26] T. Musil, M. Petrlík, and M. Saska, "Spheremap: Dynamic multi-layer graph structure for rapid safety-aware uav planning," *IEEE Robotics and Automation Letters*, vol. 7, pp. 11 007–11 014, 2022.

[27] N. Hughes, Y. Chang, and L. Carlone, "Hydra: A real-time spatial perception system for 3d scene graph construction and optimization," *Robotics: Science and Systems XVIII*, 2022.

[28] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. Denniston, S.-P. Deschênes, K. Harlow, S. Khattak, L. Nogueira, M. Palieri, P. Petráček, M. Petrlík, A. Reinke, V. Krátký, S. Zhao, A.-a. Agha-mohammadi, K. Alexis, and L. Carlone, "Present and future of slam in extreme environments: The darpa subt challenge," *IEEE Transactions on Robotics*, vol. PP, pp. 1–20, 01 2023.

[29] K. Ebadi, M. Palieri, S. Wood, C. W. Padgett, and A. akbar Agha-mohammadi, "Dare-slam: Degeneracy-aware and resilient loop closing in perceptually-degraded environments," *Journal of Intelligent & Robotic Systems*, vol. 102, 2021.

[30] M. F. Ahmed, K. Masood, and V. H. J. Fremont, "Active slam: A review on last decade," *Sensors (Basel, Switzerland)*, vol. 23, 2022.

[31] M. Hsiao, J. G. Mangelson, S. Suresh, C. H. Debrunner, and M. Kaess, "Aras: Ambiguity-aware robust active slam based on multi-hypothesis state and map estimations," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5037–5044, 2020.

[32] M. T. Banich and R. J. Compton, *Cognitive Neuroscience*, 4th ed. Cambridge University Press, 2018.

[33] K. M. Lynch, "The image of the city," 1960.

[34] T. Eliav, S. Maimon, J. Aljadeff, M. Tsodyks, G. Ginosar, L. Las, and N. Ulanovsky, "Multiscale representation of very large environments in the hippocampus of flying bats," *Science*, vol. 372, 2021.

[35] L. R. Squire, "The legacy of patient h.m. for neuroscience," *Neuron*, vol. 61, pp. 6–9, 2009.

[36] J. O'Keefe and L. Nadel, "The hippocampus as a cognitive map," 1978.

[37] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, "Microstructure of a spatial map in the entorhinal cortex," *Nature*, vol. 436, pp. 801–806, 2005.

[38] G. Ginosar, J. Aljadeff, Y. Burak, H. Sompolinsky, L. Las, and N. Ulanovsky, "Locally ordered representation of 3d space in the entorhinal cortex," *Nature*, vol. 596, pp. 404 – 409, 2021.

[39] C. Barry, C. Lever, R. M. A. Hayman, T. Hartley, S. Burton, J. O'Keefe, K. J. Jeffery, and N. Burgess, "The boundary vector cell model of place cell firing and spatial memory," *Reviews in the Neurosciences*, vol. 17, pp. 71 – 98, 2006.

[40] Ø. A. Høydal, E. R. Skytøen, S. O. Andersson, M.-B. Moser, and E. I. Moser, "Object-vector coding in the medial entorhinal cortex," *Nature*, vol. 568, pp. 400 – 404, 2019.

[41] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2015.

[42] A. Forli and M. M. Yartsev, "Hippocampal representation during collective spatial behaviour in bats," *Nature*, vol. 621, pp. 796 – 803, 2023.

[43] L. Hermer and E. S. Spelke, "A geometric process for spatial reorientation in young children," *Nature*, vol. 370, pp. 57–59, 1994.

[44] R. M. Yoder, B. J. Clark, and J. S. Taube, "Origins of landmark encoding in the brain," *Trends in Neurosciences*, vol. 34, pp. 561–571, 2011.

[45] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 698–700, 1987.

[46] A. V. Segal, D. Hähnel, and S. Thrun, "Generalized-icp," in *Robotics: Science and Systems*, 2009.

[47] H. Yang, J. Shi, and L. Carlone, "Teaser: Fast and certifiable point cloud registration," *IEEE Transactions on Robotics*, vol. 37, pp. 314–333, 2020.

[48] M. Klukas, S. Sharma, Y. Du, T. Lozano-Perez, L. P. Kaelbling, and I. R. Fiete, "Fragmented spatial maps from surprisal: State abstraction and efficient planning," *bioRxiv*, 2022.

[49] A. Safron, O. Çatal, and T. Verbelen, "Generalized simultaneous localization and mapping (g-slam) as unification framework for natural and artificial intelligences: towards reverse engineering the hippocampal/entorhinal system and principles of high-level cognition," *Frontiers in Systems Neuroscience*, vol. 16, 2021.

[50] R. Epstein, E. Z. Patai, J. Julian, and H. Spiers, "The cognitive map in humans: Spatial navigation and beyond," *Nature Neuroscience*, vol. 20, pp. 1504–1513, 10 2017.

[51] A. O. Constantinescu, J. X. O'Reilly, and T. E. J. Behrens, "Organizing conceptual knowledge in humans with a gridlike code," *Science*, vol. 352, pp. 1464 – 1468, 2016.

[52] J. L. S. Bellmund, P. Gärdenfors, E. I. Moser, and C. F. Doeller, "Navigating cognition: Spatial codes for human thinking," *Science*, vol. 362, 2018.

[53] P. A. Dudchenko and E. R. Wood, "Splitter cells: Hippocampal place cells whose firing is modulated by where the animal is going or where it has been," in *Derdikman, D., Knierim, J. (eds) Space,Time and Memory in the Hippocampal Formation. Springer, Vienna*, 2014.

[54] N. R. Kinsky, W. Mau, D. W. Sullivan, S. J. Levy, E. A. Ruesch, and M. E. Hasselmo, "Trajectory-modulated hippocampal neurons persist throughout memory-guided navigation," *Nature Communications*, vol. 11, 2020.

[55] R. Cao, J. H. Bladon, S. J. Charczynski, M. E. Hasselmo, and M. W. Howard, "Internally generated time in the rodent hippocampus is logarithmically compressed," *eLife*, vol. 11, p. e75353, oct 2022.

[56] D. Aronov, R. Nevers, and D. W. Tank, "Mapping of a non-spatial dimension by the hippocampal/entorhinal circuit," *Nature*, vol. 543, pp. 719 – 722, 2017.

[57] J. C. R. Whittington, T. H. Muller, S. Mark, G. Chen, C. Barry, N. Burgess, and T. E. J. Behrens, "The tolman-eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation," *Cell*, vol. 183, pp. 1249 – 1263.e23, 2019.

[58] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Neural Information Processing Systems*, 2017.

[59] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "Carla: An open urban driving simulator," *ArXiv*, vol. abs/1711.03938, 2017.

[60] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Proceedings of the 2020 Conference on Robot Learning*, 2021, pp. 1147–1157.

[61] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihs, M. Yatskar, and A. Farhadi, "Robothor: An open simulation-to-real embodied ai platform," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3161–3171, 2020.

[62] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The mrs uav system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 26, pp. 1–28, May 2021.

[63] P. Anderson, A. X. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir, "On evaluation of embodied navigation agents," *ArXiv*, vol. abs/1807.06757, 2018.

[64] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sunderhauf, I. Reid, S. Gould, and A. Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," 06 2018, pp. 3674–3683.

[65] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *International Symposium on Field and Service Robotics*, 2017.

[66] J. Platt and K. Ricks, "Comparative analysis of ros-unity3d and ros-gazebo for mobile ground robot simulation," *Journal of Intelligent & Robotic Systems*, vol. 106, 12 2022.

[67] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra, "Habitat 2.0: Training home assistants to rearrange their habitat," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[68] P. Chattopadhyay, J. Hoffman, R. Mottaghi, and A. Kembhavi, "Robustnav: Towards benchmarking robustness in embodied navigation," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15 671–15 680, 2021.

[69] W. N. Greene and N. Roy, "Flame: Fast lightweight mesh estimation using variational smoothing on delaunay graphs," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 4696–4704, 2017.

[70] 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'*, 1997, pp. 146–151.

[71] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[72] A. Masoumian, H. A. Rashwan, J. Cristiano, M. S. Asif, and D. Puig, "Monocular depth estimation using deep learning: A review," *Sensors*, vol. 22, no. 14, 2022.

[73] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. P. Huang, "Openvins: A research platform for visual-inertial estimation," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4666–4672, 2020.

[74] J. Civera, A. J. Davison, and J. M. M. Montiel, "Inverse depth parametrization for monocular slam," *IEEE Transactions on Robotics*, vol. 24, pp. 932–945, 2008.

[75] T. Rouček, M. Pecka, P. Čížek, T. Petříček, J. Bayer, V. Šalanský, D. Heřt, M. Petrlík, T. Báča, V. Spurný, F. Pomerleau, V. Kubelka, J. Faigl, K. Zimmermann, M. Saska, T. Svoboda, and T. Krajník, "Darpa subterranean challenge: Multi-robotic exploration of underground environments," in *Modelling and Simulation for Autonomous Systems*, 2019, pp. 274–290.

[76] M. Y. Arafat, M. M. Alam, and S. Moh, "Vision-based navigation techniques for unmanned aerial vehicles: Review and challenges," *Drones*, 2023.

[77] K. Çelik and A. K. Somani, "Monocular vision slam for indoor aerial vehicles," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1566–1573, 2009.

[78] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 779–786, 2021.

[79] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.