Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Aerospace Engineering (36^{th}cycle)

Toward system level autonomy for planetary subglacial access missions

By

Guglielmo Daddi

******

Supervisor(s):
Prof. Sabrina Corpino

Doctoral Examination Committee:
Prof. Francesco Branz, Università di Padova
Prof. Elisa Capello, Politecnico di Torino
Dr. Sangwoo Moon, Jet Propulsion Laboratory, California Institute of Technology
Prof. Alessandro Morselli , Politecnico di Milano
Prof. Fabrizio Stesina, Politecnico di Torino

Politecnico di Torino
2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Guglielmo Daddi
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

Completing this dissertation has been a long and demanding journey, and it would not have been possible without the valuable contributions of many individuals.

First, I am very grateful for the trust and freedom given to me by my advisor, Sabrina, who never ceased her support despite the twists and bends of the first year of research.

I would like to thank all of my colleagues at Politecnico di Torino, particularly Lorenzo, for the support through a year of COVID lockdowns. Your camaraderie and encouragement were crucial during these challenging times.

I want to thank Fernando for your fantastic mentorship prior to my Ph.D., and for introducing me to Mitch Ingham and Robert Rasmussen. Their views on system-level autonomy sparked an interest in me that has grown into this dissertation. Thank you, Kalind, for infecting me with enthusiasm about EELS while it was still a prototype, and for bringing me onboard once the project started. Your vision and determination made all of this possible.

My deepest gratitude goes to my mentors Tiago, Rohan, Ashkan, and Hiro for all the helpful guidance, and invaluable feedback that have greatly improved my abilities as a researcher and engineer.

I could not have wished for a better group of colleagues and friends than those I found in the EELS project. Thank you, Mike P, Marlin, Bryson, Sarah, Rachel, Rob, Mike S, Peter, Ben, Rich, Felipe, Marcel, Martin, Eloise, Eric, Tristan, Alex, Lori, Yash, and more. Thanks to everyone's support, I learned in two years more than I could have ever imagined, and enjoyed every moment of a fast-paced and very demanding project like EELS. A heartfelt thank you also to everyone in EELS' field team for making the field tests such memorable experiences, and to all the members

of Direct Action Vertical for keeping us safe, teaching us some really cool rope techniques, and sparking an interest in caves and canyons in many of us.

Thanks to my group supervisor, Ben, for all the help and advice throughout my stay at JPL. Thank you, Jonathan and Mat, for the fantastic and instructive experience I had in the final weeks of the LINC project. I also want to thank all those who sat in the EELS lab with me: Fredrik, Patty, Jack, Jenny, and others, for creating an enjoyable workspace.

Finally, to my family, your unconditional love and encouragement have been my foundation. I am here because of you.

This research was funded by Politecnico di Torino's doctoral school and was carried out at the Jet Propulsion Laboratory, California Institute of Technology, sponsored by JVSRP and the National Aeronautics and Space Administration (80NM0018D0004).

Abstract

Future planetary exploration missions are increasingly targeting remote and challenging environments, exemplified by the icy terrains of Saturn's moon Enceladus. These environments pose significant challenges due to extensive light round-trip times from Earth, high environmental uncertainty, harsh radiation environments, and extreme terrains. Recognizing these challenges, this dissertation emphasizes the necessity of system-level autonomy capabilities for successful mission execution in such uncertain and extreme environments. The main application of this research is the EELS project (Exobiology Extant Life Surveyor) - a JPL-sponsored research effort to develop a next-generation planetary subglacial access robotic platform. This work delves into the detailed software architecture and infrastructure that was developed to support system-level autonomy for the EELS project, and it describes the decision-making algorithms that were deployed in the field and laboratory conditions. Although this research is applied primarily to the EELS project, it is broadly applicable to other floating-base robotic systems that require system-level autonomy to operate in extreme or uncertain environments. The architecture description detailed in this work encompasses a full robotics stack but focuses on mission planning, plan execution, and behavior implementation. Risk-aware algorithms for sequential decision-making under uncertainty are also outlined, with a review of pertinent literature and the formulation, engineering, and testing of several planners. This use-case that this work targets is surface mobility, where the robotic platform navigates to a user-defined goal, subject to exteroception failures. Under these conditions, high-level planning helps balance information-gaining actions with trying to reach the goal directly. This mission planning challenge is framed as a Task And Motion Planning (TAMP) problem under uncertainty. It is formulated through various planning paradigms, including a classical two-stage approach, a Partially Observable Markov Decision Process (POMDP) formulation, and a novel Chance-Constrained Mixed Integer Linear Program (MILP) formulation. Through

computational experiments, the effectiveness of these planning methods is rigorously evaluated. The MILP planner, in particular, demonstrates superior performance over other approaches and is subsequently integrated into hardware. The feasibility of this integration is showcased through laboratory testing. The algorithms and architectural considerations presented in this work are not solely applicable to planetary exploration, as the problem of creating risk-aware robotic platforms capable of operating in highly uncertain environments is widely applicable to terrestrial, marine, aerial, and planetary robotics alike.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work explores system-level autonomy architecture and algorithms, with a primary focus on enabling subglacial access missions and is applied to the Exobiology Extant Life Surveyor (EELS) project, a JPL-funded research initiative aimed at creating an autonomous snake-like robot for navigation through the sub-glacial conduits of Saturn's moon Enceladus. This endeavor holds significant promise for astrobiological exploration and understanding of extraterrestrial environments. Central to this work is the establishment of a software architecture designed for achieving system-level autonomy in extreme environments, such as icy moons. This architecture is conceptualized around a classical hierarchical robotics software stack, tailored to meet the unique challenges of space exploration. The research also encompasses a detailed exploration of the requirements for system-level autonomy in such demanding contexts. It highlights the importance of integrating task and motion planning into higher layers of autonomy, reflecting the complexity and unpredictability of extraterrestrial environments. A key innovation presented in this thesis is the formulation of a novel, risk-aware task and motion planner. This planner, based on Mixed Integer Linear Programming (MILP), offers a new and efficient approach to balancing between information seeking and trajectory length minimization in such challenging environments. Additionally, the thesis discusses the adaptation and implementation of two other risk-aware task and motion planners used as comparison and baseline for the MILP planner. These planners, one formulated as a Partially Observable Markov Decision Process (POMDP) and the other employing a classical decoupled approach, represent two conceptually distinct strategies that could be adopted to solve EELS' task and motion planning problem. A comprehensive com-

putational comparison of these three planners is also conducted, providing insights into their respective strengths and weaknesses. Additionally, this work describes the process of integrating the MILP planner onto the EELS software stack and deploying it to the EELS hardware platform in a laboratory environment.

The document is organized into six chapters, each delving into different aspects of this research. The first chapter sets the stage by discussing the motivation for sub-glacial access missions, their historical context, and the specific autonomy challenges they present. The second chapter offers an in-depth look at the EELS mission, including its hardware and software architecture, with a detailed look at the custom system-level autonomy software components. The third chapter is intended to provide the reader with a minimal theoretical framework, focused on sequential decision-making under uncertainty, necessary for understanding the subsequent chapters. The fourth chapter dives into the into the formulation of the task and motion planners used for EELS' surface locomotion. The fifth chapter presents the computational framework used to evaluate the planners formulated in Chapter 4, alongside the results from running this framework. The fourth chapter also describes a proof-of-concept hardware integration of the MILP planner onto the EELS platform. The final chapter summarizes this work's contributions, draws conclusions, and recommends future work.

1.1 Motivation for planetary sub-glacial access missions

The Icy moons Europa, Enceladus, and Titan are primary points of interest in astrobiology due to the strong evidence of liquid oceans hidden beneath the ice sheets of these gas giants' moons. [6]. Subsurface oceans, in combination with geologically active, rocky cores in direct contact with the water, lead to ideal conditions for life to emerge due to the availability of life's building blocks and environmental stability [7]. Hence, it is a primary interest in space exploration to learn more about these icy moons, in the hope of shedding light on the origin of life on Earth and answering the question: "Are we alone in the universe" [8].

Jupiter's moon Europa is the first moon to have attracted astrobiological interest due to the evidence collected by Voyager's flyby, and later the Galileo Mission

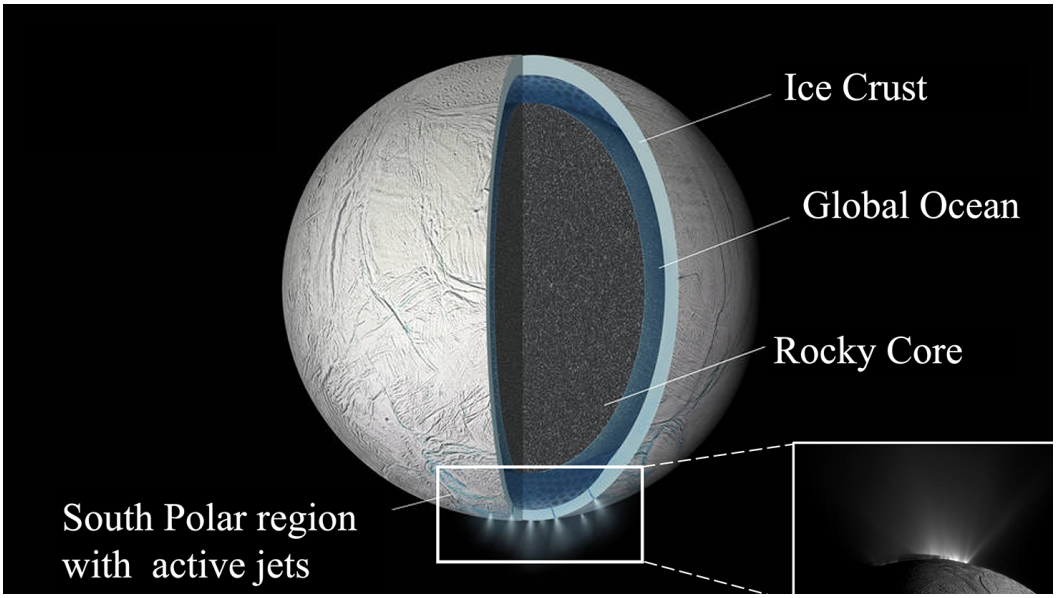


Fig. 1.1 Enceladus Illustration. Credit NASA/JPL-Caltech

[9, 10]. Multiple lines of evidence with multiple instruments all point toward the existence of a sizeable subglacial ocean with ice-sheet thickness believed to be as thin as 1 km at places [6]. The upcoming Europa Clipper flagship NASA mission is focused on studying the moon and better characterizing its subsurface environment with remote sensing techniques [11] . One aspect to note about Europa is that its surface radiation environment and lack of direct access pathways to the subglacial oceans make an in-situ ocean sampling mission a very tough technological challenge.

Saturn's moon, Enceladus, was primarily studied by the Cassini spacecraft and - like Europa - is also believed to have an extensive liquid water ocean beneath its frozen icy surface [12–16]. Similarly to Europa, Enceladus is suspected of having a direct interface between its rocky core and subsurface ocean, leading to a mineral-rich ocean [17] and hydrothermal vents [18]. A recent study has shown the presence of phosphorous in the liquid ocean. Phosphorous compounds mixed with water are an important component of biological activity on Earth, thus making Enceladus an even more compelling destination. [19] The uniqueness of Enceladus which sets it apart from all other icy moons is the presence of plumes [12, 14] that eject material from the sub-glacial ocean into Saturn's orbit. The Cassini mission was able to characterize these plumes, finding frozen salt water likely directly coming from the subglacial ocean. These plumes origin on Enceladus' South Pole from fissures known as *Tiger Stripes* (as seen in figure 1.1). Recent analysis from NASA's

Fig. 1.2 Artist impression of the Cassini spacecraft flying through Enceladus' plume. Credit NASA/JPL-Caltech

James Webb Telescope suggests that Enceladus' plumes are likely to be stable over the time scale of decades as the ejected mass flux has not varied by much since Cassini's observations. [20]. Knowledge about the vents and the surrounding surface environment is minimal, and the best orbital maps available have a resolution of $\tilde{6}$ meters per pixel, leaving considerable uncertainty about the surface's characteristics. Hence, the origin and driving mechanism of these vents are debated, with many competing hypotheses, each based on different models [1, 21–23].

Figure 1.3 shows two leading vent models. The model on the left shows an *Open Conduit* model, where a crevasse-like structure directly reaches the ocean. In this model, the liquid boils as pressure drops, and is ejected with low dynamic pressure. In the second leading model, the conduit presents a converging-diverging constriction which accelerates the liquid vapours to supersonic speeds creating a high dynamic-pressure environment. Unfortunately, the existing data from Cassini is insufficient to further constrain the vent's characteristics such as depth, geometry, flow velocity, and material properties.

With the information that is currently available, Enceladus is the most appealing body in the solar system for Astrobiological research. This is primarily due to the presence of a direct pathway from the moon's surface to the underlying ocean. In fact,

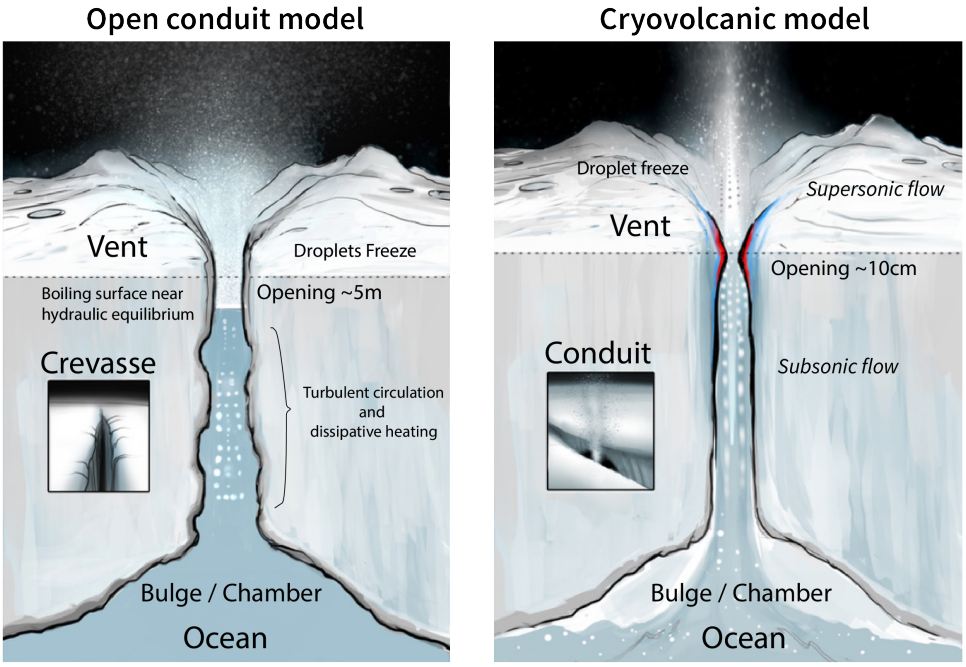


Fig. 1.3 The two leading Enceladus plume formation models. On the left is the open conduit model [1], and on the right is the vent model. Credit NASA/JPL-Caltech

the 2023-2032 Planetary Science and Astrobiology Decadal Survey [24] has placed an Enceladus Orbilander mission as the second highest flagship mission priority after a probe to the gas giant Uranus.

There are several ways an Enceladus mission might perform astrobiological research. Biosignatures might be detected by analyzing plume samples collected during a spacecraft flyby. This sampling strategy can be problematic due to the high relative velocity between particles and sampling instrument (km/s) [25]. A lander mission would be preferable as it would allow collecting and analyzing significantly larger amounts of plume material with lower relative velocity if a landing site is selected sufficiently close to the tiger stripes. This would allow the lander to be in the direct path of plume ejecta falling back to Enceladus' surface, attracted by the moon's weak gravitational pull. A lander would also enable analyzing the surroundings of the vents and the planet's geology in greater detail when compared to an orbiter. However, key astrobiological questions can only be answered by sampling the water from the ocean or the vent (as close to the ocean as possible) to minimize exposure to the harsh environment of Saturn's orbit [26, 27]. A mission to Enceladus intending to

Fig. 1.4 Examples of Cryobot designs, including Prometheus [2], Tunnelbot [3], SLUSH [4] and EnEx IceMole [5]

find life unequivocably would thus need to explore the vents and their surroundings and ideally reach and sample the buried ocean. [28]

1.1.1 Subglacial access technology

The previous section has outlined *why* it is of great interest to reach a sub-glacial ocean on our solar system's icy moons. In this section, there will be a brief review of *How* the planetary exploration community has thought about achieving this goal.

The most straightforward concept for penetrating ice sheets consists of melting through the ice sheet over time with systems called "melt probes". Alternative names can be "thermal" or "Philberth" probes. In the cryogenic ice of the solar system's icy moons, melt probes are generally known as *Cryobots* [29, 30].

Melting has been used for decades as a viable mobility technique to reach the bottom of ice sheets on Earth [31]. Typically, Cryobot mission concepts for icy moon exploration rely on Radioisotope Thermoelectric Generators (RTGs) to generate the electrical and thermal power necessary to penetrate through the ice. A main limitation of thermal ice drilling is the dependency of melting rate on power and

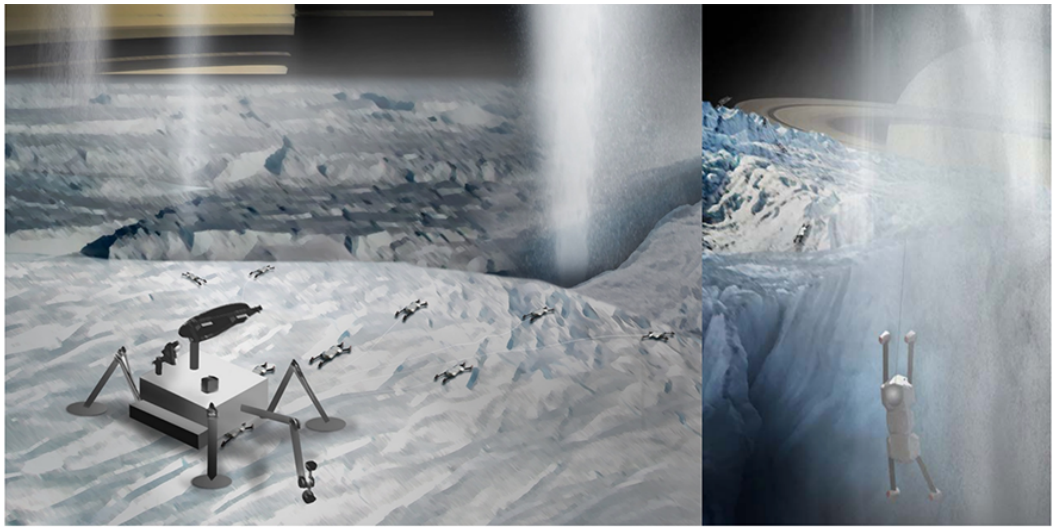


Fig. 1.5 Enceladus Vent Explorer (EVE) concept. Credit: NASA-JPL/Caltech

gravity. [32] The low-gravity environment of Europa and Enceladus vastly reduces the melting rate, and current RTG technology is relatively costly and has a power output of 100-250 watts, leading to long mission times. Long mission times on the surface of Europa are problematic due to the high radiation environment. Melting rates are even worse on Enceladus due to the low gravity, but mission duration is less of a problem because of the less extreme radiation environment.

A slow-moving system through an ice sheet that is dynamically deformed by tidal forces leads to interesting communication architecture challenges [33]. Another issue with thermal probes is the tendency of sediments to accumulate next to the probe's tip, decreasing the thermal conductivity between the probe and the ice bulk. Several projects have looked into ways to enhance thermal ice drilling and overcome their limitation through mechanical fracturing [4], water jetting [33], pulsed plasma discharges [34, 35], surface power generation and beaming through a fiber optic cable [36], mechanical screws [5], Closed Cycle hot water drilling [37]. Some examples of these cryobots can be seen in Figure1.4.

After the discovery of a direct pathway through Enceladus' vents to its subglacial ocean, there has been a growing interest in mobility systems capable of leveraging this direct path by moving *over* the ice and in the vent system rather than penetrating the ice sheet at significant energy expense. The first study to introduce this concept [38] assessed the feasibility of using a four-legged mobility system capable of surface traversal and resisting the upward pressure caused by the vents (Figure 1.5). Ice

screws are used to grip the ice wall. The main conclusion that this study reached is that reaching the ocean with this mobility strategy is feasible so long as the vent's diameter is larger than 10 cm at its tightest spot. The Enceladus Vent Explorer (EVE) architecture trade study [26] outlined the mission architecture for an Enceladus vent exploration mission. The architecture consists of a Host Module (i.e., a lander), capable of providing power and communication and a tethered vent Descent Module. The Descent module is kept intentionally generic and could be the 4-legged robot introduced in [38], or an EELS snake-like robot or more. The study concludes that such a mission is technically feasible, but significant risks are associated with environmental uncertainty. If precursor (reconnaissance) missions are not dispatched to Enceladus to better constrain the vent environment, the significant uncertainty surrounding the vent's mechanism and surface environment characteristics would require a mobility system capable of safe locomotion in a vast array of potential terrains.

1.2 System-Level Autonomy challenges

Several works in the literature have broadly outlined autonomy needs and challenges for future planetary exploration [39] [40], placing emphasis on an ever-increasing need for autonomy capabilities in uncertain environments. This section takes inspiration from these works and outlines the autonomy advancements that are needed to enable safe sub-glacial access missions. Delegating decision-making functionality to the flight system comes with risk and V&V challenges, as any decision-making algorithm is based on imperfect system and environment models. Thus, ground-in-the-loop operations are often preferred unless onboard autonomy enables some mission-critical capability.

In the ground-in-the-loop paradigm, sequences of commands are created and uplinked by ground controllers and executed by the spacecraft. Telemetry is sent back from the spacecraft and used to inform ground-based decision-making. These sequences of commands can not adapt to unexpected events, as they specify what action needs to be taken at what time. If off-nominal conditions are detected, the system is expected to enter a safe mode and await further instructions. When thinking about moving decision-making from the ground to a flight system, the most influential factor is a combination of the frequency of uplink-downlink cycles

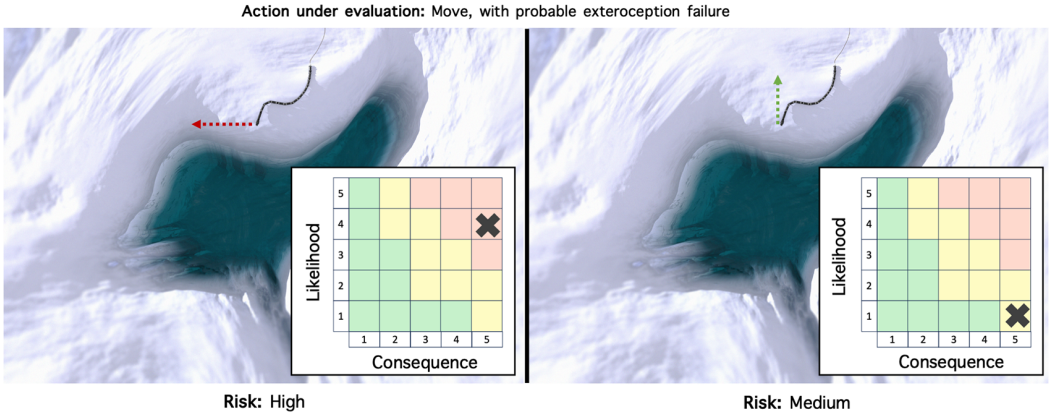


Fig. 1.6 Examples of two locomotion actions evaluated through the lens of risk. The figure shows a top-down view of the robot close to a crevasse. On the left, it is considering moving parallel to the crevasse, and on the right it is considering moving away from the edge. Both actions have a constant consequence (fall down crevasse), but the latter is much less likely to lead to failure and thus is a lower-risk action

and the time scales over which decisions must be made. The need for onboard autonomy grows as the period between successive uplink opportunities grows or the time scale over which decisions must be made decreases. Uplink opportunities are influenced by communication latency, network architecture, ground processing time, and light-speed round trip time. On the other hand, decision-making time scales are influenced by dynamical considerations (e.g., slipping down a crevasse has a time scale of X seconds, and the decision to resist that slip needs to occur in less than X) but can also be grounded in operational considerations (e.g., to reach the goal before the electronics fry, mobility decisions need to be taken with a minimum period of Y). An example of autonomy capabilities that must necessarily be onboard is Terrain Relative Navigation for Mars Landers [41], as communication latency is much larger than the whole Mars atmospheric entry phase duration.

Successfully operating a probe on Enceladus requires reacting to events that occur over time scales of minutes to seconds, with successive uplink opportunities in the scale of at least several hours, as communication with Earth will be fundamentally constrained by the two-hour-long light round trip time, and lack of robust communication relay nodes in the Saturn system. The system will also have to operate against the clock, as the radiation and temperature environments, and potentially the presence of consumable resources like batteries, could mean that the window to achieve the mission goals would likely be limited. This leads to a strong need to

delegate most of the system's decision-making to onboard algorithms.

Future exploration missions will likely operate in extreme environments like Enceladus' vents, Lunar lava tubes, Titan's surface, and more. These environments have in common a scarcity of detailed information available to mission designers. Systems operating in such environments would benefit from explicitly accounting for *Uncertainty* in their decision-making algorithms. Another concept that closely follows reasoning about uncertainty is reasoning about **Risk**. Risk is a concept used in many disciplines and can be formalized as *the likelihood of an undesirable event multiplied by the severity of the consequence of this event happening*. Therefore, risk enables reasoning about likelihood and consequence jointly, thus weighing catastrophic events that are not very probable similarly to very likely events with undesirable outcomes. Decision-making is more efficiently focused when it explicitly accounts for risk, as it minimizes the tendency of biased focus on improbable, catastrophic events.

An example of risk-aware decision-making can be seen in Figure 1.6. In the image, an agent evaluates an action through the lens of the risk of falling down a crevasse. The outcome "falling down a crevasse" has a constant high consequence outcome, but one action mitigates the risk by decreasing the event's probability, while another action subjects the agent to unreasonable risk. Another aspect to note is that the consequences of failure are not always necessarily constant. For example, a failure to turn on a floodlight might be severe when exploring a cave but completely innocuous on the surface, A mission might model several risk factors in decision-making, and a likely outcome of this risk awareness would be a less conservative behavior without increasing the probability of mission failure (modulate constraints based on the risk they pose to the mission). A less conservative system will, in turn, lead to higher scientific returns as previously unreachable areas might become reachable, and more sites might be visited given a fixed time. The current ground-in-the-loop paradigm makes it so that risk-aware decision-making is performed on the ground by operators that bake risk assessments into uplink sequences.

Other characteristics that are needed from an autonomy system on Enceladus are *Adaptability*, *Reactivity*, and *Resiliency*. Adaptation is needed both as a way to modulate behavior when more information about the environment is perceived through observation and as a way to be prepared to operate over a vast array of potential terrains (e.g., surface, vent, ocean). The dynamic nature of Enceladus' vent

systems also justifies baking in reactivity in the software stack. Resiliency is also needed to operate in extreme, unknown environments. **Resiliency** is *the capability of operating when hardware or algorithms fail and can be built both in hardware and in software*. From an autonomy perspective, resiliency emerges from a system where algorithms can work even when a subset of sensors have failed and *Fault Detection Isolation and Recovery (FDIR)* is tightly coupled with the mission planning layer. An additional need is to enhance high-level decision-making with information about a lower-level motion planning problem. There are fundamental system-level tradeoffs between information-gaining and risk-taking that can only be optimized with high-level cognizance of the system's motion. Decoupling motion planning from task planning is indeed possible but can lead to significant plan suboptimality.

1.3 Related system-level autonomy work

The autonomy challenges described above are extensive and will require many technological breakthroughs - especially consequence assessment for authentic risk-aware planning. The following sections will outline some previous work investigating system-level autonomy frameworks, reference architectures, and practical examples in space and terrestrial applications.

1.3.1 Frameworks and reference architectures

Many frameworks and architectures have been developed for system-level autonomy for planetary exploration. These frameworks are valuable tools that incorporate the accumulated decades of wisdom and lessons learned in a non-specific format. JPL's Mission Data System (MDS) [42] was developed at the turn of the millennium to unify flight and ground data systems and reconcile software and systems engineering for flight missions. Central to MDS is the explicit use of state and environment models, a goal-directed approach to spacecraft commanding, separation between estimators and controllers, tight integration of fault protection, acknowledging state uncertainty, and more. JPL Coupled Layered Architecture for Robotic Autonomy (CLARAty) [43] is an autonomy architecture focused on robotic mobility systems split into two layers. The two layers are a *decision-making* and a *functional* layer. The functional layer comprises all the necessary components for the robot's lower-level

functions, while the decision layer performs mission planning. This is a departure from the classical architectural separation in Planning-Execution-Functions which leads to three layers. This separation can be advantageous as it promotes tighter integration between planning and execution. FRESCO was recently published by JPL [44] and contains a set of guiding architectural principles for autonomy architecture development. The framework focuses on generality and agnosticism in software implementation. Similarly to MDS, FRESCO includes state-based goal definitions and centralized state knowledge management; in contrast, FRESCO takes a much less top-down approach to function partitioning and is, therefore, better suited for a project that wishes to integrate pre-existing software. Similarly to other autonomy architectures, fresco outlines the roles of controllers, estimators, system executives, and planners alongside their interactions.

The European Robotic Goal-Oriented Autonomous Controller (ERGO) framework [45] that aims at blending deliberative with reactive autonomy. The agent design consists of a functional layer and an executive tasked with coordinating a set of control loops that interact with each other through goals and observations. The framework is designed to facilitate integration with fault detection. The LAAS architecture [46] is another European autonomy architecture that predates ERGO. Similarly to ERGO, it is focused on robotic and spacecraft autonomy and contains elements of Fault Management, a separation of planning, execution, functions, and hardware interface.

The information-driven, risk-bounded enterprise architecture [47] is a planning and execution framework that uses the Reactive Model-Based Programming Language (RMPL) and decomposes operator intent into executable goals by orchestrating a combination of science, path, and activity planners. The framework also manages plan execution and activity/state monitoring.

1.3.2 Examples of spacecraft system-level autonomy

The first example of goal-directed system-level autonomy deployed to a flight system is Remote Agent Experiment (RAX) [48], which flew on NASA's Deep Space 1 (DS1) spacecraft [49] depicted in Figure 1.7 (left). The Remote Agent experiments included an AI automated planner and scheduler, a model-based fault identification engine, and an executive capable of dispatching plans and reacting to faults.

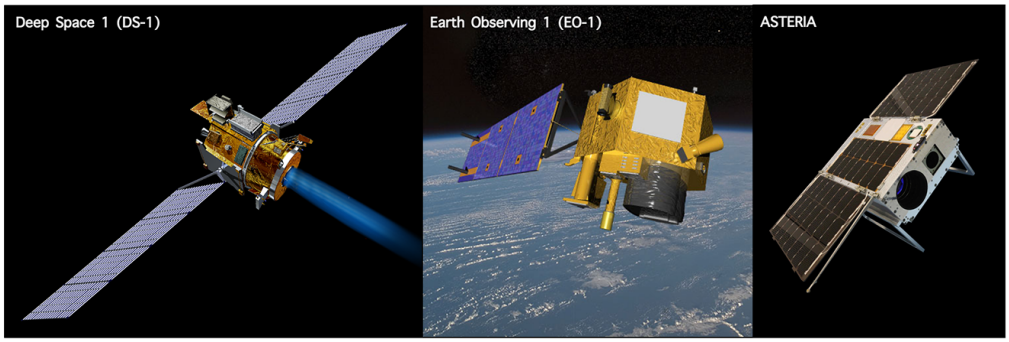


Fig. 1.7 Examples of system-level autonomy in spacecraft. Image credits: NASA-JPL/Caltech

Another example from JPL is Autonomous Sciencecraft Experiment (ASE) on the Earth Observing One (EO-1) spacecraft shown in Figure 1.7 (center), which demonstrated the capability of dynamically reacting to opportunistic science events, including volcanic eruptions, flooding, ice breakup, and cloud cover change [50]. This is achieved by combining an image science analyzing tool that detects events worth investigating with an automated planner and executive.

MEXEC [51] is an integrated planning and scheduling software developed by JPL. It represents goals as hierarchical task networks and refines them up to the point where they can be dispatched as individual actions accounting for resource timelines. [52] The ASTERIA mission is a CubeSat telescope shown in Figure 1.7 (right), used in its extended mission as a mission-autonomy testbed. The paper focuses on assessing the feasibility of autonomous orbit determination by combining the capabilities of MEXEC, an automatic navigation state estimation pipeline, and fault identification software. When the mission planner determines the need to reduce state uncertainty, a task net is issued for execution, containing all the constraints required to select the appropriate action. Action maintenance condition checking is also proven to be an effective way to adapt to failures.

While examples of end-to-end system-level autonomy deployments such as ASE, RAX, and ASTERIA are still uncommon in flight, there are many development efforts aimed at migrating system-level decision-making responsibilities from the ground system to the flight system. Some examples might be the PROBA [53], APSI [54], and EUROPA projects. Further examples can be found in [55, 56].

Historically, planetary mobility systems have not been characterized by a large system-level autonomy component. A lot of development work has gone into de-

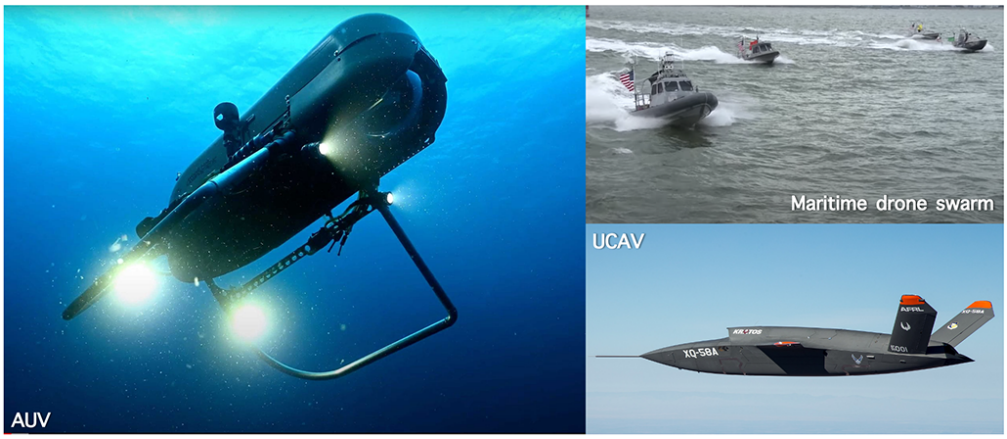


Fig. 1.8 Examples of marine and aviation system-level autonomy applications. Image credits: WHOI, NASA-JPL/Caltech, AFRL

veloping and flying sophisticated fault management systems [57], cutting edge functional-level autonomy components such as M2020's landing system terrain relative navigation [41], AutoNav [58], the Mars Exploration Rover's navigation stack [59], and more [60]. High-level autonomy does not have much flight heritage in robotic surface explorers, where the system is in a much more dynamic environment than orbiters. Large, costly and risk-averse surface exploration missions have until recently included minimal high-level autonomy and they have often been designed in a way to allow ground-in-the-loop operations. System-level autonomy has slowly been making its way into flight missions as planning technology matures (risk decreases), and the advantages of including on-board high-level reasoning start outweighing the risks. For example, the Perseverance rover relies on ground-generated activity schedules but is planned to switch to onboard scheduling later during its mission [61].

1.3.3 System-level autonomy in other domains

System-level autonomy has also been developed for several other applications (Fig. 1.8).

The closest analog to planetary exploration systems are Autonomous Underwater Vehicles (AUV) as they are subject to very similar operational constraints [62]. Similarly to planetary environments, underwater agents operate in a dynamic, unknown terrain and often have limited communication bandwidth and minimal

onboard resources to achieve a mission goal. Similarly to deep space exploration, key themes in underwater mission planning are reasoning about uncertainty, adapting to the system's health state and opportunistic goals, resource management, and more [63]. Perhaps unsurprisingly, there is significant research overlap between AUV and spacecraft mission planning, and often the same planning techniques (and same researchers) work on both problems. For example, a risk-bounded approach to mission and motion planning is outlined in [64]. Several other AUV autonomy examples are reviewed in [65].

Another thread of high-level autonomy research is directed toward enabling multi-agent autonomy, with specific attention to swarms [66]. Swarm research has also focused on flocking and formation keeping, but the mission planning problem here can be stated as allocating resources or/and assigning tasks to the agents that make up the swarm while respecting a set of constraints. Given the similarity with spacecraft mission planning, it is no surprise that many of the same techniques have been investigated (e.g., heuristic search). The primary difference between multi-agent autonomy and single-agent planetary exploration is the focus on computation distribution and information propagation. An example of maritime multi-agent autonomy architecture, where planning, execution, and behaviors appear, is the CARACaS autonomy framework [67], where all agents plan the entire mission for all agents but execute only their portion of the plan.

Another application domain where deliberative planning and autonomous execution are being actively developed is defense. Research efforts are centered on increasing the autonomy of multiple airborne [68], ground [69], and underwater systems. These systems all have to deal with the problem of breaking down a mission goal into actions, executing these actions in an uncertain and shifting environment with few communication opportunities, and reacting to faults and unexpected events are central themes.

1.3.4 Thesis objective

System-level autonomy is a vast discipline that encompasses both theoretical and experimental research. Clearly, this work can not focus on every aspect that would enable system-level autonomy. Rather, the objective of this dissertation is twofold: the development of a system-level autonomy architecture, and the creation of risk-

aware task and motion planning algorithms. This work is centered around making progress toward enabling subglacial access planetary exploration missions, particularly applied to the Exobiology Extant Life Surveyor (EELS) project, whose details will be outlined in the next chapter.

The primary focus of this work is the development of novel risk-aware task and motion planning algorithms. These algorithms are essential for decision-making under uncertainty, a critical aspect when operating in the unpredictable and harsh environments of icy moons. The research introduces a Mixed Integer Linear Programming (MILP) based planner, which offers a novel approach to balancing information-seeking actions with trajectory optimization. Additionally, two other planning paradigms are implemented and evaluated: a Partially Observable Markov Decision Process (POMDP) solved through Monte Carlo Tree Search, and a classical approach that decouples task planning from motion planning. Through computational comparisons, the effectiveness of these planners is tested, demonstrating their potential to enhance the autonomy and operational capabilities of the EELS robot. This work not only contributes to the specific goals of the EELS project but also provides valuable insights and methodologies applicable to other robotic systems operating in extreme and uncertain environments. To enable the deployment of these decision-making algorithms, this work also delves into the development of a system-level autonomy architecture. This architecture is based on a classical hierarchical robotics software stack and is tailored to meet the unique challenges posed by extraterrestrial exploration.

Chapter 2

Exobiology Extant Life Surveyor

This work focuses on making progress toward the system-level autonomy capabilities outlined in Section 1.2. While system-level autonomy is broadly applicable to multiple concepts, the framework in which this work was developed is a specific planetary ocean access research project at JPL. The project is Exobiology Extant Life Surveyor (EELS) that is a subglacial-access technology development and demonstration effort funded through the JPL-NEXT program as a way to develop the capabilities necessary to access subglacial planetary environments.

This chapter outlines the EELS' mission concept, hardware architecture, and software architecture. It is important to clarify that the author was responsible for the design and implementation of the system-level autonomy software components, beginning in section 2.4. The following sections provide essential context for understanding the system-level autonomy challenges addressed in this work.

2.1 The EELS concept

The EELS concept aims at leveraging the existing open pathway from Enceladus' surface to its subsurface ocean, assessing the environment's habitability and searching for signs of life. Using ice as a terrain over which to move sidesteps the issues of thermal drilling in cryogenic ice that plague cryobots design. The concept's high-level architecture is described in [26] and consists of a lander and a tethered serpent-like mobility system.

Fig. 2.1 EELS Concept art. Credit: NASA/JPL-Caltech

The EELS robot is designed to be a versatile, instrumented mobility platform capable of navigating the vast array of terrains that can be expected from surface and subsurface locomotion on ocean worlds. Examples of these terrains can be surface fluidized media close and much softer than fresh snow, hard ice, enclosed terrains, cracks, and ultimately also liquids.

While Enceladus is the main driver for the EELS' hardware and software architecture design and autonomy capabilities, the project has focused on building a system capable of scientific operations on Earth's glaciers which are the closest available analogs to Enceladus. Intra-Glacial and subglacial channels on glaciers are both of great scientific interest [70], are hard to reach, and are hazardous to human explorers. The inherent risks of operating in these subglacial environments makes them well suited for robotic exploration. Furthermore, there is a direct mapping from glacial environments on Earth to Enceladus exploration due to the mix of vertical and surface mobility and the materials. It is also interesting to note that the maximum dynamic pressure expected in Enceladus' vent systems results in an overall force on an EELS-like robot that is roughly equivalent to its gravitational pull on Earth.

The EELS project focuses on furthering and proving the technology that goes into developing the capabilities needed for an Enceladus vent exploration mission. The software that controls this system is also under development, with numerous parallel efforts on a wide range of topics such as proprioceptive control, gait-aware path planning, robust perception, motion planning, global localization and mapping, gait selection, scientific sensing, activity planning, and risk awareness. A reader interested in further information about the EELS concept could refer to [71].

2.2 Hardware architecture

This section is intended to give a high-level overview of the capabilities of the EELS system. An interested reader could refer to [72] for more details.

Snake-like robots have long been built and studied for Earth-based architecture, with the first systems being built in the 1970s [73]. Their high reconfigurability and capacity to traverse rough and granular terrains make them good candidates for many applications in extreme environments such as Search and Rescue (SAR) or oil and gas [74–76]. Most of the previous work in snake robot development has focused on small-scale modular platforms capable of shape-based locomotion [77]. This biologically-inspired paradigm of locomotion [78] requires an an-isotropic skin friction mechanism similar to a snake's scales [79]. In this type of locomotion, the skin is said to be *passive* as it contributes to locomotion only through its friction characteristics. The use of continuous shape changes to achieve motion has traditionally limited the size of snake robots, as actuators operating outside a quasi-static regime take an efficiency loss.

The EELS platform (Fig. 2.2) is similar to traditional snake robots in the sense that it is made of self-repeating elements but differs from many systems as it is capable both of shape change and active skin propulsion enabled through external screws. Limited research has been performed on snake robots with active skin propulsion, and even less research has focused on active-skin propulsion based on screws, but previous works have found it as promising and effective [80–83]. Specifically, EELS uses Archimedes screws for propulsion [80] and is the latest incarnation of centuries of screw-based propulsion experiments for off-road vehicles [84]. The use of screws allows to decouple propulsion from shape change and enables EELS to be much larger than typical ground-based snake robots. In fact, the

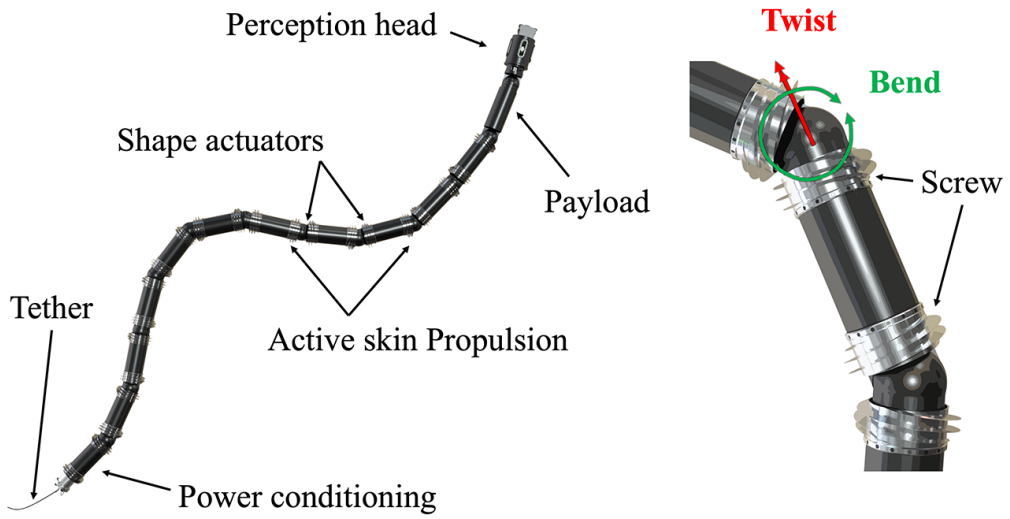


Fig. 2.2 Illustration of EELS' hardware, with a detailed view of a single module and its degrees of freedom.

platform is more than 4 meters long and weighs 100 kg. Such a length allows it to conform to the internal structure of vents with larger diameters, pushing outward to resist the flow. On the other hand, the low diameter (15 cm) allows the system to reconfigure its shape and fit into tight cracks if needed.

The robot is composed of 10 segments and a perception head. Each segment weighs around 10 kg and contains three actuators. The two **shape actuators** modify the position of the following module by rotating it along the *twist* and *bend* axes. The shape actuators have a peak torque of 400 Nm. Each module can be configured with two counter-rotating Archimedes screws actuated by a single motor. The system's modularity is also a catalyst for graceful degradation, as any single actuator or gearbox failure does not entirely compromise mobility performance. Developing algorithms and strategies for locomotion under degraded hardware conditions is outside of the scope of this project but would undoubtedly be an interesting future research direction. The screws can be quickly changed, as they are attached to the rotating chassis via fasteners. In Figure 2.3, there are examples of screw prototypes built and tested throughout the project. Screw parameters such as pitch, length, outer diameter, and material have a strong influence on the system's mobility performance, similar to how wheel parameters affect vehicles on and off the road. Smaller, CNC-machined metal screws enable good locomotion performance on ice or ice simulants.

Fig. 2.3 Screw prototypes for the EELS robot active skin propulsion

On unconsolidated media such as sand or soft snow, longer screws with larger diameters tend to perform better. For rapid iteration in unconsolidated, abrading terrain, screws were 3D printed using SLA technology.

The system's power and compute architecture is not representative of an Enceladus mission architecture, as the project is optimized for surface operations on Earth. This version of the EELS concept is intended to demonstrate the platform's mobility and autonomy capabilities - not the construction of a flight system. Electrical power is generated in Electrical Ground Support Equipment (EGSE) and transmitted to the robot's tail section via the 80 V power lines of the robot's tether. The tail section contains a power conditioning module, which steps the voltage down and dampens voltage oscillations caused by the tether's inductance. It is worth noting that there are parallel development efforts aimed at designing low-mass, high-voltage tethers, and their power-handling electronics [85]. Tether management is also outside of the scope for this phase of the project, and there are no spooling mechanisms either on the GSE or the platform's side. Opposite the tail, the robot mounts a perception head module. The tail also includes an Inertial Measurement Unit. The head is different from other modules as it does not have any actuators and mounts four stereo cameras orthogonally to each other, a single, forward-facing Ouster 3D LiDAR, an Inertial Measurement Unit (IMU), a barometer, and temperature sensors. The head also includes a compute platform whose function is processing front-end sensor data for the localization and mapping pipeline. Low-light operations are enabled both by the LiDAR and by LED boards attached to each stereo camera. Proprioceptive sensors are distributed throughout the robot's architecture. Each motor has position encoders

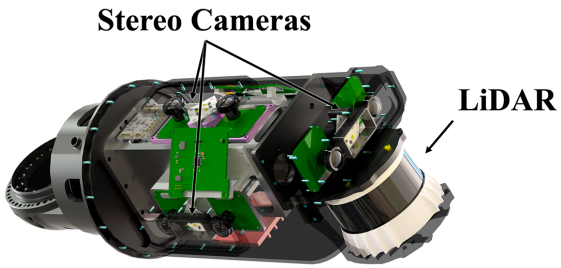


Fig. 2.4 EELS' perception head and its sensing capabilities.

and current and temperature sensors. Most computations are offloaded to Computers in Ground Support Equipment, while the only onboard computer is located in the perception head. Further iterations of the EELS robot focus on improved actuator torque density and sensing.

2.3 Software Architecture

EELS software is composed of *Controllers*, *Planners*, and *Estimators*.

Estimators *receive sensor information and use it to estimate the system's state*. **Controllers** and **planners** both *receive estimated a goal (desired states), and generate subgoals for lower-level modules (control action)*. In fact, it could be argued that controllers and planners operate fundamentally in the same way and that the distinction in terminology is primarily due to historical reasons. Controllers and planners differ in time scales and levels of abstraction and, consequently, in the type of mathematical instruments that can be utilized to derive their outputs.

Robotics stacks are typically hierarchically organized as it facilitates development and allows breaking down intractable problems (e.g., given the full robot state, achieve the mission goal) into treatable sub-problems (actuator control, path planning, etc.). EELS' software stack has been architected following this hierarchical paradigm.

The software is divided into *modules*, named with a C, followed by a number. Each C module encapsulates a specific capability, and as the number increases, the

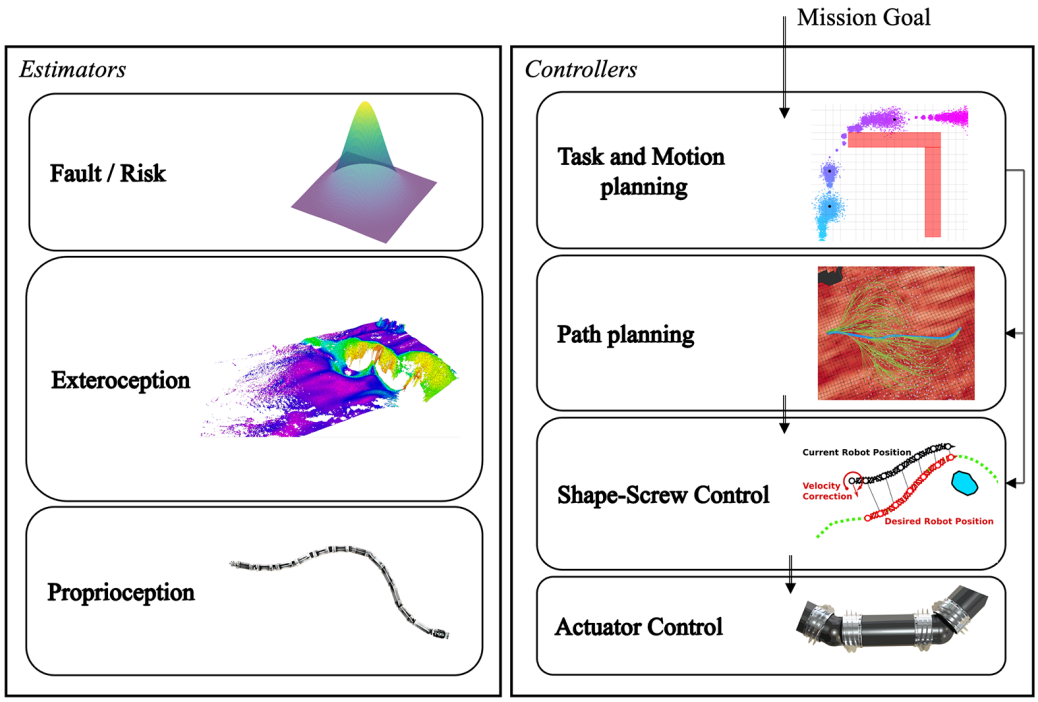


Fig. 2.5 Overview of EELS' hierarchical software architecture. Estimators are displayed on the left column, whereas Controllers are on the right.

level of abstraction of that capability also roughly increases. Capabilities range from C0 to C9, but through the project only a subset of the capabilities have been implemented. Even numbers are controllers, while odd numbers are estimators. **C0** encapsulates *Actuator Control*. This module takes as input desired joint states and figures out how to control the motors throughout the system to achieve that desired joint state. **C1** implements *Proprioception*. C1 is the lowest level estimator, and is tasked with inferring state information from sensors that have access only to state variables internal to the robot. The types of sensors that are used by C1 are encoders, IMUs, Thermocouples, Force Torque Sensors, etc. The type of information that is generated by C1 are contact states, IMU pose estimates, orientation estimates, slip detection, and more. **C2** contains most of the robot's *control* logic, in the classical sense of feedback control. Some examples are screw contact controllers used for climbing, screw velocity allocator used to translate a desired velocity into screw rotation velocities, backbone adaptation to govern the robot's shape and more. The internal architecture of C2 is complex and out of scope of this work, but it includes and orchestrates a multitude of controllers. **C3** represents the *exteroceptive state estimation* layer. This layer receives information from the robot's sensor suite

(LiDAR, stereo cameras, pressure altimeter, etc), and transfroms it into pose and map estimates, running a custom Simultaneous Localization and Mapping (SLAM) algorithm [86]. **C4** is the *Path Planning* layer and consumes map, pose and a Waypoint and outputs a path (sequence of poses) to the goal. The modules **C5**, **C6**, **C7** were not implemented as of late 2023, because the development effort was focused on mobility. The functions envisioned for these three modules are *Science Planning*, *Science Perception*, and *Gait-aware planning*. Science planning and perception refers to the function of estimating and prioritizing the potential scientific value of taking measurements from any point in the environments, and coming up with a set of points of interest to send to the mission planning layer. The gait-aware planner is supposed to provide a path plan that accounts for the multiple mobility strategies that can be adopted by the EELS platform. **C8** is the mission planning and execution layer, that sees the robot from the point of view of actions or behaviors, receives operator intent and system-level state information and generates a schedule of commands. **C9** represents the risk / health monitoring capability. It is tasked with estimating the system's health and risk, by observing state from the whole system. Its outputs are intended to guide the mission planning process and to serve as state signal for onboard fault management. It is interesting to note that the time scale of the processes observed or controlled by the modules tends to increase as level of abstraction increases. This means that the internal architecture of C8 can be vastly different from C0's architecture, as their respective latency requirements differ by orders of magnitude. Low level modules need to run in the range 10^2 - 10^3 Hz, while higher level planners and estimators run in the 1 - 10 Hz interval.

Separate from the Cx, capability modules, there are cross-cutting packages that deal with software infrastructure, software setup, user interface, software testing, and more.

Due to the field-testing intensive nature of the project, interfaces have been architected in a way that enables teleoperation at all levels of abstraction. The operator can ask for specific shapes or joint velocities, can trigger individual behaviors (e.g., go to waypoint), or load a plan and request a fully autonomous mission execution.

Networking and communication between software components is handled by the Robot Operating System (ROS) middleware. The stack runs on ROS-1 Noetic. Once viewed from a ROS centric respective, the distinction between modules become less strict. A capability module can be made of several processes, using the ROS

middle-ware both for intra and inter module communication. The line of demarcation between modules is therefore not clearly drawn in memory, but is rather a conceptual difference. Namespacing is used to help keep this conceptual distinction. For example, the prefix `/c8/node_1` and `/c8/node_2` can be two node names in the C8 capability group. This convention helps reduce confusion, decrease the chances of duplicate node names and preserve architectural integrity.

Another guiding principle of the stack's development is *middleware-agnosticism*. This means that capabilities should not be tightly coupled with ROS-specific code wherever possible and serves the purpose of facilitating portability. Middleware agnosticism is achieved by implementing capabilities as libraries using a minimal set of third-party dependencies and then writing wrappers for the ROS system. With this approach, middleware can be swapped by writing a new set of wrappers, rather than re-engineering entire capabilities (i.e., re-writing algorithms and support functions).

Within this extensive software system, this work focuses on the capabilities at the C8-C9 level.

2.4 System-level autonomy Software

When discussing High-Level autonomy (C8-C9 level) the system is seen from the point of view of behaviors or operational modes. This is the highest level of abstraction typically used to reason about the tasks that an agent needs to perform to achieve a goal. This layer of autonomy can also be seen as answering the question "What behavior should I be doing now?". Lower-level planners are invoked by the task planner to break down tasks into actionable sequences. A simple example showing the distinction between mission planners and lower-level planners is a human baking a cake. Baking has a mission goal that can only be achieved by a sequence of actions. For example, an initial plan might be to collect ingredients, then cook. These tasks need to be further broken down into driving to a grocery store, searching a sequence of areas within the store, paying, going home, laying out cooking material, then following the recipe. The mission planner is responsible of this goal breakdown into sequences of tasks. Lower-level planners in turn take care of the details of routing, controlling the steering wheel and gas pedal, avoiding obstacles, keeping balance, etc.

The three main components of a system-level autonomy software system are (a) a set of behaviors, (b) a plan execution component, and (c) a planner.

Broadly speaking, the behaviors implement an interface between the mission planning system with the rest of the software system's components. In the previous scenario, the behavior would be "go to a store", and it would consist of sending a goal to the navigation/path planning system, turning on the desired set of controllers, and monitoring their performance throughout execution. The Execution layer manages behavior dispatching and monitoring using the plan generated by the mission planner. In other words, the executive makes sure that the sequence of behaviors specified by the planner are activated in the correct order, at the correct time, and makes sure that each action successfully concludes.

The classical and most common architectural pattern in mission planning systems is a decoupled planner and executive architecture. Separating concerns leads to easier-to-implement components, more maintainability, and increases the ease of making changes to the planner or executive in isolation.

In the literature, there are examples of more tightly coupled approaches to planning and scheduling, such as the MEXEC [87].

In the EELS project, the iterative and bottom-up approach to capability development does not fit well with a tightly coupled planner and executive. New requirements arise from field tests, and the planning problem formulation could change significantly over time, driven by bottom-up discoveries. Thus, minimizing the effort required to adapt to these changes is the primary driving architectural need. For this reason, the mission planning layer is architected following a classical separation between planner and executive.

As shown in Figure 2.6, the planning and execution components are two separate components connected by an interface that can also directly be used by operators and the integration testing framework. The mission planner consumes a *mission goal*, set either by the operators or by the integration tests. The planner decomposes this goal into a sequence of activities (not necessarily a complete plan from the initial state to the goal state). The planner can dispatch full or partial plans and request the executive to start dispatching. The executive can receive plans directly from operators or integration tests, dispatches the requested behaviors sequentially, and monitors their execution. The executive provides feedback to the mission planner about the execution status.

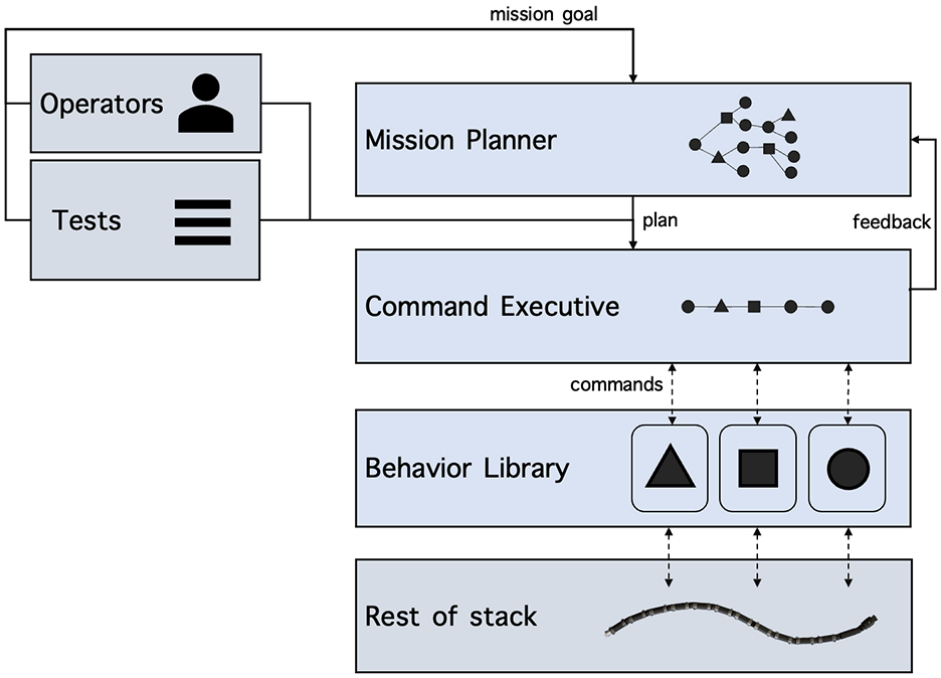


Fig. 2.6 Mission planning architecture. The main components are a Command Executive, a Planner and a library of behaviors. The planner and executive can receive goals from human operators or software integration tests. The behavior library interacts with various components of the software stack.

Behavior Library

Behaviors encapsulate the lowest level of abstraction at which the mission planner reasons. Over the span of a project, the set of behaviors that a system is capable of tends to increase as lower-level software modules are developed and matured, and the capabilities of the robot emerge from testing. Designing behaviors thus requires particular attention to *modularity*. It is convenient if behaviors have a common static interface with the execution layer. Furthermore, the interface needs to be sufficiently generic not to change over time. As an input, behaviors receive a set of parameters. Figure 2.7 shows the architecture of the behavior library that was developed for the project. Each behavior starts with a precondition check that has access to relevant state information and the goal parameters. For example, the behavior to move the robot in a moulin using vertical mobility checks that the robot is in a ready position before dispatching. All behaviors have a main execution loop, where commands are issued to the rest of the stack. When the conditions necessary to successfully exit the execution loop are met, the behavior can asses a set of post-conditions. For example, a command requesting a specific shape of the robot will assert that the encoder data received at the execution end indicates that the shape was, in fact, reached. If the post-condition check is successful, the activity *succeeds*, and reports that success back to the executive. At any point during the activity, failure conditions might be triggered. When this happens, the behavior reports the failure to the executive, alongside the reason for the failure (e.g., "the robot failed to scan the environment because one of the actuators faulted mid-execution"). Furthermore, the behavior should in general be preempt-able at any point during execution as a condition might arise that requires rapid change of plans. **Preemption**, is *the act of stopping the execution of an action, and is typically followed by a cleanup routine*. For example, the robot might be moving toward a goal and science perception might indicate that a target of opportunity has popped up next to the robot. In this case, the agent should be able to preempt the movement action and dispatch a new plan that includes a sampling behavior.

EELS' software stack uses ROS to manage communication. Thus, setting goals, sending preemption requests, and receiving feedback can be greatly facilitated using the client-server architecture provided by the *ROS actionlib* package. Using this library, behaviors are servers, and within the executive, there are clients that manage goal dispatching and feedback collection. This approach was used when designing

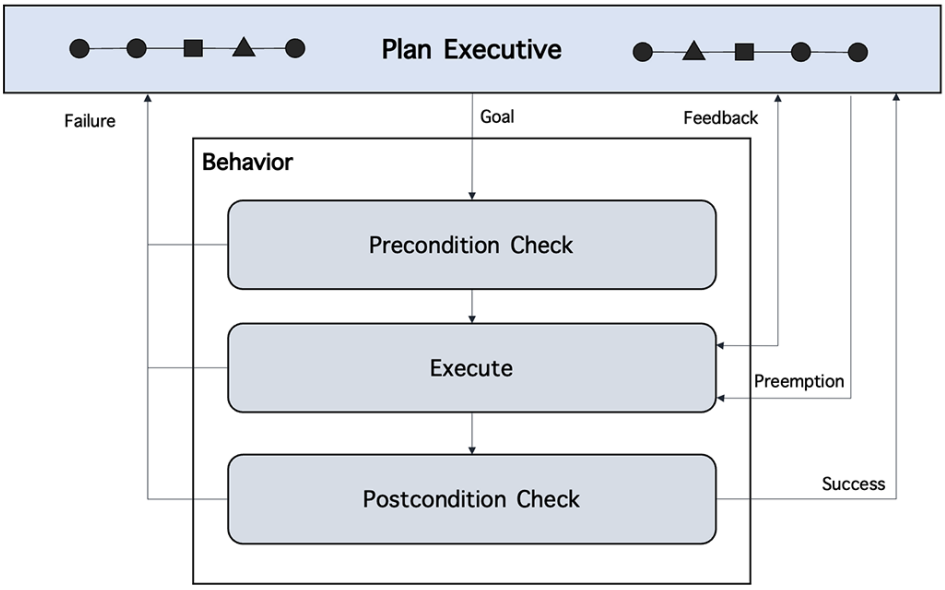


Fig. 2.7 Internal structure of the behaviors in the behavior library. Each behavior interacts with the command executive and includes a precondition check, a main execution loop and a post-condition check.

the behavior library. Behaviors inherit from a generic template that implements server management logic and provides empty shells in which to add execution logic and condition checking.

Each behavior receives goals and provides feedback in the form of strings. This is done for the sake of type consistency, maximize action representation flexibility and avoiding continuous interface re-definition. In fact, dictionaries of generic data types and more complicated objects can be embedded and effortlessly decoded as `json` or `yaml` strings and then parsed on the server's side with the aid of a parsing library.

For the EELS robot, most behaviors relate to mobility modes. This is unsurprising as the initial phases of the project focus on maturing the mobility system. Examples can be seen in Figure 2.8. For surface mobility, commonly used behaviors are `move_to_goal` and `move_to_direction`. With these behaviors, the robot can plan and execute a path to a waypoint in a closed or open-loop manner. When no exteroception is available, moving in a direction is allowed by sending velocity commands to the lower-level controller. Subsurface mobility-specific behaviors such as that `contact_subsurface` were also implemented. These behaviors make sure

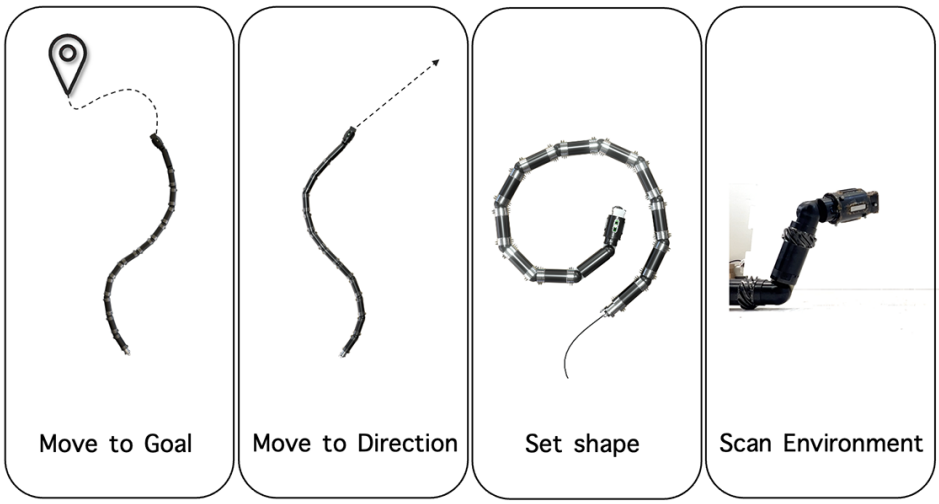


Fig. 2.8 Example of behaviors implemented for the EELS robot.

that the lower-level subsurface controllers are operating in the correct mode at all times. The `set_shape` behavior can be invoked to achieve explicit changes in shape. This was implemented as common shapes are set to begin runs or as preconditions for other actions. The `scan_environment` action implements many ways to raise the robot's head above the ground and gain information about the surrounding. This action is central to the following move-scan mission planner formulations and thus will be described in more detail in the next chapter. For commands that act on the robot's operating system, there are `os_commands` that can execute generic, user-defined scripts.

Executive layer

As mentioned earlier, the execution layer receives a *plan* and manages dispatching and monitoring of the behaviors included in that plan. The first decision to make when designing an execution (and planning) layer is the *plan representation*. The plan representation represents how the behaviors that make up the plan are connected, what the properties of these connections are, how many commands can be concurrently dispatched, and whether the time of dispatching activities is essential.

Plan representation conditions have wide-ranging effects on the system's capabilities and the effort required to implement and maintain the executive.

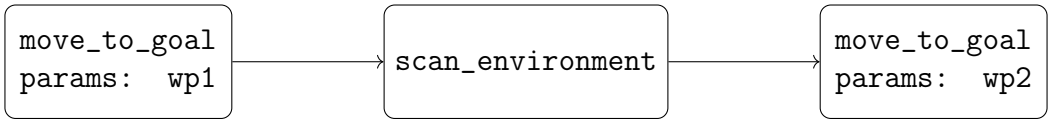


Fig. 2.9 Example of a plan as a sequence of actions

Plans, in general, can be seen as graphs of actions. Some representations can be complex and allow parallel execution and conditional actions. An example of such a "plan representation" is the Business Process Modeling Notation (BPMN) implementation of TRACE, developed for the SubT DARPA grand challenge [88]. The execution layer might also opt to represent a *Policy* or conditional plan. Policies are mappings between state and actions ($\pi : X_k \rightarrow a_k$). Policies can be computed offline and provide more execution flexibility as they adapt by definition to unexpected consequences of actions. A widespread plan representation for spacecraft autonomy is through *timelines* [89]. Timelines support the parallel execution of actions and represent complex time-based relationships between events (e.g., instrument X will turn on only after the temperature reading in sensor Y reaches value Z).

During the first years of the EELS project, the mission planning problem did not require complex time-based plans or parallel execution. Time-based scheduling was deemed unnecessary as the types of decisions being taken relate to mobility modes in uncertain terrain, thus creating complex behavior schedules would come with a greater implementation cost with minimal effect on onboard decision making effectiveness. Furthermore, the type of action space being considered for the first years of the EELS project does not require concurrent execution as most actions are mutually exclusive (e.g., the robot can not be scanning the environment while moving toward a goal). For this reason, it was deemed sufficient to represent the plan as a path or linear graph. Path graphs are acyclic graphs where all nodes have at most two vertices connected to them. A path graph is a simple way to describe a sequence of behaviors that do not have complex dependencies. An example of a path graph could be moving to a waypoint (wp1), scanning the environment, and then moving to another waypoint (wp2) (Fig. 2.9). This example can also be seen in yaml format below:

```
plan:
   - name: move_to_goal
     pose: [x1,y1,z1]
```

```
- name: scan_environment
- name: move_to_goal
  pose: [x2,y2,z2]
```

A custom execution engine was built from the ground up as full control over its development would enhance flexibility. In Figure 2.6 it is shown how the executive was designed to support both mission planning, but also high-level teleoperation, and as the backbone of system-level software integration tests. Figure 2.10 shows the internal structure of this executive which corresponds to a finite state machine. The entry point to the program is an idle state, where the executive is waiting for a plan and for a request to start executing. When an execution start is commanded, the executive will loop through the actions contained in the plan and will dispatch them sequentially using actionlib clients connected to the behavior servers. The executive periodically checks the execution status of the active server and waits for it to either finish, fail, or timeout. At any time, asynchronous callback may request a pause or a plan execution end. When a request is detected, the system preempts active servers, cleans up the plan if necessary, and resets. If a behavior server reports a failure, the executive transitions to a failed state, reporting information about the failure (when it occurred, what task caused the failure, and any additional failure meta-data provided by the behaviors) and waits for new directives. Some mission executives (notably the MEXEC execution system) include elements of decision-making. For example, MEXEC will allow for specific recovery behaviors to be triggered by task failures. The EELS executive does not include any decision-making. On the other hand, the low computational complexity of managing a linear sequence, dispatching commands, and checking for execution status allows the executive to run at frequencies well above 10 Hz, providing fast responses to callbacks and plan execution requests. The executive is entirely agnostic to the number and exact specification of actions, as each behavior action client shares the same interface, and the action specs are treated as parameter strings that contain no actionable information for the executive. This allows to modify the behaviors without interfering with the executive's code. Removing decision-making elements from the execution loop also decreases the number of changes to the executive over time, allowing the rest of the stack to rely on executive functions (e.g. preemption management) as a central part of the robot's operations.

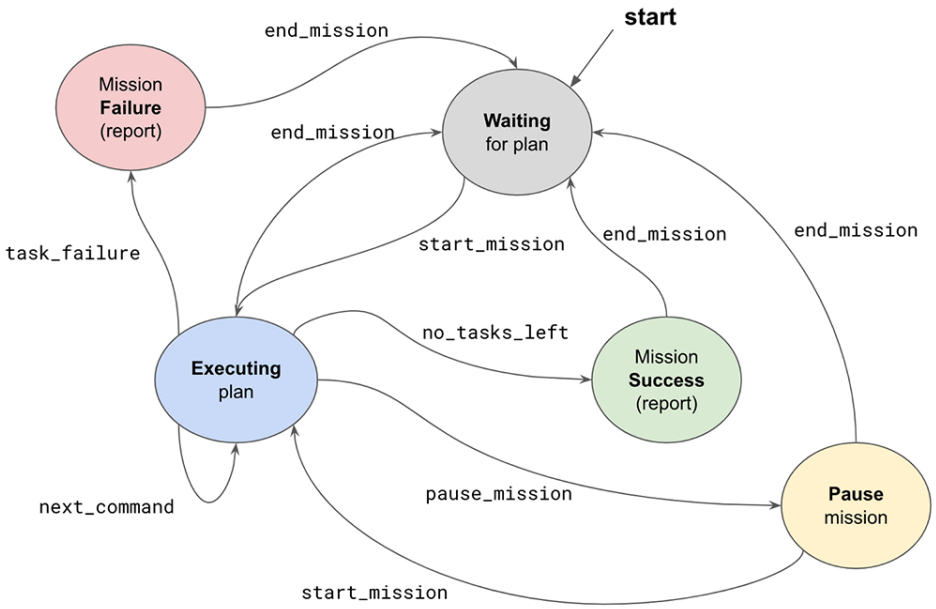


Fig. 2.10 Diagram of the execution engine's internal architecture

To further increase modularity, a ROS node wraps around the execution engine, and any signal to and from the executive is passed through the ROS network. An library was developed to facilitate the integration of command execution with the software capabilities interested in sending commands to the system. This library abstracts away from its user the management of network communication to and from the executive, providing a lexicon of commands and command-management methods that can be utilized to set a plan, start execution, preempt, get information about the execution and more. Facilitating network communication through a well-documented API increases productivity and reduces the chances of errors. A simple Python example of setting a single action plan should be sufficient to showcase the helpfulness of providing an executive API. Without an API, any user of the executive's capabilities would have to manually set up communication and correctly format the message according to the executive's representation. The following code would be necessary to send a command over the network without an external API:

```
import rospy
import json
plan_client = rospy.ServiceProxy(/plan, msgtype)
plan_client.wait_for_server()
```

```
command = {"plan":[{"name":"move_to_goal", "pose":[x,y,z]]}}
plan_client(json.dumps(command))
```

Note that the developer needs to be aware of networking details (know that the planner uses a service to accept plans, know what the service is called, the service type, etc.). Additionally, the user and needs to know how that the plan is formatted as a json string and the exact structure of this string. Here, any change in the command setting structure would need to be propagated in any place that used that capability. It is not unreasonable to assume that the topic, message type, or interface architecture might change over time. But, if any interface change would entail propagating the changes to numerous other software modules, there would be significant resistance to changes that might prove helpful. An interface fully abstracts away the network communication management from the user, hidden behind the API's implementation. The API implementation might change, but as long as the syntax remains unchanged the effects will not be felt by the API users. For example, the command-setting code outlined above could become a single line, much less prone to errors:

```
from execution import executive_interface
executive_interface.send_command(Command.move_to_goal(params))
```

Planning layer

Before describing the mission planner's architecture, it is worth outlining some of the requirements. The AGILE [90] approach to field robotics that the project followed led to a strong emphasis on requirement discovery and bottom-up capability development. This emphasis on rapid iteration can lead to changes in the planning paradigm and algorithm specifics. Due to the changing environment of fast-paced research project, it is essential to design an architecture that is general and easily modifiable. Without a robust architecture, changes gradually decrease the code's maintainability and eventually lead to an unusable system - regardless of the underlying planning algorithms and their quality. It can be thus stated that the primary driver for the Mission Planning layer's architecture is *Flexibility* to change.

In general, a mission planner should receive a goal, break the goal down into one or more actions based on the system's current state, dispatch these action/s to the execution layer, and monitor execution. Note that this way of describing the mission

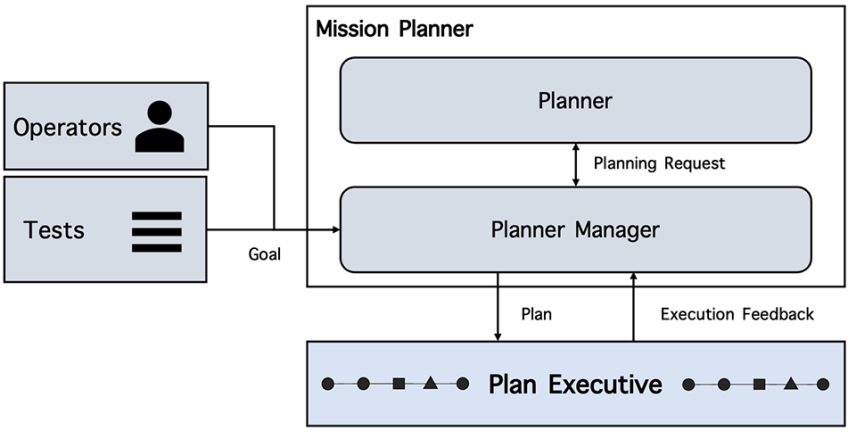


Fig. 2.11 Mission planner internal architecture

planning layer encompasses both planners that evaluate a *Policy* $\pi(a_k|s_k)$, or a *Plan* $\Pi = (a_1, \ldots, a_n)$. The concept of a Plan is quite intuitive and represents a sequence of actions. A policy on the other hand is a mapping from state to action and can be seen as a conditional plan.

In EELS' architecture, the central element of the mission planning layer is a *Mission Planner Manager* node that takes care of goal management, plan dispatching, and execution monitoring. Goals can be crafted either by operators or by integration testing. Once a goal is received, the planner manager evaluates whether the goal needs to be elaborated. If elaboration is necessary, the planner manager sends a planning request to the *Planner* component through a ROS actionlib interface. The Planner is a generic component that implements this planning interface. Each planner needs to implement goal parsing, accessing relevant state information, a planning algorithm, and response encoding. By keeping goal and state parsing responsibility within the Planner nodes, the system becomes more flexible as the planner manager is allowed to have minimal understanding of goal and plan representations. The planner manager is thus robust to goal representation changes and will encapsulate execution monitoring logic that would require effort to modify after each change. It should also be clear how this architecture can be used both for classical planning and Policy evaluation. The only change is the number of actions that make up the Planner's response. It is also worth noting that code duplication at the Planner nodes can be minimized by writing a template class designed to wrap around a generic planning algorithm and manage goal parsing and response crafting. The mission planner architecture can be seen in figure 2.11

Another design choice that affects flexibility is the data type of messages being sent over the ROS network. Specifically, the actionlib interface requires defining a message with a goal, result, and feedback. For example, the planning problem could be stated as reaching a goal respecting a specific risk tolerance . The action space of the robot might be moving to a goal, and performing information gaining actions. The result of the planning actions needs to be capable of encoding both moving to a point and non-geometric actions. The planner manager might also be interested in receiving as feedback the percentage of planning completion. An inflexible (yet computationally efficient) way to implement this interface would be to explicitly use the data type of each element of goal, result and feedback:

```
# Goal
waypoint geometry_msgs/Pose
risk_tolerance Float
---
# Result
plan geometry_msgs/Pose[]
action_type uint_8
---
# Feedback
percentage_completion Float
```

If the user wants to add an input to the planner (for example maximum planning time), the message definition would change. In ROS systems, message definition changes entail a hash change that complicates replaying historic data by requiring the correct message version locally compiled to interpret the data. Additionally, if the objective is to support multiple "Plug and Play" planners that have slightly different goal or plan representations and implement different algorithms, requiring a separate message for each planner can lead to headaches. A better solution to this problem can be seen below. Here, every element of the message is a generic string of characters, and the planners themselves have the responsibility of parsing this string into useful data structures.

```
# Goal
goal String
---
```

```
# Result
result String
---
# Feedback
feedback String
```

This implementation comes with an overhead both in terms of performance and requiring parsers on each end of the actionlib message but it is highly flexible to plan representation changes.

2.4.1 Limitations of architecture

One major component missing from the architecture outlined in this section is the fault management and risk estimation layer. A complete system-level autonomy architecture should consider fault management as an integral part of the planning problem. The planning and execution layers have been designed to be flexible, allowing the inclusion of fault estimates as inputs to the planning layer and enabling the execution of fast reactive plans. This flexibility is facilitated by the separation between the planner manager and planners, which can operate at various speeds.

The specifics of risk estimation, fault management, and fault-aware deliberative planning are left for future research. However, the architecture presented here is deemed sufficient to achieve the primary goal of this work: to formulate and test decision-making under uncertainty algorithms for the EELS mission profile. The following chapters will delve into these algorithms, which run within the planner module. From this point forward, the software architecture will serve as an invisible but necessary enabler for the remaining chapters.

Chapter 3

Theoretical Background

This chapter provides a minimum theoretical framework necessary for understanding the subsequent chapters. The core focus is on sequential decision-making under uncertainty and the tools used to formulate and solve various problems within this framework. This includes high-level onboard decision-making, managing resource constraints, and handling incomplete state information through frameworks like MDPs, POMDPs, and techniques such as Monte Carlo Tree Search (MCTS). These concepts lay the foundation for understanding the detailed algorithms and implementations in the subsequent chapters. A reader already familiar with these techniques can skip to Chapter 4.

3.1 Introduction

System-level autonomy's core responsibility is *Onboard Decision Making* at a high level of abstraction. As described in the previous chapter, the agent's action space is seen as a set of behaviors and the parameters that are passed to these behaviors. For exploration mobility systems, this might mean deciding where to go, when to communicate, when to gain information about the environment, or where to take instrument samples. In the presence of resource constraints, system-level autonomy should also coordinate resource management. For example, prioritizing what activities to do, given limited battery life, deciding when to stop all actions and recharge the batteries, and more. Given the environmental uncertainties that future

planetary missions will likely face, the decision-making process should explicitly incorporate uncertainty about the environment, system state, and world models.

Given the prominence of uncertainty, decision-making under uncertainty becomes a focal point for system-level autonomy. Foundational to this is the understanding of probability and decision theory. In the following sections, there will be a brief review the core elements of these theories, reaching up to sequential decision-making under uncertainty. These foundations are essential to formulate and solve the system-level planning problems for future planetary subglacial access autonomy systems.

Out of probability theory comes a set of tools useful to perform inference and deduction, which are necessary but not sufficient elements of decision-making under uncertainty [91]. For example, consider a system tasked with identifying and responding to a fault F, given observations Z across numerous sensors. Probability theory provides the inference tools necessary to compute the fault's probability, given the observations $p(F|Z)$, but will not prescribe when and what corrective action should be taken. Decision theory, on the other hand, is a conceptual framework capable of prescribing actions from probability distributions and preferences. Decision theory began as a comprehensive analysis of human decision-making [92], whose early objective was to understand the mechanisms underlying human decision-making processes and use them to develop effective decision-making strategies. Some of the earliest work on decision theory was spearheaded by Bernoulli and Laplace, who defined the problem within the realm of monetary returns. Their premise was that rational behavior should invariably maximize the expected return of an action. Expanding on this, they noted that a sum of money M possesses a "utility" $U(M)$, which is proportional to the logarithm of the amount:

$$U(M) \propto \log(M) \tag{3.1}$$

This simple yet profound relation underpins several key concepts in decision theory.

Firstly, utility theory sets the standard for rational decision-making, suggesting how decisions *should* be made in an ideal world [92]. It places *maximizing expected utility*, or the value associated with each possible outcome, at the heart of the decision-

making process. This encapsulates the essence of decision theory, which is concerned with making choices that optimize some form of utility.

However, additional investigations into human decision-making reveal that reality often deviates from this theoretically optimal process, a phenomenon frequently described as "irrationality" [93, 94]. This "irrationality" originates from heuristics used to speed up decision-making at the expense of optimality [95, 96]. This trade-off is a fundamental aspect of human decision-making, and understanding it is crucial for any system that seeks to model or mimic human-like decision processes.

These principles, originally developed to understand and guide human decision-making, find significant applications in the field of artificial intelligence. The lessons learnt from studying human cognitive processes have lead to a class of algorithms that enable Agents to make decisions under uncertainty. It is worth re-iterating that the classical decision theory approach (not accounting for biases and irrationality) is suitable for developing an optimal decision-making algorithm as it is a normative framework that describes how decisions *should be made*. Prospect theory better explains how humans actually make decisions and might be a good source of inspiration for approximate algorithms, but it is not necessary to consider it to develop agents. The remaining sections of this chapter are organized as follows: Firstly, the concepts and formalisms necessary to formulate and solve single and multi-action decision-making problems are briefly reviewed, and then there will be a deep dive into the algorithms developed and implemented for the EELS robotic platform.

3.2 Single decisions

Having a *Preference* is defined as desiring outcome A more than outcome B $A \succ B$. Additionally, lotteries are considered as a set of outcomes $\{O_1, \dots O_n\}$ each associated with a probability $[O_1 : p_1, \dots, O_n : p_n]$. Note that $\sum_i p_i = 1$. Adding some assumptions about preferences, enables building a normative model of rational

decision-making. These assumptions are:

$$\text{Completeness: } A \succ B \vee A \prec B \vee A \sim B \tag{3.2}$$

$$\text{Transitivity: } A \succeq B \wedge B \succeq C \implies A \succeq C \tag{3.3}$$

$$\text{Continuity: } A \succeq B \succeq C \implies \exists p \mid [A : p, B : 1-p] \sim C \tag{3.4}$$

$$\text{Independence: } A \succ B \implies [A : p, C : 1-p] \succ [B : p, C : 1-p] \quad \forall C, p \tag{3.5}$$

These constraints naturally lead to the existence of the concept of *utility function* U which follows these properties:

$$U(A) > U(B) \iff A \succ B \tag{3.6}$$

$$U(A) = U(B) \iff A \sim B \tag{3.7}$$

Note that with this definition, utility is a relative measure. Utility is not absolute, as for any $m, b > 0 \implies mU(A) + b > mU(B) + b$. That is, the preferences induced by $U' = mU + b$ are the same as for the original U. Since the scale is not absolute, it is often convenient to work with normalized utility functions that return values between 0 (the least desirable outcome) and 1 (the most desirable outcome). Therefore, the utility over a lottery L can be written as:

$$U(L) = \sum_{i=1}^{n} p_i U(O_i) \tag{3.8}$$

Let $a \in A$ be an action. Acting optimally from a utility theory perspective means selecting a in a way that maximizes the expectation of utility, given that action.

$$\arg\max_a \mathbb{E}[U(a)] \tag{3.9}$$

Let there be a probability distribution $p(O|a)$ of outcomes, given an action a. In this case, the problem of acting optimally can be framed as:

$$\arg\max_a \sum_i p(O_i|a) U(O_i) \tag{3.10}$$

3.3 Sequential decision making

Problems in robotics and aerospace can seldom be framed as single decisions. In fact, most applications involve an interplay between a decision-making agent and the environment, such that goals can only be achieved by sequences of actions. This requires extending the single-decision decision-theoretic framework outlined in the previous sections to sequential chains of decisions. This extension inevitably brings complexity and an array of computational challenges, making the issue of sequential decision-making an active area of study. Economics and AI / Controls do not have a unified lexicon, but many concepts from Utility theory can be found in other domains, albeit with slightly different names. In Utility theory, decisions are made by maximizing a Utility function, while in sequential decision-making, decisions are made by maximizing (or minimizing) a reward (or cost) over the full decision sequence. The total Utility of a sequence of actions U_s can be computed by summing the reward of each action.

$$U_s = R_1 + R_2 + \cdots + R_n \tag{3.11}$$

It is often preferable to devalue future rewards in favor of immediate rewards; this can be done via a discount factor $\gamma \in (0,1)$.

$$U_s = R_1 + \gamma R_2 + \cdots + \gamma^n R_n \tag{3.12}$$

Multi-action decision-making problems typically maximize the reward (or discounted reward) of a sequence of actions. The outcome of this optimization problem can be either a Plan or a Policy. A *Plan P* is a sequence of actions directly executed, while a *Policy* π maps state/belief to an action. $\pi : B \to A$. Policies can also be interpreted as conditional plans. Plans tend to be simpler and less flexible than policies and require online replanning when operating in the presence of uncertainty, but policies tend to have more significant computational overhead and are more challenging to communicate/verify.

3.4 The Markov Assumption

Before describing the prevalent frameworks for sequential decision-making, it is worth spending a few paragraphs describing a key simplifying assumption that underpins most sequential decision-making frameworks. The question that this assumption answers is "What information is relevant to predict state transitions?". In principle, an agent whose state s_k evolved from $s_0, s_1 \dots s_k$, after having taken actions $a_0, a_1 \dots a_k$ might want to consider the entire state and action history to predict the future. The Markov assumption on the other hand states that state evolution s_{k+1} depends only on the current state s_k and the action a_k taken in this state:

$$\Pr(s_{k+1}|s_k, a_k, s_{k-1}, a_{k-1}, ..., s_1, a_1) = \Pr(s_{k+1}|s_k, a_k) \tag{3.13}$$

This expression indicates that the future state is *only* dependent on the present, and thus past state evolution and decisions can be excluded from the decision-making process. The Markov assumption suggests that the system's entire history is encapsulated in the current state and rejects the possibility of hysteresis. Assuming this property vastly simplifies decision-making problems as it ensures that the current state and the action taken in the current state when planning for future actions are the only quantities that need to be considered. If decision-making problems had to consider the entire state's history, they would quickly become intractable.

3.5 Markov Decision Process

A prevalent framework for dealing with sequential decision-making problems under uncertainty is the Markov Decision Process (MDP). Defined as a 5-tuple, an MDP can be represented as:

$$MDP = \langle S, A, T, R, \gamma \rangle \tag{3.14}$$

In this representation, S denotes a set of states, while A represents a set of actions that an agent can perform. The transition function, $T : S \times A \to S$, determines the probability of transitioning to a new state s' given the current state s and action a, expressed as $T(s', a, s) = \Pr(s'|s, a)$. $R : S \times A \to \mathbb{R}$ serves as a reward function,

giving the expected immediate reward $R(s,a)$ for performing action a in state s. γ is the discount factor, which is used to discount future rewards.

The concept of a decision-making *horizon* is pivotal in sequential decision-making. In essence, the horizon refers to the number of steps in the future that the agent considers while making a decision. It provides a temporal boundary on the foresight of an agent, dictating the number of future states it uses for planning its actions. When considering decision-making over a finite horizon of k steps, the objective of the planning problem could be written as the maximization of expected cumulative reward within the horizon:

$$U = \sum_{t=0}^{k-1} r_t \tag{3.15}$$

However, for infinite horizon problems $k = \infty$, utility written in the form of Equation 3.15 could diverge to infinite values, making a meaningful comparison between two policies all but impossible. Making decisions over infinite horizons thus requires modifications to the expression of Utility/Reward. The most common approach is the introduction of a *discount factor* γ. The discount factor enforces the assumption that immediate rewards are preferred to delayed rewards by scaling down accrued rewards as time increases. Utility can thus be written as:

$$U = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3.16}$$

Note that γ has the intended effect of discounting future decisions only if $0 \leq \gamma < 1$. There are also other, less common approaches to formulations of infinite-horizon decision-making such as average expected reward.

3.6 Partially Observable Markov Decision Process

Sequential decision-making problems in which the agent does not have perfect state information are generally modeled as Partially Observable Markov Decision Processes (POMDPs). A POMDP extends the MDP framework to account for

incomplete state information and can be defined as a 7-tuple:

$$POMDP = \langle S, A, T, O, R, \gamma, \mathscr{Z} \rangle \tag{3.17}$$

Similarly to an MPD, POMDP describes state and action spaces (S, A). These are respectively the set of all possible states and the set of all possible actions. POMDPs are also comprised of a Transition Function $T = T(s'|s,a) : S \times A \to \Pr(S)$ and reward function $R : S \times A \to \mathbb{R}$ and a discount factor $\gamma \in [0,1)$, used to scale future rewards.

The key difference between the POMDP and MDP frameworks are *state observations*. As can be seen in Figure 3.1, it is assumed that the agent in the POMDP framework has a set of sensors that enable observations from the environment. After each action, the agent receives observations. Given the noisy nature of sensor estimates, the observation is sampled from a probability distribution. More formally, there is an *observation space O* that encapsulates all possible observations an agent might perceive. An *observation function*, $\mathscr{Z}$, given by $\mathscr{Z} : S \times A \to \Pr(O)$, provides a probability distribution over observations based on the executed action and the resulting state. This is expressed as $\mathscr{Z}(o|s,a,s')$. In the definition above, s' stands for the state after a transition. In scenarios where explicit models for T, $\mathscr{Z}$, or R are unavailable but a simulator can be used, the concept of a Generative Model comes into play: $s', o, r \leftarrow G(s,a)$. A simple example of observation space and function can be used to understand better the underlying concepts: An agent equipped with a GPS sensor might have an observation space encompassing its position in 3D space $O = \mathbb{R}^3$ and might receive direct observations of its state with some added noise $\mathscr{Z} = s + \mathscr{N}(0, \Sigma)$

A key component of the POMDP framework is the Belief State, $b(s)$, which represents a probability distribution over states and can be expressed as:

$$b_t(s) = \Pr(s_t = s|h_t) \implies \sum_{s \in S} b_t(s) = 1 \tag{3.18}$$

Where h_t is a history of actions and observations. For continuous state spaces, belief states are depicted as continuous probability density functions:

$$\int_s b_t(s) ds = 1 \tag{3.19}$$

Belief States eliminate the need for tracking the exhaustive history of actions and observations, making POMDPs solvable despite their complexity.

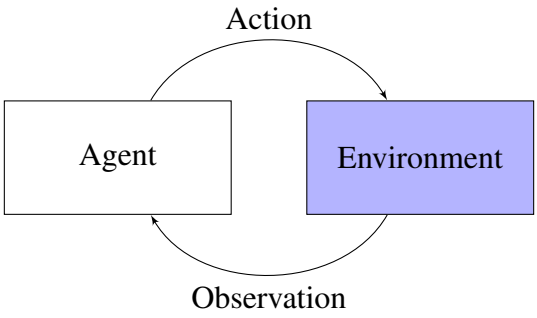


Fig. 3.1 Interaction between agent and environment in POMDP

Each time step allows the agent to receive new observations and update its belief state using an appropriate filtering technique. Despite their utility, POMDPs are generally considered intractable due to their computational complexity, making offline solutions challenging to find. Moreover, POMDPs do not differentiate between actions intended for gathering information and those intended for altering the state of the world. Instead, each decision at a time step depends on a probability distribution over states (the belief state) rather than a single state, as in the case of MDPs. This intricate interaction of the agent's beliefs, actions, and observations highlights the sophistication and applicability of the POMDP framework in sequential decision-making under uncertainty.

3.7 BeliefMDP

It is possible to frame POMDPs as MDPs over a continuous belief space. This way of casting POMDPs is knowns as Belief-MDP or BMDP and reduces the 7-tuple POMDP back to a 5-tuple. In other words, a Belif-MDP is a POMDP, seen as an MDP over Belief space rather than State space [97]. This is a trick to make POMDPs solvable without breaking the Markov assumption. The belief state is a probability distribution over states.

$$\text{BMDP} = \langle B, A, \tau, r, \gamma \rangle \tag{3.20}$$

B the set of belief states, A is the action space, τ is the belief's transition function, $r : B \times A \to \mathbb{R}$ is the reward function on belief states and γ is the discount factor. Let b be the belief state at time t, the state estimator within the agent will have to update the belief state to b' after an action is taken and an observation is received. "I have acted, what are the effects of my action?". This process will lead to a probability distribution of probability distributions (a hyperbelief). Each belief state will have $|A|$ action node successors, for each action, there will be $|O|$ possible observation. For each (a,o) there is a posterior. Similarly to filtering, the process of updating belief will se a prediction step $b_a = T(b)$, an observation $\Pr(o|a,b)$ and an update step. This transition is computed as:

$$b'(s') = \Pr(s'|o,a,b) = \frac{\Pr(o|s',a,b)\Pr(s'|a,b)}{\Pr(o|a,b)} = \frac{\Pr(o|s',a)\sum_s \Pr(s'|a,b,s)\Pr(s|a,b)}{\Pr(o|a,b)} \tag{3.21}$$

$$= \frac{\mathscr{Z}(o,s',a)\sum_s T(s',a,s)b(s)}{\Pr(o|a,b)} \tag{3.22}$$

$\Pr(o|a,b)$ normalizes the expressions and enforces the condition $\sum_s b'(s) = 1$. The probability of a new state s' will depend on the marginal probability of transitioning to that state given a belief and the probability of observing o from s'. The BMDP will have a transition function over belief space rather than state space. Hence, the POMDP's transition needs to be

$$\tau(b,a,b') = \Pr(b'|a,b) = \sum_{o \in O} \Pr(b'|b,a,o)\Pr(o|a,b) \tag{3.23}$$

For continuous spaces:

$$\tau(b,a,b') = \Pr(b'|b,a) = \int_o \Pr(b'|b,a,o)\Pr(o|a,b) \tag{3.24}$$

The probability of observing a new belief state, given action, observation and previous belief $\Pr(b'|b,a,o)$ can be written as:

$$\Pr(b'|b,a,o) = \begin{cases} 1 & \text{if } Est(b,a,o) = b' \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

Another way of writing the updated belief can be using the delta of Kronecker:

$$\Pr(b'|b,a,o) = \delta_{b'}(Est(b,a,o) = b') \tag{3.26}$$

The term $\Pr(b'|b,a,o)$ represents the conditional transition density given some observation o. Additionally, the $\Pr(\mathrm{o}|\mathrm{a},\mathrm{b})$ needs to be dealt with explicitly:

$$\Pr(o|a,b) = \sum_{s'} \Pr(o|a,s',b)\Pr(s'|a,b) = \sum_{s'} \mathscr{Z}(s',o)\Pr(s'|a,b) = \tag{3.27}$$

$$= \sum_{s'} \mathscr{Z}(s',o) \sum_{s} \Pr(s'|a,b,s)\Pr(s|a,b) = \sum_{s'} \mathscr{Z}(o,s') \sum_{s} T(s',a,s)b(s) \tag{3.28}$$

Additionally, $P(o|b,a)$ can also be derived like this:

$$\Pr(o|b,a) = \sum_{s} \Pr(o|s,a)b(s) \tag{3.29}$$

$$\Pr(o|s,a) = \sum_{s'} T(s'|s,a)O(o|s',a) \tag{3.30}$$

$$\Pr(o|b,a) = \sum_{s} \sum_{s'} T(s'|s,a)O(o'|s',a)b(s) \tag{3.31}$$

Hence, the transition function can be written as:

$$\tau(b,a,b') = \sum_{o \in \mathscr{Z}} \Pr(b'|b,a,o) \sum_{s'} \mathscr{Z}(o,s') \sum_{s} T(s',a,s)b(s) \tag{3.32}$$

$$\tag{3.33}$$

For continuous state and action spaces:

$$\tau(b'|a,b) = \int_{o} \Pr(b'|b,a,o) \int_{s'} \mathscr{Z}(o,s') \int_{s} T(s',a,s)b(s) \tag{3.34}$$

The reward function is also expressed in terms of belief:

$$r(b,a) = \sum_{s \in S} b(s)R(s,a) \tag{3.35}$$

If the state estimator is constructed from a correct observation and transition model, the belief state will respect the true state probability distribution, and $r(b,a)$ computed from belief will be correct.
BeliefMDP is on continuous space, so exact MDP solution techniques might be

inefficient. Properties of the value function can be exploited for value iteration, but, in general, all possible futures can not be enumerated.
Another way can be to solve the underlying MDP, assuming that the most likely point in b is the real state. This assumption is valid for formulations where uncertainty is not large.

3.8 Value Iteration

As the first MDP solution algorithm, it is worth introducing the classical value iteration, as it helps better understand value and utility in sequential decision making problems. The utility of a state $U(s)$ can be computed as the immediate reward for that state $R(s)$ plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s,a) U(s') \tag{3.36}$$

Equation 3.36 takes the name of *Bellman Equation*, and is the main component of Value Iteration. In fact, value iteration consists of solving the Bellman equation over all states of the system. If there are n states, there will be n equations to solve at the same time. Each Bellman equation is nonlinear, as the max operator is nonlinear. Therefore, the way to solve the large system of equations is an iterative process.
Initialize each state's Utility with a random value, then compute the left-hand side of the Bellman equation and update the Utility's value. This process is called Bellman update.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} \Pr(s'|s,a) U_i(s') \tag{3.37}$$

$$\|U_{i+1} - U_i\| < \frac{\varepsilon(1+\gamma)}{\gamma} \leftarrow \text{termination condition} \tag{3.38}$$

It is clear that this definition of value iteration requires an enumeration of all possible states that the system can occupy and is, therefore, not applicable to continuous domains without discretization. Furthermore, Value Iteration is a full-width algorithm and scales poorly as the number of states increases.
Value iteration can be adapted to POMDPs. Let the utility of a fixed conditional plan

starting from state s be: $\alpha_p(s)$. The expected utility of executing this plan will be:

$$\mathbb{E}(\alpha_p(s)) = \sum_s b(s)\alpha_p(s) = b \cdot \alpha_p(s) \tag{3.39}$$

$$U(b) = U^{\pi^*}(b) = \max_p (b \cdot \alpha_p(s)) \tag{3.40}$$

Utility varies linearly with belief given a fixed conditional plan. Therefore, the max operator will choose piece-wise segments of these hyper-planes, leading to a convex function over belief that corresponds to the union of all the dominating segments of the belief-plan expected utility curves. Let p be a conditional plan of depth d with initial action a with a sub-plan of depth $d-1$ for observation o be $p.o$.

$$\alpha_p(s) = R(s) + \gamma \left(\sum_{s'} \Pr(s'|s,a) \sum_o \Pr(o|s') \alpha_{p.o}(s') \right) \tag{3.41}$$

It is essential to prune dominated plans to mitigate the problem's growth. Quoting Russel and Norvig directly [92], Value Iteration is "*hopelessly inefficient for larger problems*." For n conditional plans at depth d, VI produces $|A|n^{|O|}$ conditional plans at depth $d+1$. Where $|A|$ is the size of the action space, and $|O|$ is the size of the observation space. Despite efforts to make Value Iteration work for continuous spaces [98, 99], more efficient sampling-based algorithms have been developed specifically to improve scalability.

3.9 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an anytime, sampling-based tree search algorithm. MCTS used for planning has a long and rich history of practical applications, especially for building game-playing AI. It has been applied to solving MDPs by [100] because of its efficiency in dealing with large problem spaces that can not be easily enumerated. The algorithm is composed of a sequence of two alternating phases - a *Tree Search* phase, followed by a *Rollout* phase. The algorithm alternates Tree Searches and Rollouts until the allocated computational resources are exhausted. The tree search phases use sparse sampling to generate a tree of alternating state-action nodes. It starts from a root node s_0; the first layer consists of action samples $\sim A(s_1)$; the second layer expands the action nodes of the first layer, sampling a state

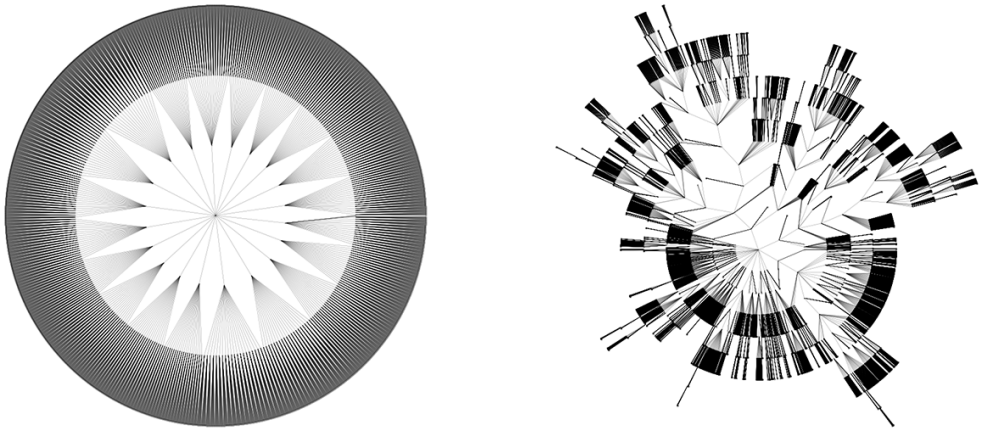
from the state distribution that arises from taking action a_1 from state s_1. Subsequent layers will repeat this pattern of alternating state-action nodes. The search phase moves along the existing tree by starting at the root note and then deciding what node to visit by finding the child node with the maximum UCT value. The meaning and formulation of UCT will be expanded later, but for now, it is sufficient to imagine choosing what child node to visit as a value maximization problem that balances exploiting promising states and exploring less-beaten paths. A Rollout is performed after a leaf node s_l is reached (a node that does not have any children) [101]. Rollouts are the application of a rollout or *Default* policy from state s_l [102]. A Rollout policy aims to estimate the value of a leaf node without exploring the tree further. This is achieved by using a pre-defined policy to select a number of actions after the leaf node. The value accrued during the rollout is treated as an estimate for the leaf node's values. There is no pre-defined way to design a rollout policy. Many options are available; rollout policies can either be deterministic or stochastic, and rollouts could even directly map leaf state to value if a heuristic is available. A well-designed rollout policy will bias the agent's decision-making toward the goal. The algorithm is recapped in Algorithm 1. In its simplest form, the time complexity of MCTS is $O(Id)$, where I is the number of iterations and d is the depth of the rollout search. Figure 3.2 shows how MCTS balances exploration and exploitation.

Algorithm 1: MCTS high level algorithm

```
1 MonteCarloPlanning(s_0 ∈ S)
2 Create root node v_0;
3 while !Timeout do
4 |   v_l = TreePolicy (v_0);
5 |   Δ = Rollout (s(v_l));
6 |   Backup (v_l, Δ);
7 end
8 return bestAction(v_0);
```

3.10 Exploration / Exploitation in MCTS

Upper confidence in Trees (UCT) and Upper Confidence Bound (UCB) are algorithms developed to solve exploration-exploitation decision-making problems. From a historical perspective, UCB emerged first from research on multi-armed bandits.



(a) Bredth First Search (b) Monte Carlo Tree Search

Fig. 3.2 Visual difference between states visited using Breadth First Search and MCTS

A multi-armed bandit is an imaginary machine with N arms. Each arm is associated with an unknown reward probability distribution, and arm pulls yield a sample $r \in [0,1]$ from that distribution. At each time step, the user can pull any arm, and the objective is to maximize reward, given a limited number of pulls. A lever pull at time t is written as $a_t \in [1,\ldots,N]$, and it represents the index of the arm being pulled. The UCB algorithm can be used to estimate the expected value $Q_{i,t}$ of the $i-th$ arm using both time t and the total number of times the arm has been chosen prior to time t ($n_{i,t}$). c is a constant that modulates exploration vs exploitation.

$$Q_{i,t} = \frac{\sum_{s=1,a=i}^{t} r_s}{n_{i,t}} \tag{3.42}$$

$$UCB_{i,t} = Q_{i,t} + c\sqrt{\frac{\ln t}{n_{i,t}}} \tag{3.43}$$

UCT extends UCB by applying it to tree exploration problems. Each node in the tree is treated like a multi-armed bandit. In MCTS, the UCB score for a node is computed for each child s' of node s, and used to select what child to move to.

$$UBC_{s'} = Q(s') + c\sqrt{\frac{\ln n(s)}{n(s')}} \tag{3.44}$$

The reward of new nodes is estimated using random rollouts. Reward is then propagated backward using an averaging backup procedure to update the estimated reward for each node in the path.

$$n(s) = n(s) + 1 \tag{3.45}$$

$$Q(s) \leftarrow Q(s) + \frac{R - Q(s)}{n(s)} \tag{3.46}$$

Chapter 4

System-level autonomy algorithms

This chapter details the core technical contribution of this dissertation, and outlines EELS' system-level planning problem and the methodologies used to address it. It is broadly divided into five sections. The first section introduces the mission planning problem. The second section frames it solely as a task planning problem cast as a Partially Observable Markov Decision Process (POMDP). The third section extends this task planning problem to encompass both task and motion planning. The fourth section presents a POMDP approach to solving this combined Task and Motion Planning (TAMP) problem. The final section formulates the TAMP problem as a novel Mixed-Integer Linear Programming (MILP) based approach.

4.1 EELS activity planning

The initial step when formulating a planner is to define the problem that needs to be solved clearly. The lack of a precise response will potentially result in a system that fails to meet the correct requirements.

In the context of the EELS system, an initial use case for system-level autonomy was the need for resilience to perception failures during surface mobility operations. The focus is explicitly set on exteroception failures caused by environment unobservability or other non-hardware-related issues. This is because this work's focus is on failures that can be recovered by changing the robot's behavior, and not just by hardware redundancy, or by proceeding with degraded localization strategies. In the event of perception failure, the robot must rely solely on proprioceptive feedback

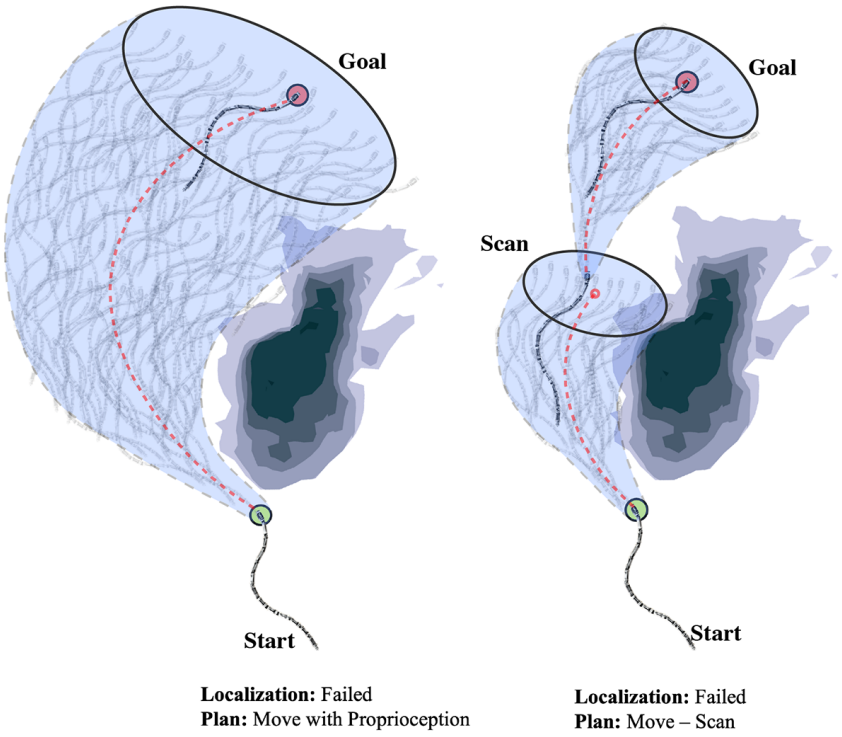


Fig. 4.1 Advantages of generating a plan that accounts for motion and scanning behaviors when compared to a plan without scans. State uncertainty is represented as a shaded blue area, and the mean trajectory as a dashed red line. The image shows how introducing a scan into the plan allows the agent to take an overall shorter path from start to goal thanks to improved state knowledge and lower collision risk with the obstacle.

(such as integrating data from an IMU and using encoders to estimate shape), leading to rapid growth of state uncertainty. Proprioceptive navigation in an environment riddled with hazards is undoubtedly perilous. However, the robot can utilize certain behaviors, like zero velocity updates [103] for the proprioception stack and scanning for exteroception, to decrease state uncertainty. The EELS system can perform scan behaviors, allowing the robot's perception sensors to move out of the degeneracy that caused the failure and relocalize against the map it has constructed. The challenge is determining the ideal moment to trigger these behaviors, as the robot's tolerance for state uncertainty should decrease as environmental hazards become more substantial. Neither the perception nor control modules possess the comprehensive knowledge to make these decisions independently. Thus, a system-level planner becomes necessary to harmonize perception and control/planning.

Having narrowed down the problem to a specific use case, the next step is to decide how to model it. The starting point is the definition of a state space. Adding too much detail to the state space will make the problem much harder to solve, so it is essential to add the least amount of information necessary to describe the problem fully and nothing more. For example, the robot's full joint state could be part of the problem's state space, alongside bus voltage and drive temperatures, but this information would not be relevant to answering the high-level question of when to perform relocalizing actions. For the level of abstraction needed, it would be feasible to consider the robot as a point mass defined by its spatial coordinates. A further decision that affects the tradeoff between model fidelity and traceability is whether these spatial coordinates are continuous or discretized:

- $X \in \mathbb{R}$ A continuous state space
- $X = \{0 \leq x \leq N, \quad x \in \mathbb{N}\}$ A discrete state space

The focus is on continuous state space for formulations to increase model fidelity. After defining the state space, the problem becomes defining an action space. The surface mobility mission planning problem has been sufficiently constrained to narrow down the action space to "moving" and "scanning." The main question is whether the movement is a binary move or scan action along a predetermined path or whether the mission planner should have greater control over its action space.

- $\mathscr{A} \in \{\text{Move}, \text{Scan}\}$
- $\mathscr{A} \in \{\mathbb{R}^n, \text{Scan}\}$

In the following sections, state space will always be continuous, but the action space will be incrementally made more complex, starting from a binary move-scan scheduling along a predetermined path to a full integrated task and motion planning formulation.

Recapping, for a 2D move scan scenario, the problem can be defined as:

$$S = \mathbb{R}^2 \tag{4.1}$$
$$A = \{U \in \mathbb{R}^2,\ Scan\} \tag{4.2}$$
$$X \in S \tag{4.3}$$
$$X_{k+1} = AX_k + BU_k + \mathcal{N}(0,\Sigma) \tag{4.4}$$
$$X_{k+1} = X_k \tag{4.5}$$
$$\Omega = \mathbb{R}^2 \tag{4.6}$$
$$\mathscr{Z} = \emptyset \tag{4.7}$$
$$\mathscr{Z} = X \tag{4.8}$$

4.2 EELS Move Scan scheduling

Let the state be a continuous random variable $X \in \mathbb{R}^{N_x}$, where N_x is the number of dimensions of the state vector. This leads to a continuous state space $S \in \mathbb{R}^{N_x}$. Let there be a discrete action space with N_a distinct actions $\mathscr{A} = \{a_1, \ldots, a_{N_a}\}$. A minimal assumption that needs to be made is that the world is Markovian. That is to say, the state at time $k+1$ is only affected by the state and the decision made at time k. Under this assumption, the action a_k will determine what transition function is applied: $X_{k+1} = f(X_k, a_k)$. Let $m(x)$ be the *stability margin* for a state $x \in S$. The concept of stability margin is equivalent to stating the probability of a state being considered unsafe. Let t_m be the time passed since the last information gain action. The move-scan problem can be modeled as a POMDP tuple as follows:

$$\text{POMDP} = \langle S, \mathscr{A}, T, O, R, \gamma, \mathscr{Z} \rangle \tag{4.9}$$

$$S = \{X, \dot{X}, t_m, m(X)\} \in \mathbb{R}^{N_x} \times \mathbb{R}^{N_x} \times [0, \infty) \times [0, 1] \tag{4.10}$$

$$\mathscr{A} = \{\text{move}, \text{stop}\} \tag{4.11}$$

$$T = p(s_{k+1}|s_k, a_k) = \begin{cases} s_{k+1} = \begin{bmatrix} X_k \\ \mathbf{0} \\ t_m \\ m(X_k) \end{bmatrix}, & a_k = \text{stop} \\ s_{k+1} = \begin{bmatrix} X_k + \dot{X}_k \Delta t \\ \pi^{C_2}(X_k) \\ t_m + \Delta t \\ m(X_k + \dot{X}_k \Delta t) \end{bmatrix}, & a_k = \text{move} \end{cases} \tag{4.12}$$

$$R_k = \mathbb{1}_{S^{goal}}(s_k) \tag{4.13}$$

$$O = \{\dot{X}, m\} \in \mathbb{R}^{N_x} \times [0, 1] \tag{4.14}$$

$$\mathscr{Z}(o^{\dot{X}}|s_{t+1}, a_t) = \begin{cases} 0 & a_t = \text{stop} \\ \dot{X}_{k+1} + \mathscr{N}(0, \Sigma_z \sqrt{t}) & a_t = \text{move} \end{cases} \tag{4.15}$$

$$\mathscr{Z}(o^m|s_{t+1}, a_t) = m(X_{k+1}) \tag{4.16}$$

$$\gamma = 0.95 \tag{4.17}$$

$$\Pr(m(X) \geq \varepsilon) \geq 1 - \Delta \tag{4.18}$$

Note that there is no uncertainty in the observation function for the stability margin. This does not mean that m_{k+1} is deterministic though, as its uncertainty will come from the uncertainty over X, not in the observation itself. Additionally, π^{C_2} is a controller policy that maps the agent's belief over state to a velocity input $\pi^{C_2} : B(s) \rightarrow \dot{X}$. In practice, this policy maps the belief's mean to a control action directed toward the goal with constant speed u.

Note that there is no uncertainty in the observation function for the stability margin. This does not mean that m_{k+1} is deterministic though, as its uncertainty will come from the uncertainty over X, not in the observation itself. Additionally, π^{C_2} is a controller policy that maps the agent's belief over state to a velocity input

$\pi^{C_2} : B(s) \rightarrow \dot{X}$. In practice, this policy maps the belief's mean to a control action directed toward the goal with constant speed u.

$$\pi^{C_2} = \frac{s^{goal} - \mathbb{E}(B(s))}{\|s^{goal} - \mathbb{E}(B(s))\|} u \tag{4.19}$$

The observation model for velocity $\dot{X}$ follows the growth trend that would be expected by integrating a white noise signal deriving from thermo-mechanical IMU fluctuations. The specifics of the growth function and uncertainty model can be easily exchanged, as they impact only the belief update step. Additionally, there are two ways of writing the chance constraint over m. The first one (which is the one included in the formulation) states that the probability of the stability margin begin in a safe range has must not be lower than a risk tolerance. The second way of stating this is a deterministic formulation. Given that m is a probability distribution, it is reasonable to use it's maximum value, or some other measure such as CVar.

$$\Pr(m(X) \geq \varepsilon) > 1 - \Delta \tag{4.20}$$

$$m \geq \varepsilon \tag{4.21}$$

Lastly, the formulation could be enhanced with chance constraints over the probability of reaching the goal. This constraint is left out of the initial formulation, as it adds needless computational complexity.

$$\mathbb{E}(\mathbb{1}_{S^{goal}}(s)) \geq 1 - \Delta \tag{4.22}$$

In practice, the agent has not enough information to infer S_k, but needs to reason about its *Belief* over the state $B(S_k)$. Reasoning over belief space, rather than state space allows us to cast this POMDP as a BMDP and solve it as a MDP. Uncertainty over position is assumed to be normally distributed $B(X), \sim \mathcal{N}(\mu^X, \Sigma^X)$. Velocity follows a similar pattern: $B(\dot{X}) \sim \mathcal{N}(\mu^{\dot{X}}, \Sigma^{\dot{X}})$. Additionally, it can be noted that there is no uncertainty over move time t, and note that uncertainty over stability

margin in general is both non-gaussian and multi-modal. The belief-MDP can be written as:

$$\text{BMDP} = \langle B, \mathscr{A}, \tau, R, \gamma \rangle \tag{4.23}$$

$$B(s) = \begin{bmatrix} \mathscr{N}(\mu^X, \Sigma^X) \\ \mathscr{N}(\mu^{\dot{X}}, \Sigma^{\dot{X}}) \\ t \\ B(m) \end{bmatrix} \tag{4.24}$$

$$\mathscr{A} = \{\text{move}, \text{stop}\} \tag{4.25}$$

$$\tau = p(b_{k+1}|b_k, a_k) = \begin{bmatrix} \mu^X, \Sigma^X \\ \mu^{\dot{X}}, \Sigma^{\dot{X}} \\ t \\ p(m(X)) \end{bmatrix}_{k+1} = \begin{cases} \begin{bmatrix} \mu^X, \Sigma^X \\ \overline{0}, \mathbf{0} \\ 0 \\ \text{Est}(p(m(X))) \end{bmatrix} & a_k = \text{stop} \\ \begin{bmatrix} \text{Est}(\mu^X, \Sigma^X) \\ \text{Est}(\mu^{\dot{X}}, \Sigma^{\dot{X}}) \\ t + \Delta t \\ \text{Est}(p(m(X))) \end{bmatrix} & a_k = \text{move} \end{cases} \tag{4.26}$$

$$R = \mathbb{E}(\mathbb{1}_{S^{goal}}(s)) \tag{4.27}$$

$$\gamma = 0.95 \tag{4.28}$$

Now, the focus can be moved on to how to perform belief updates. The transition function is trying to identify what the next belief state is, given the previous belief state and an action. For dynamics as simple as those described for the EELS use case, it is sufficient to use recursive Bayesian estimation techniques to perform the belief update step for X and $\dot{X}$.

4.2.1 Belief updates

Belief updates are needed to keep track of the agent's belief state as actions are progressively performed, and observations are received following each action. Note

that both state and observations are random variables. Transition dynamics are assumed *linear*. In general, the POMDP's state transition dynamics can be written as:

$$X_{t+1} = FX_t + GU_t + W_t \tag{4.29}$$

Where X is the state vector, F is the state transition matrix, G is the control matrix, U is the control vector, W is the process noise, t is the time step. The system can receive noisy observations from a subset of the state variables.

$$Z_t = HX_t + V_t \tag{4.30}$$

Where Z is the observation vector, H is the observation matrix, and V is the process noise. Both process and observation noise are modeled as Gaussian distributions $\mathcal{N}(\mu, \Sigma)$. The probabilities of transition and observations are conditioned on the previous state: $p(X_{t+1}|X_t)$ and $p(Z_t|X_t)$.

$$p(X_{t+1}|X_t) = Fp(X_t) + W = F\mathcal{N}(X_t, P_t) + \mathcal{N}(0, \Sigma_m) \tag{4.31}$$

$$p(Z_t|X_t) = Hp(X_t) + V = H\mathcal{N}(X_t, P_t) + \mathcal{N}(0, \Sigma_o) \tag{4.32}$$

These probabilities can be written as a Bayesian inference problem that consists of finding the posterior distribution of state, given a prior, likelihood, and observation.

$$p(X_t|Z_t, X_{t-1}) = \frac{p(Z_t|X_t, X_{t-1})p(X_t|X_{t-1})}{p(X_t)} \tag{4.33}$$

Following these considerations, the belief update can be written as a Kalman Filter, which is a simple recursive Bayesian filter. Conceptually, the Kalman filter can be divided into two operations. Firstly, a posterior prediction uses the prior belief and a system model to estimate the system's evolution. The second part of the filter is an update that incorporates the measurement into the prediction. More formally, the prediction step at time t predicts the system state at time $t+1$, denoted as $\hat{X}_{t+1}$. The system's covariance growth is also propagated by adding process noise.

$$\hat{X}_{t+1} = FX_t + Gu_t \tag{4.34}$$

$$\hat{P}_{t+1} = FP_tF^T + Q \tag{4.35}$$

The update step can be written as:

$$K_t = \hat{P}_{t+1} H^T (H P_{t-1} H^T + R_t)^{-1} \tag{4.36}$$

$$X_{t+1} = \hat{X}_{t+1} + K_t (z_t - H\hat{X}_{t+1}) \tag{4.37}$$

$$P_{t+1} = (I - K_t H)\hat{P}_{t+1}(I - K_t H)^T + K_t R_t K^T \tag{4.38}$$

$$\tag{4.39}$$

Where I is the innovation matrix, and K is known as the Kalman gain. Kalman filters can be used for systems with linear dynamics whose uncertainty distribution can be approximated by a Gaussian distribution. Non-linear dynamics can be dealt with using linearization techniques (e.g., Extended Kalman Filter). In general, problems that are characterized by multi-modal uncertainty distributions require more sophisticated belief update techniques.

4.2.2 Tracking on 2D plane with noisy velocity observations

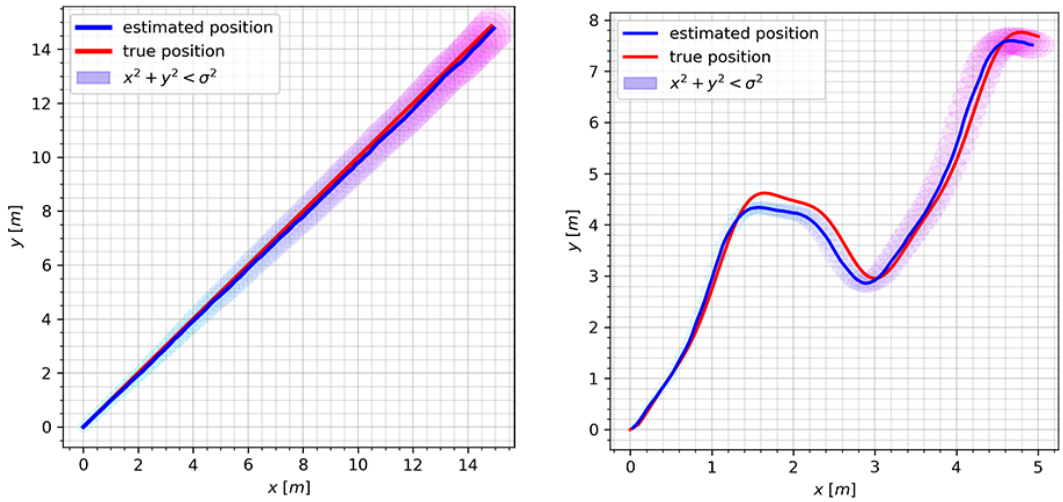
The move-scan BMDP formulation outlined in Section 4.2 presents a belief update problem that can be in part interpreted as tracking an agent on a 2-D plane, given a deterministic state transition function and measurement noise. During exteroception failures, the agent can only observe its velocity. Noise in the uncertain velocity observations can be modeled as a Gaussian distribution around a mean μ_x, with variance $\sigma_{\dot{x}}$ and $\sigma_{\dot{y}}$. The system starts from a state of low uncertainty X_0, P_0. Let (x, y) be a Cartesian reference frame, the state transitions can thus be written as:

$$X = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \quad X_{t+1} = \begin{pmatrix} x_t + \dot{x}_t \mathrm{d}t \\ y_t + \dot{y}_t \mathrm{d}t \\ \dot{y}_t \\ \dot{x}_t \end{pmatrix} \tag{4.40}$$

The other matrices that make up the Kalman Filter can be written as:

$$P = \begin{pmatrix} \sigma_x & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{yx} & \sigma_y & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{\dot{x}x} & \sigma_{\dot{x}y} & \sigma_{\dot{x}} & \sigma_{\dot{x}\dot{y}} \\ \sigma_{\dot{y}x} & \sigma_{\dot{y}y} & \sigma_{\dot{y}\dot{x}} & \sigma_{\dot{y}} \end{pmatrix} \quad F = \begin{pmatrix} 1 & 0 & \mathrm{d}t & 0 \\ 0 & 1 & 0 & \mathrm{d}t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad H = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R = \begin{pmatrix} \sigma_{\dot{x}}^2 & 0 \\ 0 & \sigma_{\dot{y}}^2 \end{pmatrix}$$

$$\tag{4.41}$$



(a) Straight-line path (b) Curved path

Fig. 4.2 Kalman filter for deterministic linear 2D motion with noisy velocity measurments

This filter is easy to implement and accurately estimates position depending on the measurements' noise. Results can be seen in Figure 4.2, where it is clear that the filter can estimate a variety of motion profiles.

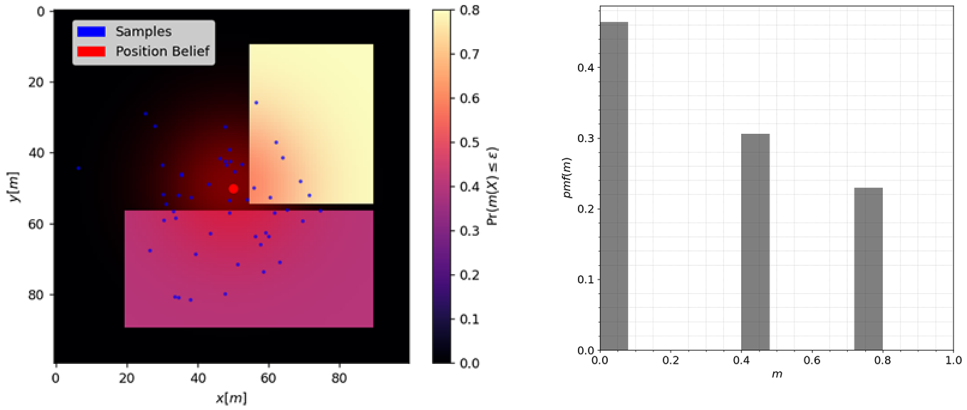
4.2.3 Estimating stability margin

In contrast with the ease of estimating state uncertainty due to motion, updating the stability margin requires more elaboration. The robot can observe the world around it and evaluate the probability that each pose exceeds the stability margin $\Pr(m(X) \geq m)$. In general, it can not be known what shape this probability distribution will follow $X \in \mathbb{R}^2$. For simplicity, though, a Bernoulli distribution is assumed $\Pr(m(X) \geq m) \sim \text{Ber}(m(s))$. When trying to find the probability distribution of m, the theorem of total probability can be used:

$$p(m) = \int_{s \in S} p(m|b(s))b(s)\mathrm{d}s \tag{4.42}$$

As can be seen from Figure 4.3, even though state uncertainty can be modeled as a Gaussian, the probability distribution of m, given position uncertainty, does not have a shape that can be known a priori. Furthermore, it can be noted that this

distribution is multimodal in the general case. In the image, the stability margin's distribution is estimated with a sampling-based approach and shows three peaks, being zero elsewhere. The first peak in the distribution refers to empty space ($m = 0$), while the second and third peaks refer to the two likely obstacles close to the agent's belief mean. One obstacle has a uniform probability of being an obstacle of 0.45, and the second obstacle has a higher likelihood of 0.75. These two are the corresponding peaks in the stability margin distribution. Since there is no closed form solution to Equation 4.42, the path to follow is to find an approximation to that solution with Monte Carlo Integration.



(a) Position belief and map of stability margin probability estimates

(b) Stability margin distribution at a given position belief

Fig. 4.3 Showcase of the stability margin's non-Gaussian, multi-modal probability distribution, given a position belief

It is also worth spending a few paragraphs explaining the termination condition. The objective of this formulation is to ensure the agent agent will make a sequence of decisions aimed at reaching a goal position with a high probability. A way to frame this question is to ask when the probability that the agent is within a circular region of radius ρ centered in the goal region (x_g, y_g) exceeds a threshold Δ, given that the agent's state probability distributions is a 2D Gaussian distribution $\mathcal{N}(\mu_x, \Sigma)$. Essentially, this problem revolves around integrating the distribution within a circular region. A good starting point is to simplify the problem to a 1D case. Cutting a circle with a non-tangential plane yields an interval, therefore the integral of the belief distribution $\mathcal{N}(0, \sigma^2)$ over the interval $[x_0 - \rho, x_0 + \rho]$ needs to be computed.

$$p(x \in x^{goal}) = \frac{1}{\sqrt{2\pi\sigma}} \int_{x_0-\rho}^{x_0+\rho} \exp\left(-\frac{x^2}{2\sigma^2}\right) \mathrm{d}x \tag{4.43}$$

This integral is easily solvable by recalling the concepts of cumulative density function $\phi(x)$ and error function for normal distributions:

$$\phi(x) = \frac{1}{2\pi} \int_{-\infty}^{0} \exp\left(-\frac{t}{2}\right) \mathrm{d}t, \;\; \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} \exp\left(-\frac{t}{2}\right) \mathrm{d}t \tag{4.44}$$

$$\tag{4.45}$$

Even though the error function is approximated numerically, there are many, well documented, efficient algorithms that do so:

$$\frac{1}{\sqrt{2\pi\sigma}} \int_{x_0-\rho}^{x_0+\rho} \exp\left(-\frac{x^2}{2\sigma^2}\right) \mathrm{d}x = \frac{1}{2}\left[\mathrm{erf}\left(\frac{x_0+\rho}{\sqrt{2}\sigma}\right) - \mathrm{erf}\left(\frac{x_0-\rho}{\sqrt{2}\sigma}\right)\right] \tag{4.46}$$

When moving to the 2D case, solving the integral over a circular region becomes impossible. Firstly, an assumption of isotropic covariance matrix is made: $\sigma_{xy} = \sigma_{yx} = 0, \;\; \sigma_x = \sigma_y = \sigma$. A further assumption is that the distribution's mean μ is located in the Cartesian coordinates $(0,0)$. Hence, the probability density function can be expressed as:

$$f(x,y) = \frac{1}{\sqrt{2\pi\sigma_x\sigma_y}} \exp\left[-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)\right] \tag{4.47}$$

Our objective is to integrate this expression over the ball of radius ρ, centered in (x_{goal}, y_{goal})

$$\iint_{B_\rho(x_0,y_0)} f(x,y)\,\mathrm{d}x\mathrm{d}y = \frac{1}{\sqrt{2\pi\sigma^2}} \iint_{B_\rho(x_0,y_0)} \exp\left(-\frac{1}{2\sigma^2}(x^2+y^2)\right) \tag{4.48}$$

There is no closed-form solution to this integral. Once again, intractable integrals as this can be approximated using Monte Carlo Integration. The approach consists of extracting N samples from the belief distribution and comparing how many of these samples fall in the goal region (N_i) against the number of samples outside the goal region (N_o).

$$\iint_{B_\rho(x_0,y_0)} f(x,y)\,\mathrm{d}x\mathrm{d}y \approx \frac{N_i}{N_o+N_i} \tag{4.49}$$

Note that as the number of samples approaches infinity, this expression becomes exact.

$$\lim_{N_0+N_i\to\infty} \frac{N_i}{N_o+N_i} = \iint_{B_\rho(x_0,y_0)} f(x,y)\,\mathrm{d}x\mathrm{d}y \tag{4.50}$$

Sampling is expensive, and since termination condition checking needs to occur any time a state is evaluated, reducing the computational burden of this operation is critical to improve solver performance. Thus, this faster yet inexact method is used to determine whether termination conditions are met. In practice, there is an additional check if the position's mean is within the goal region and enforce a constraint over the state's covariance matrix using an empirically determined threshold ε_Σ.

$$\|\mathbb{E}(B(\mathbf{x})) - \mathbf{x}_{goal}\| \leq \rho \tag{4.51}$$

$$\mathrm{Tr}(\Sigma) \leq \varepsilon_\Sigma \tag{4.52}$$

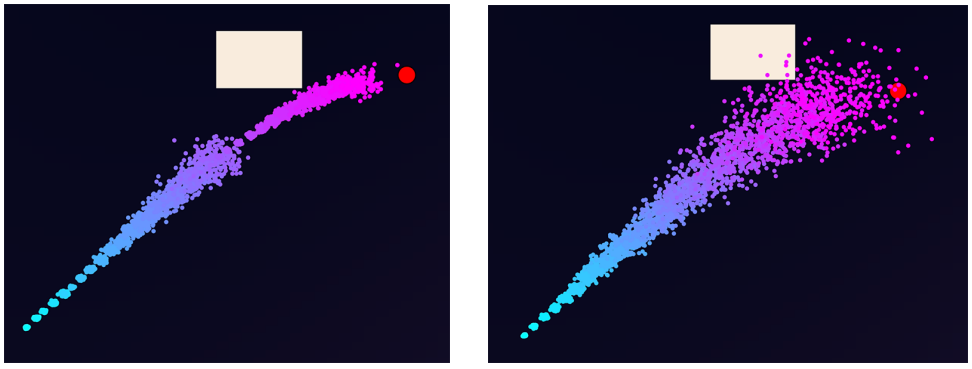
4.2.4 Solving the POMDP formulation

This model was implemented using the Julia POMDP library [104] due to its ease of use and the wide number of solvers available for benchmarking. Since the problem's state space is continuous, one-shot solving algorithms such as value iteration were avoided in favor of anytime algorithms. A variation of MCTS called MCTS-DPW [105] was selected for this work. This search algorithm distinguishes itself from vanilla MCTS by introducing the concept of *Double Progressive Widening* (DPW). In fact, MCTS is not well suited for continuous state spaces as each action node

will potentially have infinite successor state nodes. Thus, the algorithm will keep expanding state nodes that are very similar to each other and will never reach significant search depth. MCTS-DPW is designed to improve the lack of search depth that regular MCTS has for continuous state spaces by providing fine control over the search's branching factor. Two user-defined controls $\alpha, k \in \mathbb{R}$ determine whether a leaf node is expanded or not. Specifically, a node will be expanded only if:

$$n < kn^{\alpha} \tag{4.53}$$

Where n is the node's visit count. All of these aspects are illustrated in Figure 4.4, which shows how samples over the agent's position belief propagate over time under two different chance constraints. The color progression indicates time progression, where magenta represents the simulation's last step, while turquoise represents the start. The red dot is the goal, and the grey rectangle is an area with a high probability of being an obstacle. It can be observed how a low-risk tolerance will schedule an information gain action in the obstacle's proximity to ensure a low probability of intersecting that obstacle. On the other hand, a high-risk tolerance will allow state uncertainty to grow without scheduling scans.



(a) Low risk tolerance (b) High risk tolerance

Fig. 4.4 Position belief over time in the presence of obstacle. The red dot represents the goal, the other dots are samples from position belief over time, and the white box is an obstacle

Note that the action space for this planner is only activities. This means that a lower-level planner must provide the path. Since the belief propagation model is a straight-line control input toward the next goal, the inclusion of curved paths in this POMDP formulation can be achieved by breaking them up into dense sequences of

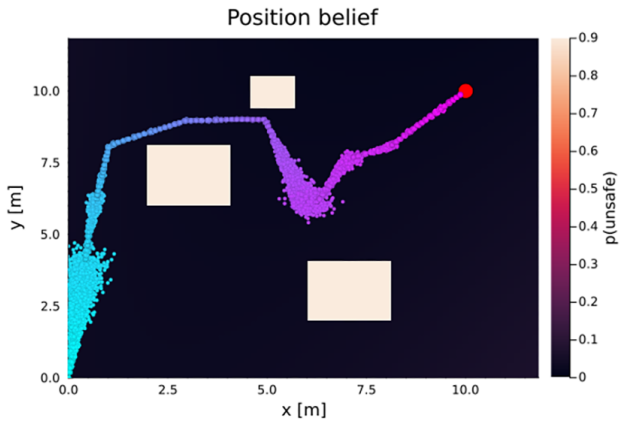


Fig. 4.5 Multi waypoint path move-scan POMDP agent simulation

waypoints $W = w_1, \ldots, w_n$, and enforcing the termination condition on covariance only for the last waypoint. The idea of an active waypoint i is introduced so that the controller policy π^{C_2} becomes a straight-line movement from the current belief state to the current waypoint:

$$\pi^{C_2} = \frac{x^{W_i} - \mathbb{E}(B(s))}{\|x^{W_i} - \mathbb{E}(B(s))\|} u \tag{4.54}$$

. The active waypoint can be incremented by checking whether the belief state's mean is within a circle of radius ρ of that waypoint. For $i < |W|$, the conditions for incrementing i can therefore be written as:

$$\|x^{W_i} - \mathbb{E}(B(s))\| < \rho \quad i < |W| \tag{4.55}$$

The last point in the path has different termination conditions to ensure that the final goal is reached.

$$\begin{cases} \|x^{W_i} - \mathbb{E}(B(s))\| < \rho \\ \mathrm{Tr}(\Sigma) < \varepsilon_\Sigma \end{cases} \qquad i = |W| \tag{4.56}$$

Figure 4.5 shows an example simulation of this agent's decisions over a sequence of waypoints in a map with three obstacles. The agent tends to stop more often when approaching obstacles or when getting close to the end goal.

4.3 Task And Motion Planning

The previous move-scan problem formulation assumes that there is a low-level planner that generates a path. A better performance can be achieved if the planner jointly reasons about path selections and scan scheduling. Depending on how expensive scanning is, the agent could keep farther or closer from obstacles. A joint optimization of tasks and motion falls under the realm of *Task And Motion Planning* (TAMP). Task and motion planning aims to solve a problem that includes elements of continuous space motion planning, discrete space task planning, and discrete-continuous programming [106]. Each field in isolation is not suited to tackle the combined problem, yet a unified formulation allows building agents to operate in unstructured environments. The classic example of a TAMP problem requiring a unified solution is a kitchen robot that needs to cook a dish. The elements of task planning are figuring out the sequence of actions to take (open drawer, take ingredients, process ingredients, cook, etc.) and the trajectories that need to be taken to perform these actions. A critical insight that motivates TAMP as a hybrid discrete-continuous approach is that the world's configuration space is highly modal. So, this structure can be exploited by planning both tasks (what mode to plan in) and performing detailed planning for each mode. The following sections will examine various task and motion planning approaches applied to the EELS move-scan use case.

4.4 POMDP TAMP

The core difference between this formulation and the move-scan activity scheduling formulation presented in section 4.2 lies in the action space choice. The agent's action space is imagined as a mix of discrete activities $a \in A$ and discretized control inputs $u \in U$. Although continuous control inputs could be used in principle, doing so would further decrease the problem's computational tractability.

$$\mathscr{A} = \{U, A\} = \{u1, \ldots, u_{n_u}, a_1, \ldots, a_{n_a}\} \tag{4.57}$$

In the EELS surface mobility case, $A = \{\text{scan}\}$ and U is the versor that determines the control input's direction on a 2D space, discretized in 8 actions.

$$U = \left\{ [0,1], [1,0], [0,-1], [-1,0], \left[\frac{1,1}{\|1,1\|}\right], \left[\frac{-1,-1}{\|-1,-1\|}\right], \left[\frac{1,-1}{\|1,-1\|}\right], \left[\frac{-1,1}{\|-1,1\|}\right] \right\} \tag{4.58}$$

Additionally, the assumption is made that the robot's transition function can be expressed as a linear system with action-dependent transition dynamics.

$$X_{k+1} = {}^{a_k}AX_k + {}^{a_k}BU_k + {}^{a_k}Qw \tag{4.59}$$

A consequence of including a velocity control input in the action space is that the velocity vector $\dot{X}$ can be excluded from the state space. A further assumption is that the system can move at a constant velocity v. Unlike the previous formulation, the assumption that the controller can execute the control policy without error is not made. A constant process noise of $\mathcal{N}(0,\Sigma)$ is applied every time a move action is performed. Assuming perfect execution would eliminate the uncertainty source.

$$POMDP = \langle S, \mathscr{A}, T, O, R, \gamma, \mathscr{Z} \rangle \tag{4.60}$$

$$S = \{X, m(X)\} \in \mathbb{R}^2 \times [0,1] \tag{4.61}$$

$$\mathscr{A} = \{U, \text{scan}\} \quad |A| = 9 \tag{4.62}$$

$$T = p(s_{k+1}|s_k, a_k) = \begin{cases} s_{k+1} = \begin{bmatrix} X_k \\ m(X_k) \end{bmatrix} & a_k = \text{scan} \\ s_{k+1} = \begin{bmatrix} X_k + u_k v \Delta t + \mathscr{N}(0,\Sigma) \\ m(X_k + X_k + u_k v \Delta t + \mathscr{N}(0,\Sigma)) \end{bmatrix} & a_k = u \in U \end{cases} \tag{4.63}$$

$$R = \mathbb{1}_{S^{goal}}(s_k) \tag{4.64}$$

$$O = X, m \in \mathbb{R}^2 \times [0,1] \tag{4.65}$$

$$\mathscr{Z}(o^X|s_{k+1}, a_k) = \begin{cases} X_k & a_k = \text{scan} \\ \varnothing & a_k = u \in U \end{cases} \tag{4.66}$$

$$\mathscr{Z}(o^m|s_{k+1}, a_k) = \begin{cases} m(X_{k+1}) & a_k = \text{scan} \\ \varnothing & a_k = u \in U \end{cases} \tag{4.67}$$

$$\gamma = 0.95 \tag{4.68}$$

$$\Pr(m(X_k) \geq \varepsilon) \geq 1 - \Delta \tag{4.69}$$

Similarly to the previous case, the agent does not have enough information about its state to plan accurately. Therefore, planning will have to occur in *belief*. Firstly, position uncertainty is assumed to be distributed as a Gaussian $X_k \sim \mathscr{N}(\mu^X, \Sigma^X)$. The equivalent MDP over belief space can thus be formulated as:

$$BMDP = \langle B, \mathscr{A}, \tau, R, \gamma \rangle \tag{4.70}$$

$$B(s) = \begin{bmatrix} \mathscr{N}(\mu^X, \Sigma^X) \\ B(m) \end{bmatrix} \tag{4.71}$$

$$\mathscr{A} = \{U, \text{scan}\} \tag{4.72}$$

$$\tau = p(b_{k+1}|b_k, a_k) = \begin{bmatrix} \mu^X, \Sigma^X \\ p(m(X)) \end{bmatrix}_{k+1} = \begin{cases} \begin{bmatrix} \mu^X, 0 \\ \text{Est}(p(m(X))) \end{bmatrix}_k & a_k = \text{scan} \\ \begin{bmatrix} \text{Est}(\mu^X, \Sigma^X | o^X) \\ \text{Est}(p(m(X))|o^m) \end{bmatrix}_k & a_k = u \in U \end{cases} \tag{4.73}$$

$$R = \mathbb{E}(\mathbb{1}_{S^{goal}}(s_k)) \tag{4.74}$$

$$\gamma = 0.95 \tag{4.75}$$

$$\Pr(m_k(X) \geq \varepsilon) \geq 1 - \Delta \tag{4.76}$$

While stability margin can be estimated in the same way as for the non-TAMP POMDP formulation, there are fundamentally different considerations to be made about the belief update. What previously was a Kalman Filter akin to tracking an object on a 2D plane with noisy observations now just consists of tracking open loop motion based solely on a dynamics model and perfect knowledge of the control inputs. Consider a system that follows these dynamics:

$$X_{k+1} = X_k + U_k \mathrm{d}t + \mathscr{N}(0, \Sigma) \tag{4.77}$$

The case of open-loop motion with no observations is now considered. Process noise and control input are assumed to be known and can be used to track belief over the system's position. The Kalman filter's update step can be used to propagate belief over time.

$$\hat{X}_{k+1} = FX_k + GU_k + w_k \tag{4.78}$$

$$\hat{P}_{k+1} = FP_kF^T + Q \tag{4.79}$$

Note that since the system dynamics have zero mean noise, $w = \mathbf{0}$.

$$X = \begin{pmatrix} x \\ y \end{pmatrix} \quad U = \begin{pmatrix} u \\ v \end{pmatrix} \quad Q = \begin{pmatrix} \sigma_n^2 & 0 \\ 0 & \sigma_n^2 \end{pmatrix} \quad F = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad G = \begin{pmatrix} \mathrm{d}t & 0 \\ 0 & \mathrm{d}t \end{pmatrix} \tag{4.80}$$

Therefore, a continuously growing covariance characterizes this filter.

4.4.1 Reward shaping

Central to decision-making algorithms is formulating and shaping a reward function. It has been argued that intelligence itself arises from the process of reward maximization [107]. Furthermore, *all* decision-making algorithms can be framed as cost-minimization or reward-maximization exercises. In Monte Carlo Tree Search, the immediate action with the highest expected reward is chosen after the search phase has exhausted all its computational resources. Hence, shaping reward involves defining the function that maps the state to a reward signal. The first type of reward structure that comes to mind is assigning a positive reward for states that are in the goal set and no reward otherwise. In other words, this structure can be explained as rewarding only the ultimate goal's achievement. Formally, this can be written as an indicator function:

$$R = \mathbb{1}_{S^{\text{goal}}}(s) \tag{4.81}$$

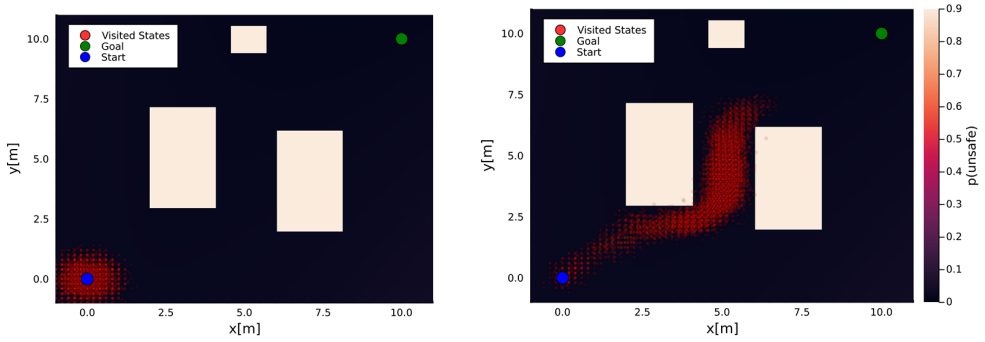
Another way of writing this reward structure is:

$$R_k = \begin{cases} 1 & \text{if:} \quad S_k \in X^{goal} \\ 0 & \text{if:} \quad S_k \notin X^{goal} \end{cases} \tag{4.82}$$

When the reward is so sparsely distributed, the agent cannot distinguish between good and bad actions, as each option seems equally unappealing. This is because the search and rollout phase of MCTS never visits goal states and thus never receives any reward. All action nodes will have an expected reward of 0, and an action will be

selected randomly - as the agent must always choose until the goal has been achieved. Random choice leads to Brownian motion interleaved with environment scans.

There are multiple ways to address the issue of sparse rewards. The most trivial is to increase search depth until search and rollouts visit the goal state. This approach bears the obvious drawback of being computationally infeasible for anything but the most trivial problems and is especially ill-suited for continuous state spaces. As seen in Figure 4.6a, a sparse reward function will lead to states visited through a process of diffusion. Search depth would have to be increased up to the point where it would take days to make a single decision - even for maps that can be solved optimally by 50 actions. A better way of dealing with sparsity is to engineer a reward function that provides continuous feedback to the agent in the form of non-sparse rewards that gradually guide the agent toward the goal. Such a function could be either an explicit reward model or a custom rollout function that maps leaf node belief state directly to a reward without simulating any decision steps. In other words, search depth can be reduced if a continuous reward gradient guides the agent toward its goal.



(a) Sparse reward

(b) Potential field reward

Fig. 4.6 Visualization of the tree search phase using MCTS and different reward functions.

Given that the goal of this task and motion planning agent is to reach a position in space, a simple reward structure could be a global, convex potential field with a maximum centered on the goal. This is similar to the cost-to-go heuristic applied in A*. The implementation details of this function are not so important, as it is sufficient that the reward of state s_1 is greater than the reward of state s_2 if and only if the Euclidean distance between s_1 and the goal is smaller than the distance between s_2 and the goal:

$$R(s_1) > R(s_2) \iff \|\mu_{x_1} - x_{\text{goal}}\| < \|\mu_{x_2} - x_{\text{goal}}\| \tag{4.83}$$

A few examples used throughout this work are 2D Gaussian distributions centered around the goal and evaluated at state s_k, and the inverse of the distance between s_k and the goal.

$$^1R(s_k) = \frac{1}{\sqrt{2\pi}} \exp\left[-\left(\frac{(\mu_{x_k} - x_{\text{goal}})^2}{2} + \frac{(\mu_{y_k} - x_{\text{goal}})^2}{2}\right)\right] \tag{4.84}$$

$$^2R(s_k) = \frac{1}{\|\mu_{\mathbf{x}} - \mathbf{x}_{\text{goal}}\|} \tag{4.85}$$

In Figure 4.6, the custom reward structure's guidance of the search toward the goal is clearly visible. This figure visualizes a single tree search phase by plotting the states visited during this phase. On the left side, sparse rewards lead to the expansion of states without any preferential direction, as MCTS perceives no reward. On the right side, a continuous reward field pushes the agent to consider states gradually closer to the goal.

With this reward structure, the agent will choose actions that move toward the goal, and the chance constraints will make the agent avoid obstacles. Additionally, the chance constraints bias the agent to select information-seeking actions when state uncertainty is high and obstacles are nearby.

Excessively greedy strategies emerge when following Euclidean-distance-based reward structures. The agent will get stuck in locations where the optimal choice is to incur a cost for a number of steps greater than the search horizon. This type of cost-incurring is necessary when getting out of local reward maxima caused by a concave obstacle interposed between the agent and the goal. In these circumstances, the agent expects more reward by staying in place rather than trying to move around the obstacle. The agent is not aware of the full reward field and is not aware that incurring costs for n steps will lead to more rewards overall. Thus, the behavior that maximizes expected reward over the search tree horizon will be to orbit the local maximum. Figure 4.7 shows an example of such a local reward maximum. It is worth noting that increasing search depth will yield better behavior as the agent

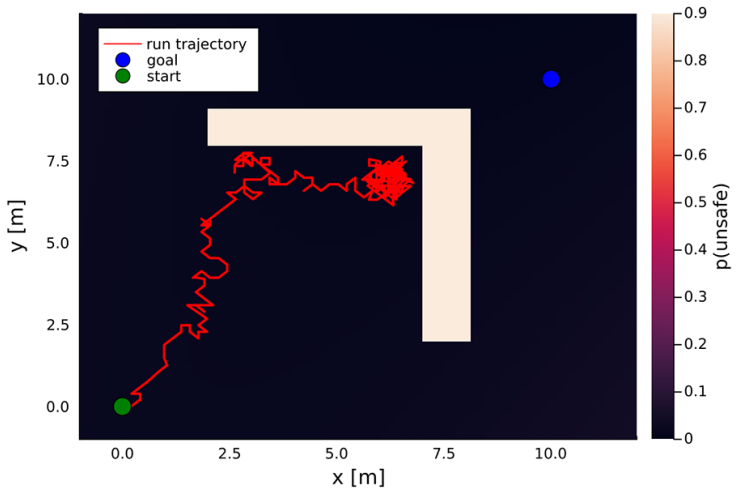


Fig. 4.7 Failure case of reward function shaped as a potential energy field.

might be able to perceive that it is in a local maximum by visiting high reward states outside of the peak. Unfortunately, it is impossible to guarantee that the agent will never get stuck without infinite computational resources.

Local reward maxima can be eliminated by approximating the cost to reach the goal from each point in the state space. To do so, a reward map needs to be computed and then fed to a reward function in MCTS that will query this map.

An easy way of computing a reward map is by leveraging an A* path from start to goal positions. The reward function can thus be expressed as the sum of the distance between the agent and A* path and the distance from the closest point on the path to the goal following the path. Let P be the ordered set of points that make up the A* path from x_0 to x_{goal}. The distance from the agent's state s_k to the path can be written as $x_{\text{sP}} = \min\{\|\mu_x - x\| \mid x \in P\}$. Additionally, the index of the minimum distance point along the path can be written as $i_{\text{sP}} = \{i \mid i \in \{1,\ldots,|P|\},\ P[i] = x_{\text{sP}}\}$. Thus, the reward function can be seen as the sum of two functions:

$$R(s_k) = f(x_{\text{sP}}) + f(|P| - i_{\text{sP}}) \tag{4.86}$$

As in the previous case, the exact details of these two functions are not crucial so long as there are perceivable reward gradients. If $\nabla(f(x_{\text{sP}})) \gg \nabla(f(|P| - i_{\text{sP}}))$, the agent will favour moving toward the path. This becomes relevant in cases where the path passes between two obstacles that can not be passed without violating safety constraints. In these circumstances, if the reward of reaching the path is much higher

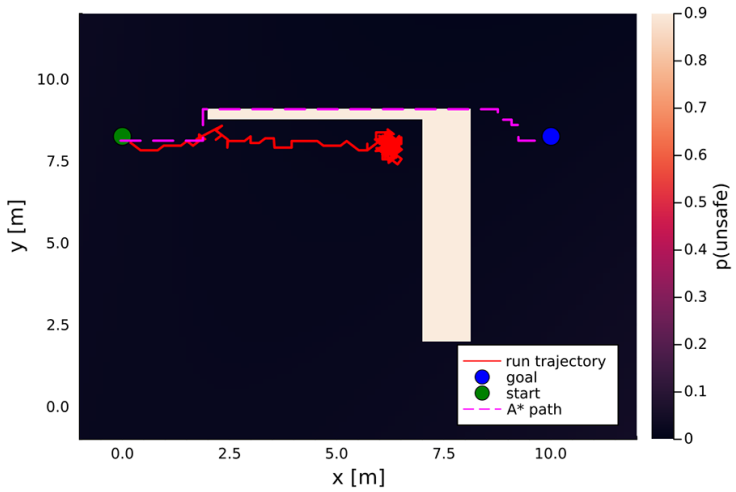


Fig. 4.8 Failure case of using a path generated with A* as part of the reward function.

than the reward of progressing along the path, the agent risks getting stuck behind obstacles. On the other hand, if the path runs parallel to a thin wall structure and the attraction toward the path is not strong enough, then the agent could get stuck on the "wrong side" of the wall, progressing until it gets trapped behind the obstacle. A visual example of this reward strategy failing to find yield desirable behavior can be seen in Figure 4.8. In the image, the agent gets stuck behind a wall and would need to incur costs by backtracking along the path to get back on the correct side of the obstacle. Some of these issues can be mitigated by planning the optimal path over an inflated obstacle set. Additionally, this reward structure's failure modes can be entirely mitigated by recomputing the optimal path every few actions at the cost of additional computational overhead.

Another way to further improve the reward structure and achieve a mixture of long-term path planning mixed with short-term obstacle avoidance and opportunistic reward maximization is to leverage a Probabilistic Roadmap (PRM) [108]. The core idea is that a PRM allows approximating the underlying cost of each state without precomputing a full cost map. First, a set of points is randomly sampled through the state space. Samples falling within obstacles are rejected, and the remaining are connected in a graph structure. Each point is connected only to other points that can be reached through a straight line without intersecting any obstacle. The goal is appended to the graph to ensure accurate cost estimates. For every ith node in the graph, a "cost to go" is computed as the minimum distance path along the graph to reach the goal node. The PRM structure can be seen in Figure 4.9.

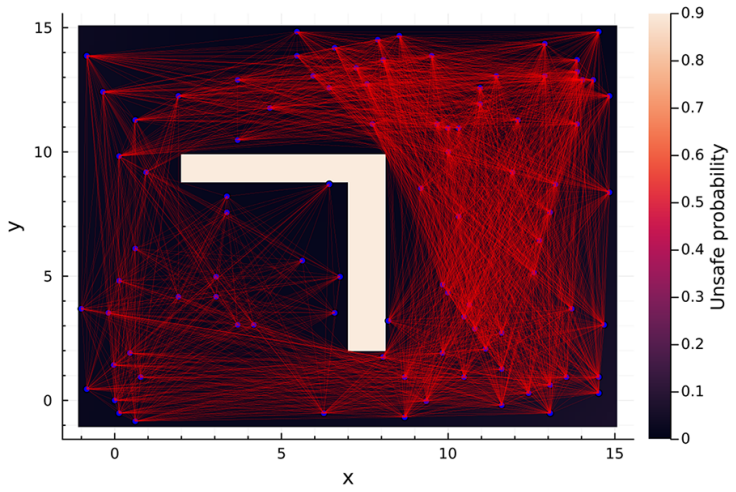


Fig. 4.9 Probabilistic Road Map (PRM) graph for an environment with a L-shaped obstacle

After having defined a "cost to go" metric, the reward function can be written as a function of the distance between the agent and the closest node in the PRM and this cost to go. This reward structure is similar to what has been written for the A* guided reward but is significantly less susceptible to failure modes such as that outlined in Figure 4.8. Let d_g be Euclidean distance from the agent and the closest node in the PRM, d_{PRM}, $c, k \in \mathbb{R}$ two parameters. The reward for state at time k (s_k) can be written as:

$$R(s_k) = ce^{\left[k(d_g + d_{PRM})\right]} \tag{4.87}$$

A further advantage of this reward structure is the ability to embed additional domain knowledge into the PRM edge weights by modifying the edge connection function. Specifically, an approximation of the probability of collision for each connection was included. Each connection is assumed to represent a straight line the agent can follow, with growing state uncertainty. Let X^{obs} be the set of obstacles in the map, $x_0, \ldots, x_n$ be a discretization of the straight line connection between the two nodes in intermediate states. The agent's state uncertainty is normally distributed, with a time-dependent growth. Assuming that each state's probability of collision does not depend on the other states, the total probability of collision for the full edge

r can be written as

$$r = 1 - \prod_{i=0}^{n} \left(1 - \int_{X^{obs}} p(x|x_i, \sigma_i) \mathrm{d}x \right) \tag{4.88}$$

This risk metric can be used to adjust the Euclidean distance of each edge in a way that favors remaining at a distance from obstacles. Note that the integral of the agent's state uncertainty distribution over the obstacle set is not analytically solvable and is thus computed through numerical approximation.

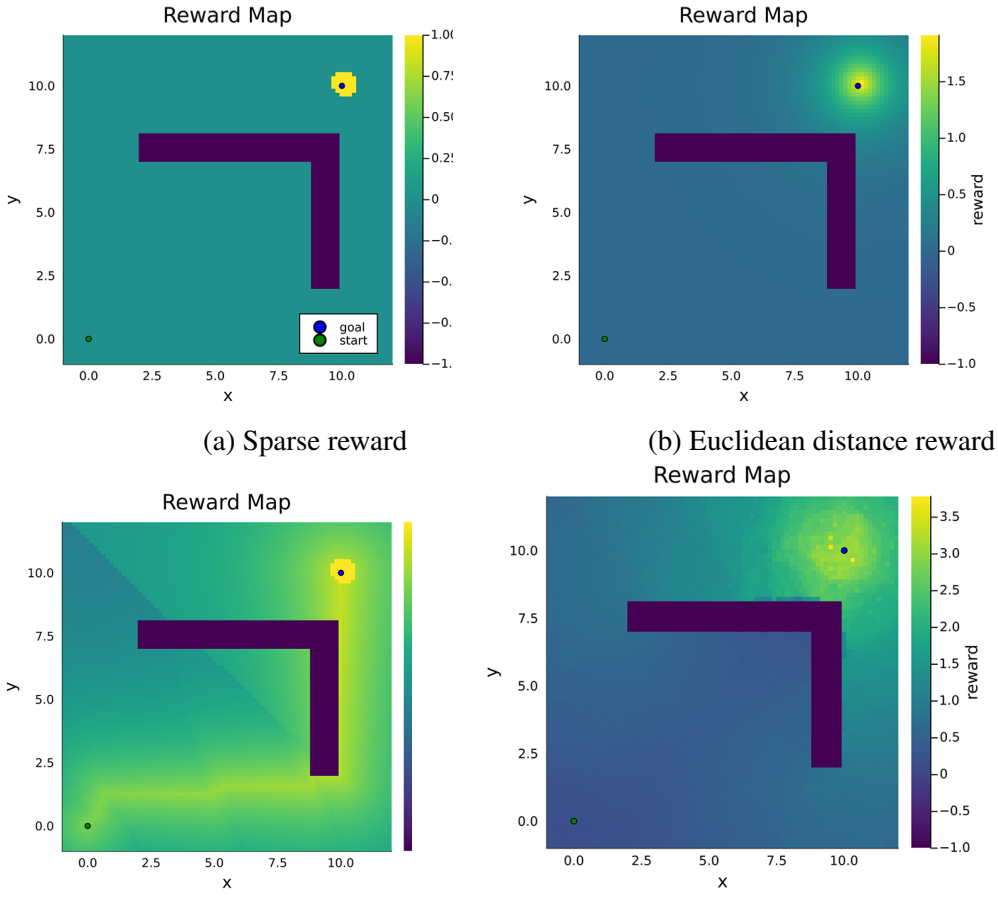
Figure 4.10 shows a comparison between several reward structures plotted over a map. It can be easily seen in this image how the sparse reward function of dispensing a fixed reward only when the agent reaches the goal region and nothing otherwise leads to a complete lack of reward gradient, leading to ineffective decision-making. On the other hand, an euclidean-distance-based reward function creates a gradient but is not aware of obstacles, thus creating a risk of entrapment into local distance minima. The PRM-based reward function (without uncertainty-aware edge weights) shows an awareness of the presence of obstacles. The darker area close to the inner corner of the L-shaped obstacle represents a low reward region and thus indicates that an agent tasked with maximizing expected reward in this reward landscape will avoid the trap.

4.4.2 Negative rewards

Up to now, only positive rewards have been considered, without discussing obstacle avoidance and information-gaining penalization. Negative reward values can be assigned to actions or states to discourage the agent from certain behaviors. In the move-scan problem, three aspects need to be discouraged:

- Colliding with obstacles
- Scanning too frequently
- Taking excessively long paths

Collisions with obstacles have been modeled collisions as terminal states. When an agent hits a terminal state, the branch of decisions that led to that terminal state is wholly discarded. This same behavior can be modeled by penalizing collisions with



(a) Sparse reward

(b) Euclidean distance reward

(c) A* reward

(d) PRM Reward

Fig. 4.10 Visualization of Reward maps

high negative rewards, but the terminal condition approach leads to a more robust avoidance of obstacles. For the move-scan problem, collision states are sufficiently sparse that it is worth modeling them as terminal states. To some degree, movement will be favored over scanning even without negative rewards, as - all being equal - movement will reap higher expected rewards by leading the agent closer to the goal. Regardless, if scanning is not penalized, the agent will not have a strong enough incentive against scanning too frequently. A tunable cost of scanning parameter $c_s \in \mathbb{R}^-$ can be introduced, and the reward function for each action can be written.

$$R_k^a = \begin{cases} 0 & \text{if} \quad a_k = \text{move} \\ -c_s & \text{if} \quad a_k = \text{scan} \end{cases} \tag{4.89}$$

This simple structure is sufficient to discourage the agent from unnecessary scans. Path length does not need to be explicitly minimized, as an agent following a reward gradient toward a goal will favor directions where the reward gradient is maximized. Nevertheless, a simple and effective way to explicitly encourage shorter path lengths is to assign a small negative cost to each movement action $c_m \in \mathbb{R}^+$. The reward function for each action can be written as:

$$R_k^a = \begin{cases} -c_m & \text{if} \quad a_k = \text{move} \\ -c_s & \text{if} \quad a_k = \text{scan} \end{cases} \tag{4.90}$$

Note that the relative magnitude between the cost of moving and the cost of scanning must be tuned to keep discouraging information gaining over movement ($c_m << c_s$).

4.5 Optimization-based approach to TAMP

The previous sections focused on formulating the move scan problem as a Partially Observable Markov Decision Process and converged on using a sampling-based solver to approximate an optimal policy for this problem. Task and Motion Planning under uncertainty can also be formulated as a convex optimization problem and solved using efficient tools developed for the Operations Research and Optimization Community. Before outlining the EELS task and motion planning formulation, it is worth approaching the problem in a step-by-step manner by first introducing how to formulate a motion planning problem as a Linear Program and gradually increasing complexity until the complete move-scan scenario can be formulated as a Mixed Integer Linear Program.

4.5.1 Motion planning as a Linear Program

Motion planning can be formulated as a Linear program. In the interest of clarity, this section will not seek to optimize any metric but will only satisfy obstacle avoidance and state propagation constraints. Let $X_k \in \mathbb{R}^{N_x}$ be a state vector of Cartesian coordinates at time k, and U_k the vector of control inputs. For a 2-dimensional planning problem, these are:

$$X_k = \begin{bmatrix} x \\ y \end{bmatrix} \quad U_k = \begin{bmatrix} u \\ u \end{bmatrix} \tag{4.91}$$

For this type of optimization problem, variables should be bounded by the following constraints:

$$x_{\min} \leq x \leq x_{\max} \quad \forall x \in X \tag{4.92}$$

$$u_{\min} \leq u \leq u_{\max} \quad \forall u \in U \tag{4.93}$$

The state transition constraints can be written in the linear form as:

$$X_{k+1} = AX_k + BU_k + C \tag{4.94}$$

In the 2-D motion planning case, with a fixed time step Δt, these constraints can be written as:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_k + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}_k \tag{4.95}$$

Initial and goal conditions need to be enforced as state constraints at time 0 and N:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{k=0} = \begin{bmatrix} x_{start} \\ y_{start} \end{bmatrix} \tag{4.96}$$

$$\begin{bmatrix} x \\ y \end{bmatrix}_{k=N} = \begin{bmatrix} x_{goal} \\ y_{goal} \end{bmatrix} \tag{4.97}$$

These constrains will ensure that the solver will find a path between two specific positions. Obstacles constraints are trickier to write down as linear constraints. The easiest type of obstacle is a half-plane aligned with one of the state vector's

coordinates. For example, avoiding an obstacle that starts at $x = b$ and goes all the way to $x = \infty$ and extends from $y = -\infty$ to $y = \infty$ can be written as:

$$x_k < b \tag{4.98}$$

An inclined half-plane constraint can be written as:

$$ax + by < c \tag{4.99}$$

For higher-dimensional problems, half-plane constraints can be written as:

$$\sum_{i=0}^{N_x-1} a_i x_i < c \tag{4.100}$$

The sign of the inequality will determine which side of the half-plane is considered as free space. Writing constraints for a convex obstacle can be more complex. The source of complication is that polygonal obstacles are a set of half-plane obstacles connected via an "OR" statement. While "AND" statements are easy to model as linear constraints, implementing an "OR" requires further algebraic tricks. An example is a rectangular obstacle with the bottom left corner in (a_1, a_3) and the top right corner in (a_2, a_4):

$$a_1 < x \vee x > a_2 \tag{4.101}$$
$$a_3 < y \vee y > a_4 \tag{4.102}$$

Typically, similar constraints can be rewritten using some form of trick that increases the total number of constraints. In the rectangular obstacle case, let M be a number much greater than the bounds of the state vector $M \gg \|X\|_{max}$. Let b be binary integer variables $b \in \{0,1\}$ [109].

$$x_k \leq a_1 + Mb_k^1 \tag{4.103}$$
$$-x_k \leq -a_2 + Mb_k^2 \tag{4.104}$$
$$y_k \leq a_3 + Mb_k^3 \tag{4.105}$$
$$-y_k \leq a_4 + Mb_k^4 \tag{4.106}$$
$$\sum_{i=1}^{4} b_k^i \leq 3 \tag{4.107}$$

Adding a large value to the inequality constraints is a way to relax them, and requiring the binary selector variables to sum up to three is a way to ensure that at least one of the original constraints is always valid. A 1-D example of why constraint relaxation works is:

$$x < a_1 \vee x > a_2 \tag{4.108}$$

When $x > a_2$, $x < a_1$ is not satisfied.

$$x \leq a_1 + Mb_1 \tag{4.109}$$
$$x \geq a_2 - Mb_2 \tag{4.110}$$
$$b_1 + b_2 \leq 1 \tag{4.111}$$

If $x > a_2$ (agent to the right of the obstacle), then the system of constraints above would have $b_2 = 0, b_1 = 1$.

$$x \leq a_1 + M \tag{4.112}$$
$$x \geq a_2 \tag{4.113}$$
$$\tag{4.114}$$

Both constraints are satisfied as $M \gg |x_{\max}|$. On the other hand, no valid assignment of x, b_1, b_2 could be found for $a_1 < x < a_2$, thus enforcing the obstacle constraint. Figure 4.11 shows a solution to a path planning problem formulated as a constraint satisfaction linear program. It is interesting to note that the solver looked for a solution starting from the obstacle constraint and found a valid assignment that is very close to the shortest path. Additionally, the path cuts through the obstacle's top-left corner. This is a consequence of the formulation, as there are no collision checking constraints between X_k and X_{k+1}. The solver is not checking if the connection between the two states is valid, rather it is checking if the two states are *valid*. Therefore, if the two states are outside of the obstacle and the distance between them is smaller than $u_{\max}\Delta t$, then the assignment is valid. A sufficiently large Δt or large values for $|u|_{max}$ will lead to the solver believing that the agent can move through the entire obstacle.

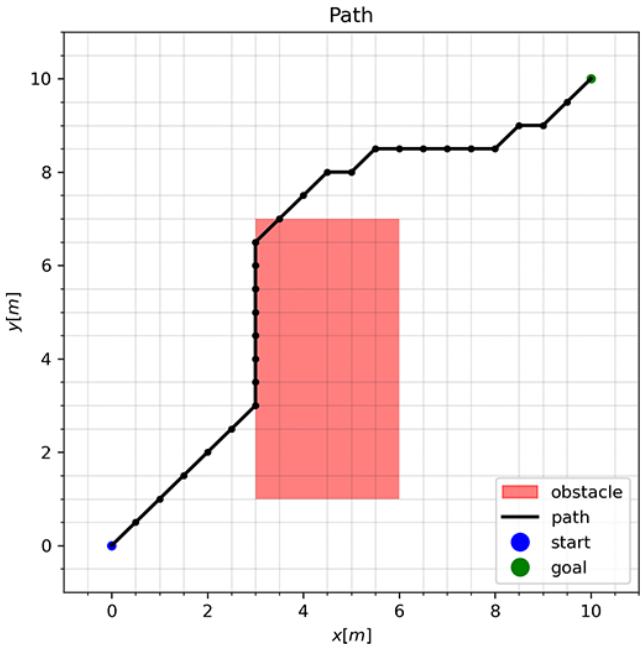


Fig. 4.11 Path planning Constraint Satisfaction linear program with rectangular obstacle

4.5.2 Optimizing for minimum path length

There are several ways to add an optimization objective to the path planning formulation. If interested in path-length minimization, the assumption could be that the agent incurs a cost proportional to the distance traveled. This can be written as a minimization of some constant c times the norm of the control vector for each time step:

$$\min_{U_1,\dots U_N, X_1 \dots X_N} \sum_{k=1}^{N} \sum_{i=1}^{N_u} c|u_k^i| \tag{4.115}$$

From a notation perspective, u_k^i means the $i-th$ component of the control vector at step k ($u_k^i \in U_k$). The absolute value is not a linear function and requires more algebraic tricks to be expressed as a linear constraint. The general idea is that $|u_i|$ needs to be substituted with a linear variable z, and add a few constraints that force z_k to behave like the absolute value of u need to be added:

$$\begin{cases} u_k & \leq z_k \\ -u_k & \leq z_k \end{cases} \tag{4.116}$$

The problem can thus be reformulated as:

$$\min_{U_1,\dots U_N, X_1 \dots X_N} \sum_{k=1}^{N} \sum_{i=1}^{N_u} c z_k^i \tag{4.117}$$

$$u_k^i \leq z_k^i \tag{4.118}$$

$$-u_k^i \leq z_k^i \tag{4.119}$$

In summary, a simple form of a motion planning linear program with a path length minimization objective can be expressed as:

$$\min_{U_1,\dots U_N, X_1 \dots X_N} \sum_{k=1}^{N} \sum_{i=1}^{N_u} c z_k^i \tag{4.120}$$

$$u_k^i \leq z_k^i \tag{4.121}$$

$$-u_k^i \leq z_k^i \tag{4.122}$$

$$X_{k+1} = AX_k + BU_k + C \tag{4.123}$$

$$X_0 = X_{\text{start}} \tag{4.124}$$

$$X_N = X_{\text{goal}} \tag{4.125}$$

$$\sum_{i=0}^{N_x} {}^o a_i^j \, {}^o x_i^j < {}^o c^j + M \, {}^o b_k^j \quad j \in \{1,\dots,N^{\text{obs}}\} \quad o \in \{1,\dots,N_{\text{faces}}^j\} \tag{4.126}$$

$$\sum_{i=1}^{N_{\text{faces}}^j} {}^o b_k^{i,j} \leq N_{\text{faces}}^j - 1 \tag{4.127}$$

4.5.3 Single-action deterministic TAMP

The motion planning formulation can be used as a starting point to move toward integrated task and motion planning (TAMP). For simplicity, optimization objectives will be dropped in this section. Furthermore, a trivial version of TAMP will be introduced first, where the agent has a single action to choose from. Let X_k and U_k be the state and control vectors at time step $k \in \{1,\dots,N\}$. The time elapsed since the first step t can be tracked by augmenting the state vector. This state-vector augmentation has no influence on single-action deterministic TAMP, but tracking time and the squared of time t^2 is necessary for multi-action risk-aware TAMP.

$$X_k = \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix} \quad U_k = \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix} \tag{4.128}$$

The linear state transition constraints can be written as:

$$X_{k+1} = AX_k + BU_k + C \quad k \in \{0,\dots,N-1\} \tag{4.129}$$

In our case, these transition constraints are:

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_k + \begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}_k + \begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix} \tag{4.130}$$

It is interesting to expand on the transition constraints for time and time squared have been derived. The time step Δt is assumed to be constant, therefore making t_{k+1} a trivial constraint to derive. t_{k+1}^2 on the other hand is derived by substituting t_{k+1}.

$$t_{k+1} = t_k + \Delta t \tag{4.131}$$

$$t_{k+1}^2 = (t_k + \Delta t)^2 = t_k^2 + 2(\Delta t)t_k + \Delta t^2 \tag{4.132}$$

Initial and terminal conditions can be written as:

$$X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_N = x_{goal} \quad y_N = y_{goal} \tag{4.133}$$

This formulation can be extended to a trivial 1-action TAMP by including a binary selector variable $s_k \in \{0,1\}$.

$$X_{k+1} = s_k(AX_k + BU_k + C) \tag{4.134}$$

This expression will contain nonlinear product terms of s_k. Specifically, these products are $(sx, sy, st^2, st, su, sv)$. These nonlinear terms can be linearized by introducing the slack variables $r_k^x = x_k y_k$.

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r^x \\ r^y \\ r^{t^2} \\ r^t \end{bmatrix}_k + \begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} r^u \\ r^v \\ 0 \\ 0 \end{bmatrix}_k + \begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix} s_k \tag{4.135}$$

$$\sum_{i=0}^{N_a} s_k^i = 1 \quad X_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_N = x_{goal} \quad y_N = y_{goal} \tag{4.136}$$

(4.137)

Each slack variable is valid only if accompanied by additional constraints:

$$\forall r_k^x : \quad \begin{cases} r_k^x \geq x_{min} s_k \\ r_k^x \leq x_{max} s_k \\ r_k^x \leq x_k - x_{min}(1 - s_k) \\ r_k^x \geq x_k - x_{max}(1 - s_k) \end{cases} \tag{4.138}$$

Since this is a trivial case of TAMP, where $N_a = 1$, the constraint over the sum of selector variables at each time step becomes $s_k = 1$. Therefore, for all time steps the slack variable constraints become:

$$\forall r_k^x : \quad \begin{cases} x_{min} \leq r_k^x \leq x_{max} \\ r_k^x = x_k \end{cases} \tag{4.139}$$

4.5.4 Multi action deterministic TAMP

The deterministic TAMP formulation can be extended to include a second action, "Scan". Scanning is assumed to have two effects: (1) the agent's pose does not change, and (2) the time elapsed since the first movement step is reset. Some clarifications on notation: ${}^{a^i}A$ means matrix A for action a^i. ${}^{a^i}r_k^{t^2}$ means slack

variable r for optimization variable t^2, at time k, for action a^i. In general, the transition constraints are:

$$X_{k+1} = {}^{a^k}AX_k + {}^{a^k}BU_k + {}^{a^k}C \quad k \in \{0,\dots,N-1\} \tag{4.140}$$

Let M be shorthand for "Move" and S shorthand for "Scan". The agent's action space $\mathscr{A}$ can be expressed as:

$$a^i \in \mathscr{A} = \{M,S\} \tag{4.141}$$

The two transition systems for move and scan are thus:

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{{}^{M}A} \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_k + \underbrace{\begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{{}^{M}B} \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}_k + \underbrace{\begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix}}_{{}^{M}C} \tag{4.142}$$

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{{}^{S}A} \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_k + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{{}^{S}B} \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}_k + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{{}^{S}C} \tag{4.143}$$

Note that t and t^2 do not represent absolute time, but the time passed since the *Scan* action. The selector variable ${}^{a^i}s_k \in \{0,1\}$ can be introduced as before.

$$X_{k+1} = {}^{M}s_k\left({}^{M}AX_k + {}^{M}BU_k + {}^{M}C\right) + {}^{S}s_k\left({}^{S}AX_k + {}^{S}BU_k + {}^{S}C\right) \quad k \in \{0,\dots,N-1\} \tag{4.144}$$

In more general terms, if there are N_a different actions ($|\mathscr{A}| = N_a$), the transition constraints can be written as:

$$X_{k+1} = \sum_{a^i=1}^{N_a} {}^{a^i}s_k\left({}^{a^i}AX_k + {}^{a^i}BU_k + {}^{a^i}C\right) \quad k \in \{0,\dots,N-1\} \tag{4.145}$$

For the move-scan case, the transition constraints can be explicitly written as:

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = {}^{M}s_k \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_k + \begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}_k + \begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix} \right) + {}^{S}s_k \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_k + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}_k + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) \quad (4.146)$$

As was the case for a one-action TAMP, nonlinear terms ${}^{a^i}y_k x_k$ have appeared in the system:

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{M}sx \\ {}^{M}sy \\ {}^{M}st^2 \\ {}^{M}st \end{bmatrix}_k + \begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{M}su \\ {}^{M}sv \\ 0 \\ 0 \end{bmatrix}_k + {}^{M}s_k \begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{S}sx \\ {}^{S}sy \\ {}^{S}st^2 \\ {}^{S}st \end{bmatrix}_k + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{S}su \\ {}^{S}sv \\ 0 \\ 0 \end{bmatrix}_k + {}^{S}s_k \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.147)$$

These nonlinearities can be removed by introducing slack variables ${}^{a^i}r_k^x = {}^{a^i}y_k x_k$.

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{M}r^x \\ {}^{M}r^y \\ {}^{M}r^{t^2} \\ {}^{M}r^t \end{bmatrix}_k + \begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{M}r^u \\ {}^{M}r^v \\ 0 \\ 0 \end{bmatrix}_k + {}^{M}s_k \begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{S}r^x \\ {}^{S}r^y \\ {}^{S}r^{t^2} \\ {}^{S}r^t \end{bmatrix}_k + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} {}^{S}r^u \\ {}^{S}r^v \\ 0 \\ 0 \end{bmatrix}_k + {}^{S}s_k \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.148)$$

The system can be written compactly as:

$$X_{k+1} = {}^{M}A\,{}^{M}R_k^X + {}^{M}B\,{}^{M}R_k^U + {}^{M}s_kC + {}^{S}A\,{}^{S}R_k^X + {}^{S}B\,{}^{S}R_k^U + {}^{S}s_kC \tag{4.149}$$

Or, in the general case:

$$X_{k+1} = \sum_{a^i=0}^{N_a} \left({}^{a^i}A\,{}^{a^i}R_k^X + {}^{a^i}B\,{}^{a^i}R_k^U + {}^{a^i}s_kC \right) \tag{4.150}$$

Each slack variable is valid only if accompanied by four constraints already seen in Equation 4.138. A problem with N steps, N_a actions, N_x state variables, and N_u control variables will have slack variable constraints in number:

$$N_{Cslack} = 4NN_a(N_x + N_u) \tag{4.151}$$

Constraints on initial state, terminal state, and on the sum of ${}^{a^i}s_k$ need to be added as in Equation 4.136. These are going to be $N + 2N_x$ additional constraints. The state transition model has N_xN constraints. This approach can be used to model an arbitrary number of actions, each applying a different linear transition dynamics model to an agent. It is also clear that the number of constraints grows linearly with the number of actions, and consequently, solution time will drastically increase as more actions are introduced.

4.5.5 Adding state uncertainty and chance constraints

Action-dependant noise ${}^{a_k}\omega_k$ can be added to the state transition function as:

$$X_{k+1} = {}^{a_k}AX_k + {}^{a_k}Bu_k + {}^{a_k}\omega_k + {}^{a_k}C \tag{4.152}$$

Assuming that the agent's state uncertainty is Gaussian $B_X \sim \mathcal{N}(\mu_X, \Sigma_x)$, that the noise ${}^{a_k}\omega_k$ has zero mean $\mu_\omega = 0$ and constant covariance ${}^{a_k}\Sigma_{\omega_k}$, state transition constraints can be re-written as:

$$\mu_{X_{k+1}} = {}^{a_k}\mu_{X_k} + {}^{a_k}BU_k + {}^{a_k}C \tag{4.153}$$

$$\Sigma_{X_{k+1}} = {}^{a_k}A\Sigma_{X_k}\,{}^{a_k}A^T + {}^{a_k}\Sigma_{\omega_k} \tag{4.154}$$

The expression for state mean propagation is already written in the same form as Equation 4.140. This form can be transformed into a mixed integer linear program by following the same steps outlined in the previous section. Selector variables ${}^{a_i}s_k \in \{0,1\}$ need to be added, and slack variables ${}^{a^i}r_k^x$ can be added. Following this process will yield a mean propagation expression in the form of Equation 4.150:

$$\mu_{X_{k+1}} = \sum_{a^i=0}^{N_a} \left({}^{a^i}A \, {}^{a^i}R_k^{\mu_X} + {}^{a^i}B \, {}^{a^i}R_k^{\mu_U} + {}^{a^i}s_k C \right) \tag{4.155}$$

Covariance propagation is trickier to express with linear constraints as it is action-dependent. Hence, it is unknown a priori. It is important to note that covariance growth is action-dependant but does *not* depend on control input. This means that the robot accrues state uncertainty based on time only when moving. This case is analogous to what is described in [110], where covariance at each time step can be pre-computed at each time step before the optimization is run. In our case, covariance can not be precomputed, but the dependence on *movement time* can be exploited. The exact covariance growth model is non-linear and cannot be expressed as a linear constraint, but a conservative quadratic covariance growth bound can be assumed. Since t^2 is an optimization variable, the covariance growth model can be bounded by a linear expression as:

$$\Sigma_{X_k} = \Sigma_0 t_k^2 \tag{4.156}$$

The covariance propagation augments the problem with $N-1$ constraints. In addition, an initial condition needs to be provided.

$$\Sigma_{X_0} = \Sigma_0 \tag{4.157}$$

As an initial chance constraint test, half-plane obstacles can once again be used. A half-plane chance constraint in its simplest form states that the probability of the system state being beyond coordinate b should be less than a risk tolerance Δ

$$\Pr(X_k \geq b) < \Delta \tag{4.158}$$

This univariate constraint can be expanded by assuming that X_k is a normal distribution $X \sim \mathcal{N}(\mu_X, \Sigma_X)$. Furthermore, since the constraint is univariate, only one of the components of X can be considered. x can be re-written in terms of the standard normal distribution $\mathcal{N}(0,1)$

$$x = \mu_x + \Sigma_x z \tag{4.159}$$

It can be noted that $\phi(x) = \Pr(x \leq b)$. Thus, the constraint can be re-written as:

$$1 - \Pr(x \leq b) < \Delta \tag{4.160}$$

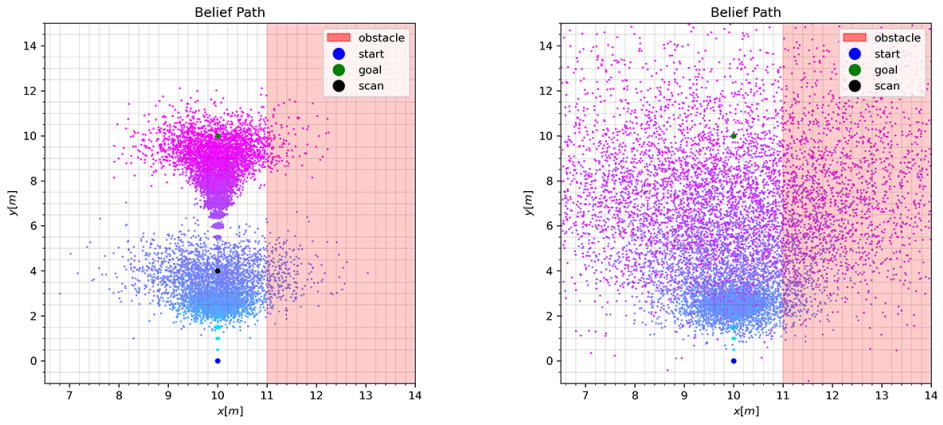
$$1 - \Pr\left(\mu_x + \Sigma_x z \leq b\right) < \Delta \tag{4.161}$$

$$1 - \Pr\left(z \leq \frac{b - \mu_x}{\Sigma_x}\right) < \Delta \tag{4.162}$$

$$1 - \phi\left(\frac{b - \mu_x}{\Sigma_x}\right) < \Delta \tag{4.163}$$

A half-plane obstacle chance constraint can therefore be written as linear in μ_x and Σ_x as:

$$\mu_x + \Sigma_x \phi^{-1}(1 - \Delta) \leq b \tag{4.164}$$



(a) Low tolerance (b) High tolerance

Fig. 4.12 Constraint satisfaction for risk-aware task and motion planning MILP formulation with a single half-planar obstacle. The two images show the effect of varying the risk-tolerance parameter in the generated plan.

Figure 4.12 shows how different risk postures affect the number of scan tasks. The image shows samples taken from the agent's state distribution over time. Hence, a wider sample distribution represents higher state uncertainty, while tighter groupings represent lower uncertainty. Colors indicate time progression. Note that risk tolerance corresponds to the value of Δ in Equation 4.164. For example, lower Δ means that the probability of $X_k \in X_k^{unsafe}$ for every time step k is allowed to be higher. Each half-plane chance constraint adds N constraints to the optimization problem.

4.5.6 Polygonal chance constraints

The half-plane chance constraint formulation can be extended to hyperplanes.

$$\Pr(a_1 x_1 + a_2 x_2 + \cdots + a_{N_x} x_{N_x} \leq b) \leq \Delta \tag{4.165}$$

In a 2D case, this formulation allows describing an inclined plane obstacle. A union of $N_e > 3$ of these constraints represents a polygonal obstacle with N_e sides.

$$\bigcup_{i=1}^{N_e} \Pr(a_1^i x_1^i + a_2^i x_2^i + \cdots + a_{N_x}^i x_{N_x}^i \leq b^i) \leq \Delta \tag{4.166}$$

The focus can now be shifted to rewriting $\Pr(a_1 x_1, \ldots, a_{N_x} x_{N_x})$, which in matrix form can be expressed as:

$$\Pr(HX \leq b) \tag{4.167}$$

State uncertainty is assumed to be normally distributed $X \sim \mathcal{N}(\overline{X}, \Sigma_X)$, where $\overline{X} = [\overline{x}_1, \ldots, \overline{x}_{N_x}]$ is the mean, and Σ_X is the covariance matrix. Substitution yields:

$$y = a_1 x_1, \ldots, a_{N_x} x_{N_x} \tag{4.168}$$

Note that also y follows a gaussian distribution

$$\overline{y} = H\overline{X} = a_1\overline{x} + \cdots + a_{N_x} + \overline{x}_{N-x} \tag{4.169}$$

$$\Sigma_y = H^T \Sigma_X H = \begin{bmatrix} a_1 & \dots & a_{N_x} \end{bmatrix} \begin{bmatrix} \sigma_{11} & \dots & \sigma_{1N_x} \\ \vdots & \ddots & \vdots \\ \sigma_{N_x 1} & \dots & \sigma_{N_x N_x} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{N_x} \end{bmatrix} \tag{4.170}$$

The probability of being in an unsafe region can now be written in terms of y, and use the same trick used for the half-plane constraint of writing y in terms of a standard normal distribution z:

$$\begin{cases} \Pr(y \leq b) \\ y = \overline{y} + \Sigma_y z \\ z \sim \mathcal{N}(0,1) \end{cases} \tag{4.171}$$

$$\Pr\left(z \leq \frac{b-\overline{y}}{\Sigma_y}\right) = \phi\left(\frac{b-\overline{y}}{\Sigma_y}\right) \tag{4.172}$$

The chance constraint can thus be rewritten as:

$$\Pr(HX \leq b) \leq \Delta \tag{4.173}$$

$$\overline{y} + \phi^{-1}(\Delta)\Sigma_y \geq b \tag{4.174}$$

$$\tag{4.175}$$

The definition of $\overline{y}$ and Σ_y can be substituted, yielding a linear expression:

$$H\overline{X} + \phi^{-1}(\Delta) H^T \Sigma_x H \geq b \tag{4.176}$$

Obstacle constraints can be written a disjunctive linear program. Note that at least one of the N_e constraints needs to be satisfied:

$$\bigcup_{i=1}^{N_e} \left[H_i \overline{X} + \phi^{-1}(\Delta) H_i^T \Sigma_x H_i \geq b_i \right] \tag{4.177}$$

A trick from the mathematical programming literature can be used to rewrite this disjunctive program as a set of linear constraints [111]. Specifically, the Big-M trick can be used previously to write rectangle obstacles. Let $M \in \mathbb{R}$ be a number much

larger than the obstacle's bounds, and $y_i \in \{0,1\}$ be a binary integer variable:

$$\bigcap_{i=1}^{N_e} \left[H_i \overline{X} + \phi^{-1}(\Delta) H_i^T \Sigma_x H_i \geq b_i - M(1-y_i) \right] \tag{4.178}$$

$$\sum_{i=1}^{N_e} y_i = N_e - 1 \tag{4.179}$$

A simple example of a 2D triangular obstacle can seen in Figure 4.13. This case study can be used to better understand the math behind chance constraints. The obstacle is defined as the area between by three lines, where the $i-th$ line can be written in the form ${}^i a_1 x_1 + {}^i a_2 x_2 = {}^i b$:

$$-1.5x_1 + 1.0x_2 = -5 \tag{4.180}$$

$$10x_1 + 1.0x_2 = 30 \tag{4.181}$$

$$1.0x_1 + 1.0x_2 = 10 \tag{4.182}$$

The three chance constraints can be written in form:

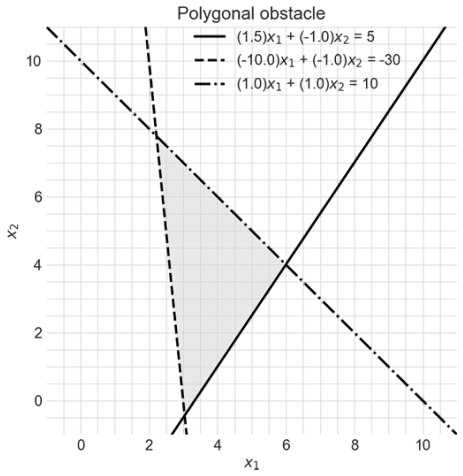


Fig. 4.13 Example of two-dimensional polygonal obstacle defined by three lines

$$\Pr(a_1 x_1 + a_2 x_2 \leq b) \leq \Delta \tag{4.183}$$

In our case, the mean and covariance of the state vector X, and obstacle coefficients H will be:

$$\overline{X} = \begin{bmatrix} \mu_{x_1} \\ \mu_{x_2} \end{bmatrix} \quad \Sigma_X = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \quad H = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{4.184}$$

A single chance constraint can be therefore written as:

$$a_1\mu_{x_1} + a_2\mu_{x_2} + \phi^{-1}(\Delta)\left[a_1^2\Sigma_{11} + a_1a_2\Sigma_{12} + a_1a_2\Sigma_{21} + a_2^2\Sigma_{22}\right] \geq b \tag{4.185}$$

Binary selection variables are added, and the the obstacle constraints are re-written as:

$${}^1a_1\mu_{x_1} + {}^1a_2\mu_{x_2} + \phi^{-1}(\Delta)\left[{}^1a_1^2\Sigma_{11} + {}^1a_1a_2\Sigma_{12} + {}^1a_1\,{}^1a_2\Sigma_{21} + {}^1a_2^2\Sigma_{22}\right] \geq {}^1b - M(1-y_1) \tag{4.186}$$

$${}^2a_1\mu_{x_1} + {}^2a_2\mu_{x_2} + \phi^{-1}(\Delta)\left[{}^2a_1^2\Sigma_{11} + {}^2a_1a_2\Sigma_{12} + {}^2a_1\,{}^2a_2\Sigma_{21} + {}^2a_2^2\Sigma_{22}\right] \geq {}^2b - M(1-y_2) \tag{4.187}$$

$${}^3a_1\mu_{x_1} + {}^3a_2\mu_{x_2} + \phi^{-1}(\Delta)\left[{}^3a_1^2\Sigma_{11} + {}^3a_1a_2\Sigma_{12} + {}^3a_1\,{}^3a_2\Sigma_{21} + {}^3a_2^2\Sigma_{22}\right] \geq {}^3b - M(1-y_3) \tag{4.188}$$

$$y_1 + y_2 + y_3 \geq 2 \tag{4.189}$$

The half plane chance constraint can be written in a way that is readable by a solver as:

$$a_1\mu_{x_1} + a_2\mu_{x_2} + \phi^{-1}(\Delta)a_1^2\Sigma_{11} + \phi^{-1}(\Delta)a_1a_2\Sigma_{12} + \phi^{-1}(\Delta)a_1^2\Sigma_{21} + \phi^{-1}(\Delta)a_2^2\Sigma_{22} - My \geq b - M \tag{4.190}$$

Under the assumption that $\Sigma_{12} = \Sigma_{22} = 0$ and $\Sigma_{11} = \Sigma_{22} = \Sigma_x$, the chance constraints for the obstacle in Figure 4.13 become:

$$1.5\mu_{x_1} - \mu_{x_2} + 6.5\phi^{-1}(\Delta)\Sigma_x - My_1 \geq -5 - M \tag{4.191}$$

$$-10\mu_{x_1} + \mu_{x_2} - -202\phi^{-1}(\Delta)\Sigma_x - My_2 \geq -30 - M \tag{4.192}$$

$$\mu_{x_1} + \mu_{x_2} + 4\phi^{-1}(\Delta)\Sigma_x - My_3 \geq 10 - M \tag{4.193}$$

$$y_1 + y_2 + y_3 \geq 1 \tag{4.194}$$

The constraints can be verified by checking that states outside of the obstacle have a valid assignment. Assuming that the agent's state is $\mu_x = \mu_y = 0$, $\Sigma_x = 0.1$, the goal is to find a valid assignment of y_i such that the constraint is respected for $\Delta = 0.1$ (probability of collision less than 10%)? Note that the inverse CDF function $\phi^{-1}(0.1) \approx -1.3$

$$-0.8 - My_1 \geq 5 - M \tag{4.195}$$
$$-25.8 - My_2 \geq -30 - M \tag{4.196}$$
$$-0.5 - My_3 \geq 10 - M \tag{4.197}$$
$$y_1 + y_2 + y_3 \geq 1 \tag{4.198}$$

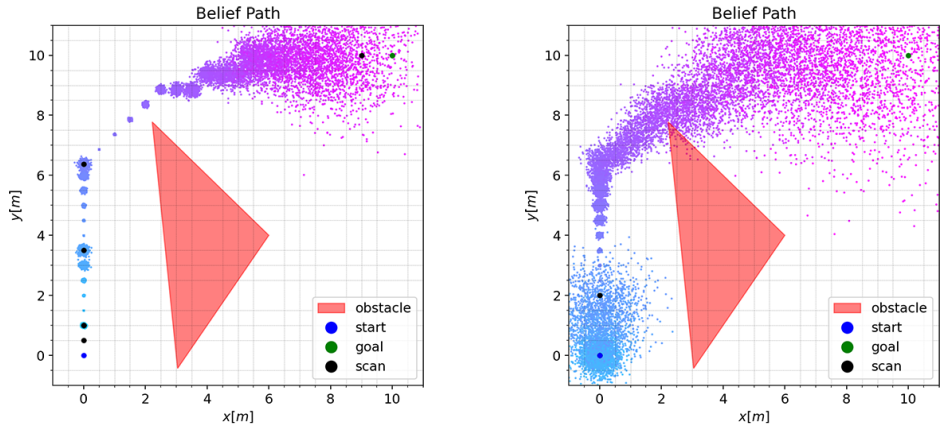
It is east to verify that when $y_1 = 0, y_2 = 1, y_3 = 0$, the constraints are satisfied. On the other hand, the constraints can be verified to show that a state within the obstacle cannot satisfy them $\mu_{x_1} = \mu_{x_2} = 4$, $\Sigma_x = 0.1$.

$$1.2 - My_1 \geq 5 - M \tag{4.199}$$
$$-69.8 - My_2 \geq -30 - M \tag{4.200}$$
$$7.5 - My_3 \geq 10 - M \tag{4.201}$$
$$y_1 + y_2 + y_3 \geq 1 \tag{4.202}$$

Note that all of the constraints are valid only if relaxed ($y_i = 0$). Since there is no valid non-relaxed constraint, there is no valid assignment of y_i such that the obstacle constraints are respected. It can also be verified that a tight covariance bound, for a state with high covariance close to the obstacle, will not have any valid assignment. For this case, $\mu_{x_1} = \mu_{x_2} = 2$, $\Sigma_x = 3$, $\Delta = 0.01$ are used.

$$-44.4 - My_1 \geq 5 - M \tag{4.203}$$
$$-1431.8 - My_2 \geq -30 - M \tag{4.204}$$
$$-23.9 - My_3 \geq 10 - M \tag{4.205}$$
$$y_1 + y_2 + y_3 \geq 1 \tag{4.206}$$

As expected, even though the state's mean is outside the obstacle polygon, the covariance term makes the constraints unsatisfiable. Figure 4.14 shows how valid assignments to the move-scan task and motion planning MILP exist in the presence



(a) Low tolerance (b) High tolerance

Fig. 4.14 Effect of risk tolerance variation for the Constraint Satisfaction of the move-scan MILP with polygonal obstacle chance constraints

of this polygonal obstacle chance constraint. Furthermore, it is shown how a different value of Δ can lead to drastically different behavior. An artifact of the way the chance constraints were formulated is that the obstacle constraints can be satisfied by an agent with μ_x, μ_y inside the obstacle region, but large Σ_x and $\Delta > 0.5$, as that is the threshold for which $\phi^{-1}(\Delta) > 0$. This can be shown effectively by testing if there is a valid assignment for $\mu_x = \mu_y = 4$, $\Sigma_x = 100$, and $\Delta = 0.51$. The result is:

$$18.3 - My_1 \geq 5 - M \tag{4.207}$$

$$462.4 - My_2 \geq -30 - M \tag{4.208}$$

$$18.0 - My_3 \geq 10 - M \tag{4.209}$$

$$y_1 + y_2 + y_3 \geq 1 \tag{4.210}$$

These constraints are valid for multiple assignments of y_1, y_2, y_3. If there is an intention to stop the agent's mean from passing inside obstacle regions, the chance constraint needs to have $\Delta < 0.5$.

4.5.7 Choosing not to act

The formulation forces the solver to look for a solution with *exactly N* steps. What can be seen in Figures 4.14 and 4.15 is that there are time steps when the solution

has reached the goal, but some action needs to be chosen. If *move* is chosen - which tends to be the case if ${}^{M}c < {}^{S}c$ - the moving time t will increment even if the control input is null $\|U\| = 0$. When the moving time t increases, covariance will increase as well. Interestingly, covariance growth without movement is equivalent to IMU integration after stopping. A more efficient way of dealing with pose estimates is to stop integrating IMU signals when the system knows it is not moving. This can be modeled as an action that does not modify the state vector. This type of action is often called a *NoOP action* and is defined with the $\emptyset$ symbol. This action will maintain the previous state unchanged, without propagating the move time as illustrated in Equation 4.211.

$$\begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\emptyset_A} \begin{bmatrix} x \\ y \\ t^2 \\ t \end{bmatrix}_{k} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\emptyset_B} \begin{bmatrix} u \\ v \\ 0 \\ 0 \end{bmatrix}_{k} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\emptyset_C} \tag{4.211}$$

As shown in Figure 4.16, adding the NoOP action improves the solution's quality. Note that NoOP actions add decision variables to the problem, thus greatly impacting the solver's performance.

Another way to inhibit covariance growth is by substituting the NoOP action with a constraint system that stops covariance from growing when the agent is close to the goal. This constraint can be expressed as:

$$\Sigma_{k+1} = \begin{cases} \Sigma_0 t_k^2 & if\ |x - x_{\text{goal}}| + |y - y_{\text{goal}}| \geq \varepsilon \\ \Sigma_k & if\ |x - x_{\text{goal}}| + |y - y_{\text{goal}}| < \varepsilon \end{cases} \tag{4.212}$$

Where ε is a distance threshold. There are obvious nonlinearities in this expression and it cannot be directly implemented in the MILP without manipulation first. Let

$M \gg \max(\Sigma_k)$:

$$\Sigma_{k+1} \leq \Sigma_0 t_k^2 + M(1 - z_k) \tag{4.213}$$
$$\Sigma_{k+1} \geq \Sigma_0 t_k^2 - M(1 - z_k) \tag{4.214}$$
$$\Sigma_{k+1} \leq \Sigma_k + z_k M \tag{4.215}$$
$$\Sigma_{k+1} \geq \Sigma_k - z_k M \tag{4.216}$$
$$z_k \in \{0, 1\} \tag{4.217}$$
$$z_k \geq -|x - x_{\text{goal}}| - |y - y_{\text{goal}}| + \varepsilon \tag{4.218}$$

The absolute values can be rewritten as linear constraints:

$$\Sigma_{k+1} \leq \Sigma_0 t_k^2 + M(1 - z_k) \tag{4.219}$$
$$\Sigma_{k+1} \geq \Sigma_0 t_k^2 - M(1 - z_k) \tag{4.220}$$
$$\Sigma_{k+1} \leq \Sigma_k + z_k M \tag{4.221}$$
$$\Sigma_{k+1} \geq \Sigma_k - z_k M \tag{4.222}$$
$$z_k \in \{0, 1\} \tag{4.223}$$
$$z_k \geq -a_k^x - a_k^y + \varepsilon \tag{4.224}$$
$$x_k - x_{\text{goal}} \leq a_k^x \tag{4.225}$$
$$-x_k + x_{\text{goal}} \geq a_k^x \tag{4.226}$$
$$y_k - y_{\text{goal}} \leq a_k^y \tag{4.227}$$
$$-y_k + y_{\text{goal}} \geq a_k^y \tag{4.228}$$

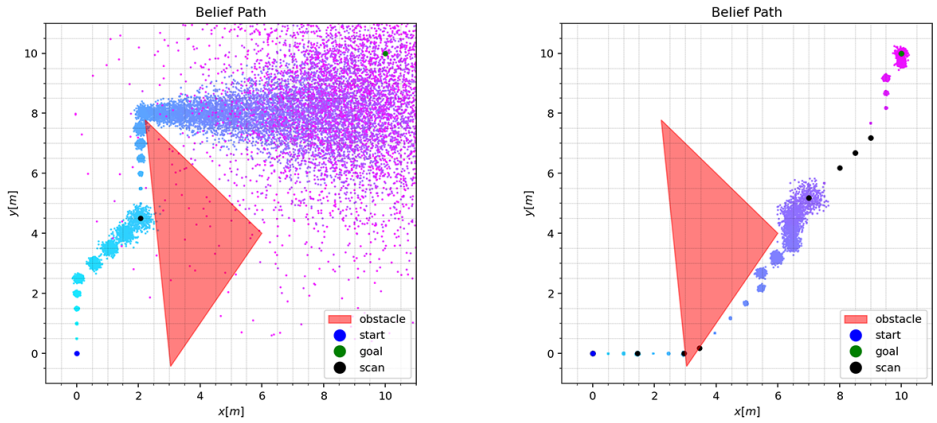
These covariance constraints reduce the solver's sensitivity to the number of steps N, allowing solutions with more steps than necessary that do not exhibit unnecessary covariance growth. Furthermore, this modeling choice improves performance over the introduction of a NoOP action. It is worth emphasizing that cases in which the optimal solution would require waiting for an event to occur mid-plan are not captured by this covariance freezing trick and require switching back to the NoOP action. An example of a use-case where NoOP action might be useful is waiting for an orbiter flyover before communicating back to the ground.

4.5.8 TAMP optimization objective

This section will examine the optimization objective for the general task and motion planning case. It is reasonable to think of minimizing the total distance traveled by minimizing the $c\|U_k\|$ as in Section 4.5.2, but this would lead the solution passing as close as possible to the obstacles, without paying attention to the number of information gain activities that are scheduled. In practice, information gaining is expensive as the *scan environment* activity takes up a time in the order of 1 minute to be completed. This cost must be factored into the optimization process to fully capture the tradeoff between distance from obstacles and stop frequency. The first, simple way to incorporate the different action costs is to write the optimization objective as:

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{k=0}^{N-1} \sum_{i=0}^{N_a} {}^{i}c_k \, {}^{i}y_k \tag{4.229}$$

Where $\mathbf{y}_k = \{{}^{1}y_k, {}^{2}y_k, \ldots, {}^{N_a}y_k\}$, be the vector of selector variables Recall also that N_a is the number of actions. This way of writing the objective translates to setting a fixed cost coefficient for each action ${}^{i}c$ and weighing the action chosen at time step k with this fixed cost. It is important to note that cost coefficients are assumed to be constant over time. If all costs are equal ${}^{1}c = {}^{2}c = \cdots = {}^{N_a}c = c$, the optimization process will not have much information that will aid decision making, as both all actions will look equally appealing. On the other hand, if the costs are different, the planner will try to maximize the number of least costly actions. This is also not desirable for the move scan scenario, and two examples are sufficient to illustrate the problem. Firstly, imagine that ${}^{S}c > {}^{M}c$. In this case, the planner would try to maximize *move* actions, searching for paths as far from the obstacles as possible. This approach would be similar to a completely risk-averse behavior. On the other hand, if ${}^{S}c < {}^{M}c$, the planner would try to assign the maximum possible number of scan actions without paying attention to whether these actions are necessary or not. In a world without obstacles, this cost structure would lead to several stop behaviors being scheduled, even though they would be completely unnecessary. This behavior can be observed in Figure 4.15. On the left side, the cost of scanning is higher than the cost of moving, leading to a trajectory with few stops.



(a) ${}^{S}c > {}^{M}c$ (b) ${}^{M}c > {}^{S}c$

Fig. 4.15 Effect of varying cost weights in the objective function (Equation 4.229). Decreaseing the cost of scanning will lead to a plan with a shorter path, but more of scans.

Thus, achieving the desired behavior is once again a matter of shaping the cost structure, just as reward function shaping was central for the POMDP case (recall that cost and reward are tightly coupled concepts). This can be done by writing:

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{k=0}^{N-1} \sum_{i=0}^{N_a} {}^{i}c_k \, {}^{i}f_k(X,U) \, {}^{i}y_k \tag{4.230}$$

This expression, in general, is non-linear, and the function $f(X,U)$ must be designed in a way that can be linearized using tricks from the optimization community. For the move-stop scenario, ${}^{S}f(X,U) = {}^{S}c$ will remain a constant function, while ${}^{S}f(X,U) = {}^{M}c|u|_k$ will be a path length minimization objective. Thus, the optimization objective can be rewritten as:

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{k=0}^{N-1} \left({}^{S}c_k \, {}^{S}y_k + \sum_{j=0}^{N_u-1} {}^{M}c_k |u|_k^j \, {}^{M}y_k \right) \tag{4.231}$$

Now, it is necessary to linearize $|u|_k \, {}^{M}y_k$. This can be done by first substituting $|u|_k^j$ with a variable z_k^j that behaves like an absolute value with the addition of two constraints.

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{k=0}^{N-1} \left({}^{S}c_k \, {}^{S}y_k + \sum_{j=0}^{N_u-1} {}^{M}c_k z_k^j \, {}^{M}y_k \right) \tag{4.232}$$

$$u_k^j \leq z_k^i \tag{4.233}$$

$$-u_k^j \leq z_k^j \tag{4.234}$$

Finally, the slack variable $r_k^{z^j} = z_k^j \, {}^{M}y_k$ can be introduced:

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{k=0}^{N-1} \left({}^{S}c_k \, {}^{S}y_k + \sum_{j=0}^{N_u-1} {}^{M}c_k r_k^{z^j} \right) \tag{4.235}$$

The following constraints need to be added for each k and j:

$$u_k^j \leq z_k^j \tag{4.236}$$

$$-u_k^j \leq z_k^j \tag{4.237}$$

$$r_k^{z^j} \geq z_{\min}^j \, {}^{M}y_k \tag{4.238}$$

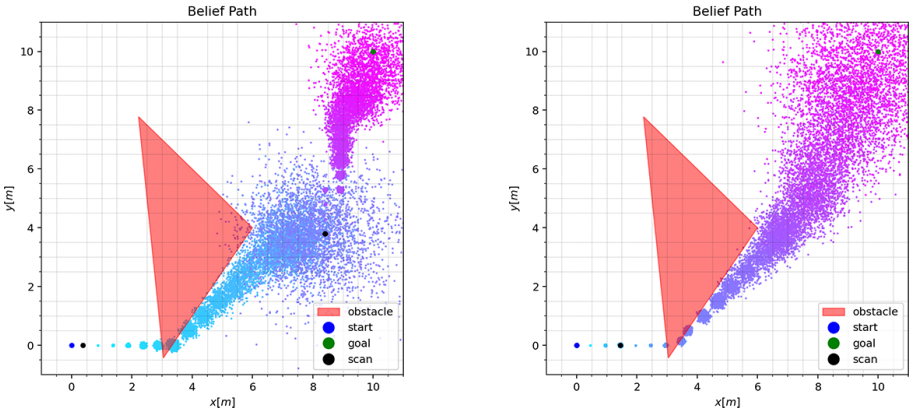
$$r_k^{z^j} \leq z_{\min}^j \, {}^{M}y_k \tag{4.239}$$

$$r_k^{z^j} \leq z_k^j - z_{\min}^j (1 - {}^{M}y_k) \tag{4.240}$$

$$r_k^{z^j} \geq z_k^j - z_{\max}^j (1 - {}^{M}y_k) \tag{4.241}$$

Choosing the formulation in Equation 4.235 will result in the solution taking N steps to reach the goal. This means that modifying N will lead to changes in the speed at which the robot moves. The fixed cost strategy can be used to achieve both the minimization of path length and number of scans, by introducing a NO-OP action. Using the simpler fixed cost coefficient approach is more desirable, as each additional constraint adds computational complexity to the problem, and near-real-time performance is the goal.

It is generally desirable to reduce the number of slack constraints to improve the formulation's scalability. This can be done by using different functions that approximate the path minimization objective. Note that minimizing scan will always consist of minimizing the scan selection variable ${}^{S}c_k \, {}^{S}y_k$. One way to formulate this is by aiming to minimize the sum of the Manhattan distance from the goal at



(a) Constraint Satisfaction (b) Cost minimization

Fig. 4.16 Effect of adding a no-op action to the robot's action space

each time step. This will incentivize the agent to move quickly toward the goal but requires computing two absolute values.

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{i=0}^{N} \left({}^{S}c_k \, {}^{S}y_k + {}^{M}c_k |X_k - X_{\text{goal}}| \right) \tag{4.242}$$

This can be rewritten by separating X into its spatial components x, y and creating two variables that behave like the absolute values of the distance from the goal a_k^x, a_k^y.

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{i=0}^{N} \left({}^{S}c_k \, {}^{S}y_k + {}^{M}c_k (a_k^x + a_k^y) \right) \tag{4.243}$$

$$x_k - x_{\text{goal}} \leq a_k^x \tag{4.244}$$

$$-x_k + x_{\text{goal}} \geq a_k^x \tag{4.245}$$

$$y_k - y_{\text{goal}} \leq a_k^y \tag{4.246}$$

$$-y_k + y_{\text{goal}} \geq a_k^y \tag{4.247}$$

This optimziation objective can lead to unwanted behavior, as the agent will try to spend as much time as possible close to the goal, even if that entails a longer path overall. This is because the minimization objective is the integral of distance, so a path that is long but stays close to the objective for most of the time steps will cost

less than a path that is shorter but spends more time farther away. Another better way of getting a true path length minimization is to create an indicator function that takes the value 1 when the agent is outside of the goal set, and 0 if inside the goal set. This is the inverse concept of what was implemented for the covariance dynamics freezing trick.

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{i=0}^{N} \left({}^{S}c_k \, {}^{S}y_k + {}^{M}c_k \mathbb{1}_k^{\mathrm{g}} \right) \tag{4.248}$$

Indicator functions are by definition binary decision variables. This binary variable $\mathbb{1}_k^{\mathrm{g}} = z_k^{\mathrm{g}} \in 0,1$ can be created and constrained to behave as the indicator function by using the big-M trick. The optimization objective, with supporting constraints can thus be written as:

$$\min_{\mathbf{U}_{0:N-1},\mathbf{y}_{0:N-1}} \sum_{i=0}^{N} \left({}^{S}c_k \, {}^{S}y_k + {}^{M}c_k z_k^{\mathrm{g}} \right) \tag{4.249}$$

$$z_k^{\mathrm{g}} \in \{0,1\} \tag{4.250}$$

$$a_k^x + a_k^y - \varepsilon \leq M z_k^{\mathrm{g}} \tag{4.251}$$

$$\varepsilon - a_k^x - a_k^y \leq M(1 - z_k^{\mathrm{g}}) \tag{4.252}$$

$$x_k - x_{\mathrm{goal}} \leq a_k^x \tag{4.253}$$

$$-x_k + x_{\mathrm{goal}} \geq a_k^x \tag{4.254}$$

$$y_k - y_{\mathrm{goal}} \leq a_k^y \tag{4.255}$$

$$-y_k + y_{\mathrm{goal}} \geq a_k^y \tag{4.256}$$

It can be noted that z_k^{g} can be set to zero only when $a_k^x + a_k^y \leq \varepsilon$, and it can only be set to 1 when $a_k^x + a_k^y \geq \varepsilon$.

4.5.9 Notes on chance constraints

With one obstacle, it has been shown how the chance constraint can ensure that the probability of the agent entering an unsafe area is never above a threshold Δ. Adding multiple obstacles requires an extension with respect to the single obstacle case. Recall that the purpose of a chance constraint is to ensure that the probability of the

system's state being in an obstacle set X^{obs} is less than Δ. This can be written as:

$$\Pr(x \in X^{\text{obs}}) < \Delta \tag{4.257}$$

For multiple obstacles, $X^{\text{obs}} = \{X_1, X_2, \ldots, X_n\}$, the correct way of writing this constraint is:

$$\sum_{i=0}^{n-1} \Pr(x \in X_i) < \Delta \tag{4.258}$$

Implementing multiple polygonal obstacle constraints separately, is incorrect as the structure would be:

$$\bigwedge_{i=0}^{n-1} [\Pr(x \in X_i) < \Delta] \tag{4.259}$$

A 1D case can explain more clearly how Equation 4.259 could lead to underestimating the current risk level. Let $x \sim \mathcal{N}(0,1)$ be the state distribution, and let there be two half plane obstacles - the first for $x < -b_1$ and the second for $x > b_2$ (where $b_1, b_2 \in \mathbb{R}^+$). In this case, the obstacle set can be written as $X^{\text{obs}} = \{x < -b_1, x > b_2\}$. If the two chance constraints are implemented separately as in Equation 4.259, it can be written as:

$$\Pr(x < -b_1) < \Delta \wedge \Pr(x > b_2) < \Delta \tag{4.260}$$

An expression that satisfies these conditions is:

$$\Pr(x < -b_1) = \Delta - \delta\varepsilon \quad \Pr(x > b_2) = \Delta - \delta\varepsilon \tag{4.261}$$

Where $\delta\varepsilon$ is an infinitesimal. Thus, the total risk level in this example would be higher than the single constraint risk level Δ:

$$2(\Delta - \delta\varepsilon) > \Delta \tag{4.262}$$

Note that the likelihood of x being in the obstacle set is by definition:

$$\int_{-\infty}^{-b_1} p(x)\mathrm{d}x + \int_{b_2}^{\infty} p(x)\mathrm{d}x = \Pr(x < -b) + \Pr(x > b) = \phi(-b_1) + 1 - \phi(b_2) \tag{4.263}$$

Constraints have been posed on the probability of violating safety requirements at each time step. This is different from enforcing a constraint over the probability of mission success. In fact, obstacle chance constraints can prevent the agent from colliding with obstacles with a likelihood of $1-\Delta$ at each time step, and still have an overall mission success probability close to zero if the mission is long enough. The reason for this is that the probability of mission failure is the integral over all time steps of the probability of intersecting obstacles.

$$\sum_{k=0}^{N} \Pr(x_k \in X^{\text{obs}}) < \Delta \tag{4.264}$$

Considering multiple obstacles leads to:

$$\sum_{k=0}^{N} \sum_{i=0}^{n-1} \Pr(x_k \in X_i^{\text{obs}}) < \Delta \tag{4.265}$$

These aspects are typically solved with the concept of risk allocation [112]. The idea is to assign a risk budget that can be spent at each time step.

4.5.10 Removing corner cutting

In the MILP formulation, the system is seen through the lens of discrete time steps. When deploying a discrete-time system to a continuous-time environment, constraints that are satisfied in the discrete case might be violated in continuous space. This happens because constraint checking in the MILP solver occurs only at a finite number of time steps, so behaviors like corner cutting can emerge due to a lack of explicit constraints. To avoid these circumstances, the constraints need to be tweaked. For the corner-cutting case, there are two sub-problems. Firstly, the mean of the trajectory should not pass through the obstacle. Secondly, the chance constraints should not be violated at any of the intermediate points between time samples. The mean constraint has been investigated in [113] and can be solved by forcing the agent at time k to share the same active obstacle constraint at time $k+1$ for each obstacle. Let i be the obstacle's index, j an index for the obstacle's lines, and $p_{k,i,j} \in \mathbb{R}^+$ a continuous variable. Assume that for each polygonal obstacle i, there will be N_i corners $j \in \{1,\dots,N_i\}$. In addition, let the ith obstacle activation variable of the obstacle's jth side be $y_{k,i,j}$. A no-corner-cutting constraint can be

enforced for the agent's mean for each time $k \in \{1,\dots,N\}$ and for each obstacle $i \in \{1,\dots N_o\}$ as:

$$\sum_{j=1}^{N_i} p_{k,i,j} \geq 1 \tag{4.266}$$

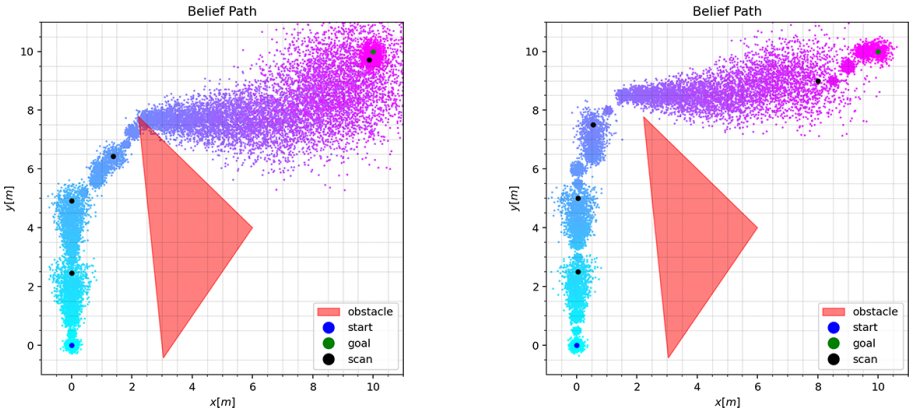
$$p_{k,i,j} \geq y_{k,i,j} + y_{k-1,i,j} - 1 \tag{4.267}$$

$$p_{k,i,j} \leq y_{k,i,j} \tag{4.268}$$

$$p_{k,i,j} \leq y_{k-1,i,j} \tag{4.269}$$

$$p_{k,i,j} \geq 0 \tag{4.270}$$

Note that if the problem has N time steps, N_o polygonal obstacles, each with N_i sides, this system will have NN_oN_i additional constraints. In Figure 4.17, it is clear how adding no-corner-cutting constraints has forced the agent to move farther away from the obstacle's corners.



(a) Solution without no-corner-cutting constraints

(b) Solution with no-corner-cutting constraints

Fig. 4.17 Effects of adding no-corner-cutting constraints

4.5.11 Recapping the MILP formulation

The CC-TAMP, MILP formulation was incrementally built in the previous sections, starting from a simple motion planning problem. Here, all the elements that go

into the Task and Motion Planning under uncertainty formulated as a Mixed Integer Linear Program are collected.

$$\min_{U_{0:N-1},y_{0:N-1}} \sum_{i=0}^{N} \left({}^{S}c_k \, {}^{S}y_k + {}^{M}c_k z_k^{\text{og}} \right) \tag{4.271}$$

$$z_k^{\text{og}} \in \{0,1\} \tag{4.272}$$

$$a_k^x + a_k^y - \varepsilon \leq M^g z_k^{\text{og}} \tag{4.273}$$

$$\varepsilon - a_k^x - a_k^y \leq M^g (1 - z_k^{\text{og}}) \tag{4.274}$$

$$x_k - x_{\text{goal}} \leq a_k^x \tag{4.275}$$

$$-x_k + x_{\text{goal}} \geq a_k^x \tag{4.276}$$

$$y_k - y_{\text{goal}} \leq a_k^y \tag{4.277}$$

$$-y_k + y_{\text{goal}} \geq a_k^y \tag{4.278}$$

Mean Transition Dynamics:

$$\mu_{X_{k+1}} = \sum_{i=1}^{N_a} \left({}^{a^i}A \, {}^{a^i}r_k^{\mu_X} + {}^{a^i}B \, {}^{a^i}r_k^{u} + {}^{a^i}C \right) \tag{4.279}$$

Slack variables mean:

$$\begin{aligned} r_k^{\mu_X} &\geq \mu_{X\min} \, {}^{a^i}y_k \\ r_k^{muX} &\leq \mu_{X\max} \, {}^{a^i}y_k \\ r_k^{\mu_X} &\leq h_k - \mu_{X\min}(1 - {}^{a^i}y_k) \\ r_k^{\mu_X} &\geq \mu_{X_k} - \mu_{X\max}(1 - {}^{a^i}y_k) \end{aligned} \tag{4.280}$$

slack variables for control input:

$$\begin{aligned} r_k^{u} &\geq u_{\min} \, {}^{a^i}y_k \\ r_k^{u} &\leq u_{\max} \, {}^{a^i}y_k \\ r_k^{u} &\leq h_k - u_{\min}(1 - {}^{a^i}y_k) \\ r_k^{u} &\geq u_k - u_{\max}(1 - {}^{a^i}y_k) \end{aligned} \tag{4.281}$$

State transition matrix definitions:

$$N_X = 4, \quad N_u = 4, \quad N_a = 2 \tag{4.282}$$

$$\mathscr{A} = \{Move = M, Scan = S\} \tag{4.283}$$

$$X = \begin{bmatrix} x & y & t^2 & t \end{bmatrix}^T \tag{4.284}$$

$$^{M}A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2\Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \; ^{M}B = \begin{bmatrix} \Delta t & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.285}$$

$$^{S}A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \; ^{S}B = 0_{4x4}$$

$$^{M}\omega = \mathscr{N}(0, \Sigma^{\omega}), \quad ^{S}\omega = 0 \tag{4.286}$$

$$^{M}C = \begin{bmatrix} 0 \\ 0 \\ (\Delta t)^2 \\ \Delta t \end{bmatrix}, \quad ^{S}C = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{4.287}$$

Covariance propagation with growth freeze in a neighbourhood of the goal:

$$\Sigma_{k+1} \leq \Sigma_0 t_k^2 + M^c(1 - z_k^{\text{ig}}) \tag{4.288}$$

$$\Sigma_{k+1} \geq \Sigma_0 t_k^2 - M^c(1 - z_k^{\text{ig}}) \tag{4.289}$$

$$\Sigma_{k+1} \leq \Sigma_k + z_k^{\text{ig}} M^c \tag{4.290}$$

$$\Sigma_{k+1} \geq \Sigma_k - z_k^{\text{ig}} M^c \tag{4.291}$$

$$z_k^{\text{ig}} \in \{0, 1\} \tag{4.292}$$

$$z_k^{\text{og}} + z_k^{\text{ig}} = 1 \tag{4.293}$$

Polygonal chance constraints:

$$\bigcap_{i=1}^{N_e^o} \left[H_i^o \mu_{X_k} + \phi^{-1}(\Delta) H_i^{oT} \Sigma_{X_k} H_i^o \geq {}^o b_i - M^o (1 - y_k^{o,i}) \right] \tag{4.294}$$

$$\sum_{i=1}^{N_e^o} y_k^{o,i} = {}^o N_e - 1 \tag{4.295}$$

4.5.12 Observations about the planning problem's structure

In general, an agent that acts in an environment and receives observations when an information-gaining action is selected has been considered so far. The critical insight is that due to noise in sensors and imperfect models, both state and observations are random variables. As seen in Figure 4.18, the act of observing will yield a sample from a *probability distribution* - specifically, a sample from the pre-scan belief state. When propagating beyond the first scan, the belief state becomes more complex to visualize; it becomes a *probability distribution over probability distributions*. Planning over a distribution of distributions is known as planning over *Hyperbeliefs*. The visualization in Figure 4.18 also highlights the key problem that is hidden in hyperbelief planning. When projecting into the future, the belief state's distribution grows *exponetially*. This type of problem can be formulated as a Partially Observable Markov Decision Process (POMDP):

$$POMDP = \langle S, A, T, \Omega, R, \gamma, \mathscr{Z} \rangle \tag{4.296}$$

Where S is the state space, A the action space, $T = p(s_{k+1}|s_k, a_k)$ the transition function, Ω the observation space, $\mathscr{Z} = p(o_{k+1}|s_{k+1}, a_k)$ the observation function, R the reward function and γ the discount factor [97]. Since the agent does not have perfect state knowledge, planning in POMDPs occurs in Belief Space - probability distribution over states. When projecting belief state in the presence of observations, the problem becomes a hyperbelief planning problem. In hyperbelif planning, the state's probability distribution is a function of random observations. Thus, state belief becomes a probability distribution over probability distributions (Figure 4.18).

POMDP solvers get around this complexity in continuous state spaces by approximating this hyperbelief structure through samples - similarly to a particle filter. The MILP formulation includes an assumption about the observation's structure

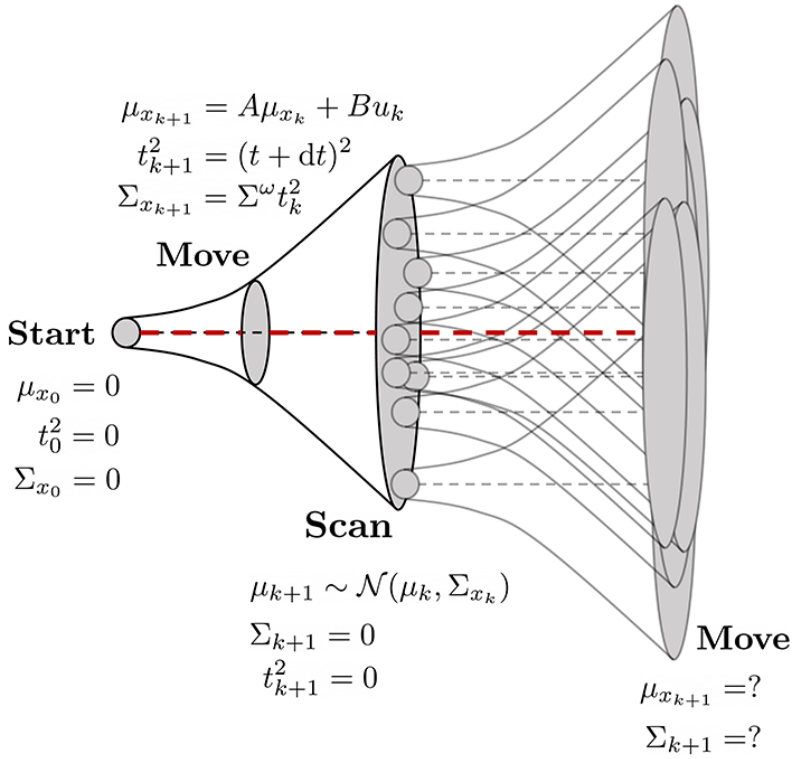


Fig. 4.18 Visualization of the hyperbelief-planning structure of the move-scan problem

that sidesteps hyperbelief planning at the cost of optimality guarantees. In fact, the Scan transition dynamics have assumed that the agent receives an observation that coincides with the most likely estimate of its pre-scan uncertainty distribution. If an entire plan generated by the MILP planner were to be executed, neither optimality nor adherence to safety constraints would be guaranteed, as the MILP does not model the possibility of being outside the distribution's MLE after scanning. A way around this suboptimality is to re-plan after each scan when a new state observation is received. This guarantees that the safety constraints are respected but does not guarantee overall plan optimality. This approach is similar to a Model Predictive Control (MPC) approach (Figure 4.19). Even though optimality guarantees are not provided, the MILP planner could still perform well against other state-of-the-art planners tasked with solving the same problem. The following sections will outline the comparison effort that was undertaken in this direction.

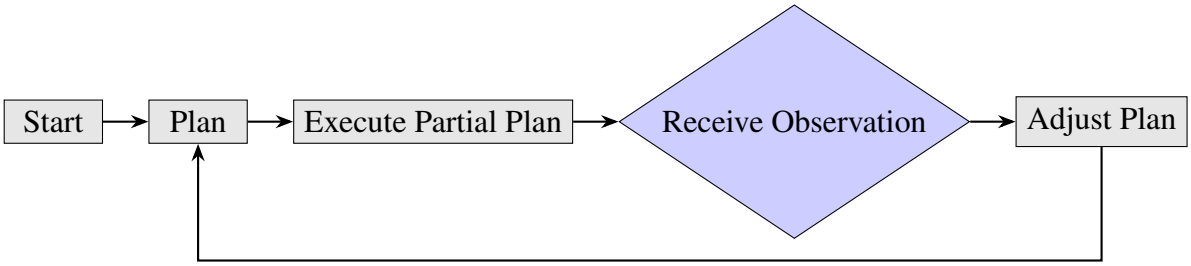


Fig. 4.19 Model Predictive Control approach

4.5.13 Notes on sensor noise when scanning

Sensor noise in an important factor in determining the robustness of any decision making under uncertainty algorithm, where an agent interacts with an environment while receiving observations from imperfect sensors. However, for the sake of simplifying the model and focusing on the core aspects of the task and motion planning algorithm, throughout this work there is an assumption that the "scan environment" action leads to no observation noise. In a real world scenario, sensor noise would make it so that the agent can not trust the observations it receives, which would have to be expressed as:

$$Z_k = H(X_k) + V_k \tag{4.297}$$

Where Z_k is the observation, H is the observation function, and V_k is noise. In this dissertation, the source of uncertainty is the *lack of reliable observations* during the move action, not the sensor noise. The agent's inability to observe its state when moving, combined with the noisy transition dynamics leads to uncertainty over time. Modeling noisy sensors would add a further layer of complexity to an already complex formulation, and was therefore left for future work. Neglecting sensor noise can be a justifiable assumption if the sensor noise is small compared to the robot's and environment's scale. Future work should address the integration of realistic sensor models, including observation noise, into the planning framework, and should evaluate the resulting impact on planning performance and robustness.

Chapter 5

Experiments and Results

This chapter details the computational experiments used to evaluate the algorithms presented in Chapter 4, and includes a hardware implementation of the most promising algorithm. The chapter begins with the introduction of a baseline planner inspired by the classical approach of decoupled task and motion planning, serving as an anchor for comparative evaluation. The second part introduces the computational framework used to evaluate the planners against each other, presenting the results and insights derived from this framework. The final section outlines the hardware implementation of the MILP algorithm, marking an initial qualitative step toward a more detailed analysis of the planner's performance on hardware, which is left for future work.

5.1 Baseline Planners

As a baseline for comparison, the classical approach of decoupling task from motion planning is taken. Two-stage approaches are advantageous as they allow the decoupling of the path planning problem from the uncertainty-aware task scheduling problem, thus greatly simplifying the formulation effort. Firstly, a path is planned using the A* heuristic graph search algorithm [114] over an inflated obstacle set. The path planner module starts by inflating each obstacle using a filter that increases the obstacle's size by ε_g at each pass. This inflation procedure is done n_I times, and in the end, each obstacle will have grown by $h = n_I \varepsilon_g$. The planning stage searches for a minimal length path over this inflated set. The second stage consists of move-scan

scheduling, performed as the mission progresses. State uncertainty can grow in difficult-to-predict ways if relying on exteroception. For the sake of simplicity, it is assumed that covariance follows a known growth law f that is a function of moving time ${}^{M}t$, as would be the case for IMU integration. This growth function should be positive and monotonic. This implies that the inverse f^{-1} should be computable.

$$\Sigma^2 = f({}^{M}t) \tag{5.1}$$

If covariance grows following a predictable law, then the threshold planner becomes a fixed scanning frequency planner, and this scanning frequency will be a function of the worst-case mobility scenario. Figure 5.1 shows this worst-case scenario, where the planner is always moving along the boundary of an inflated obstacle; this scenario is used to determine the frequency at which the planner should stop to ensure that the safety constraints are never exceeded. Heuristic planners such as A* find paths at the obstacle set's boundary. Thus, if the obstacle inflation has increased the obstacle's size by h, a trajectory passing by states x_k and x_{k+1} along the inflated obstacle set's boundary can be imagined. The covariance at time $k+1$ is only a function of the moving time ${}^{M}t$. Thus, a chance constraint can be written that will give the maximum moving time that the planner can safely schedule, without performing any state projection. The same trick is used where $p(x)$ is re-written as a function of μ_x, Σ_x, z, where $z \sim \mathcal{N}(0,1)$, as a way to express the chance constraint as a function of the standard normal distribution's cumulative density function. Note that in this case, the origin can be centered in x_{k+1}, therefore setting the obstacle at $-h$ and $\mu_x = 0$. Additionally, the symmetry of this case is leveraged by looking only along the axis that joins x_{k+1} with the obstacle.

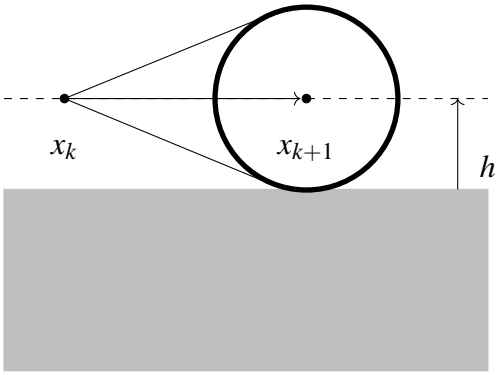


Fig. 5.1 Covariance growth of the agent moving along the inflated obstacle set's boundary

$$\Pr(x \leq -h) \leq \Delta \tag{5.2}$$

$$x = \mu_x + \Sigma_x z \quad z \sim \mathcal{N}(0,1) \tag{5.3}$$

$$\Pr(\mu_x + \Sigma_x z \leq -h) \leq \Delta \quad \mu_x = 0 \tag{5.4}$$

$$\Pr(z \leq -\frac{h}{\Sigma_x}) \leq \Delta \tag{5.5}$$

$$\phi\left(\frac{-h}{\Sigma_x}\right) \leq \Delta \tag{5.6}$$

$$\Sigma_x \geq \frac{-h}{\phi^{-1}(\Delta)} \tag{5.7}$$

Here, the property that Σ_x is a function of time is exploited. Furthermore, the $\geq$ becomes an equality, to express the constraint's boundary.

$$f\left(\max({}^{M}t)\right) = \frac{h}{\phi^{-1}(\Delta)} \tag{5.8}$$

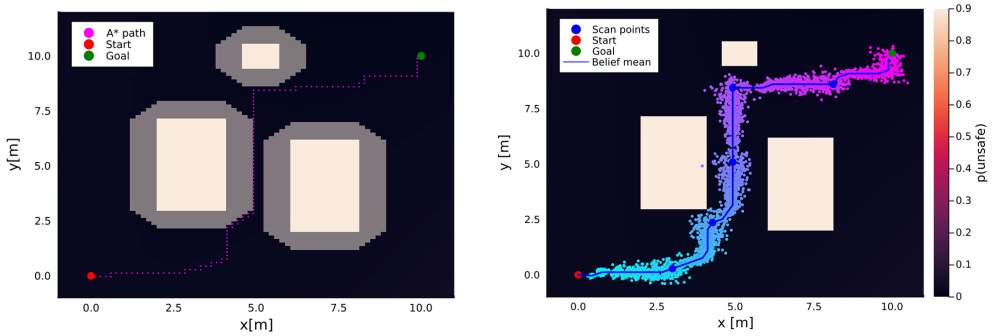
$$\max({}^{M}t) = f^{-1}\left(\frac{h}{\phi^{-1}(\Delta)}\right) \tag{5.9}$$

The rationale for this planner is that it is simple and reliable. By knowing how state uncertainty will evolve, it can be guaranteed that the safety constraints are never violated with a simple formulation. Additionally, the planning time can easily be neglected when compared to the execution time, as solving A* over a grid map is a trivial task, and scan scheduling can be done a priori based on a simple expression.

Thus, the mission execution time can be assumed to be approximately equal to the action execution time (the planning time can be neglected):

$$t_E \gg t_P \implies t_M \approx t_E \tag{5.10}$$

The obvious downside to this approach is that the planner is unaware of the robot's state and can not adapt the move-stop cadence to the robot's environment. In fact, it might not be much of a problem for the agent if state uncertainty grows above the threshold when no obstacles are in sight. This task and motion planner will always schedule scans, assuming it acts under the worst possible movement condition along an obstacle boundary. It is easy to see how, for a majority of the time, this assumption is too conservative and will lead to wasted execution time.



(a) Stage 1: Path planning

(b) Stage 2: Task scheduling

Fig. 5.2 The two stages of the baseline planner. On the left, 5.2a shows the path planning stage, whereas on the right 5.2b depicts scan scheduling

5.2 Baseline planner improvements

Our previous approach to managing risk along a path worked on the principle of maintaining a maximum risk threshold at all times. While effective in maintaining safety, this method was inherently conservative. It continuously presumed worst-case conditions and instigated stops more frequently than necessary. This behavior can result in prohibitive costs, mainly when the actions designed to gain information are expensive.

A more nuanced strategy would be to schedule scans only when the risk surpasses a predetermined threshold. The advantage of looking at the current risk level is that

it is aware of the risks the environment poses to the agent, thus reducing unnecessary information-gaining actions and lowering the overall plan execution cost.

Mathematically,the time step to schedule a stop, denoted by t_{stop}, can be defined as the shortest time t such that the probability of our system state X being in the obstacle set X^{obs} exceeds a predefined risk threshold Δ. This can be expressed as:

$$t_{stop} = \min\{t : Pr(X_t \in X^{obs}) < \Delta\} \tag{5.11}$$

This formulation implies that a stop is scheduled at the earliest time step when the risk of encountering an obstacle exceeds the threshold Δ. The probability of being in the unsafe set is computed by Monte Carlo integration by sampling a 2D Gaussian distribution fitted onto the planned path. It is assumed that covariance can be computed as a function of the movement time.

$$\Sigma_k = \Sigma_0 f(t) \tag{5.12}$$

For this application, a linear covariance growth with time is used. In Figure 5.3, there is a representation of this planner's behavior. The image shows several runs of the same agent tasked with reaching a goal behind a triangular obstacle. Initial and terminal conditions are constant, and each run shares the same noise model. It is apparent from the figure that path planning and task scheduling are solved in stages, as the path passes closer to the obstacles than it needs to, and scans are scheduled close to the obstacle to avoid violating chance constraints.

5.3 Monte Carlo Simulation Framework

The performance of the different planners was compared through a Monte Carlo Simulation (MC) framework. This framework consists of two key components: the planning component and the execution component. The execution component encapsulates the problem's noisy linear dynamics and consists of a simulator that transforms a sequence of control actions into a sequence of states, all the while

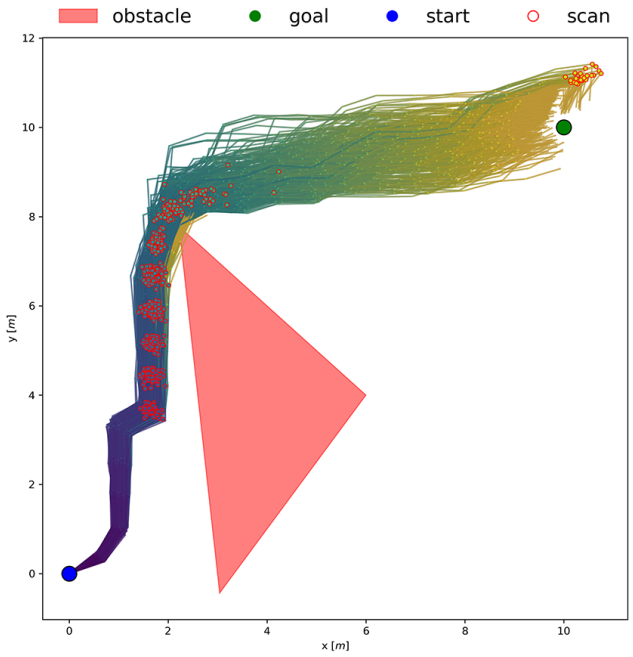


Fig. 5.3 Trajectories of Monte Carlo runs of baseline planner in a single polygonal obstacle environment.

checking for safety constraint violations. Equation 5.13 shows how the dynamics are implemented in the Execution component.

$$X_{k+1} = {}^{a_k}AX_k + {}^{a_k}BU_k + {}^{a_k}\omega_k \tag{5.13}$$

Where $\omega_k \sim \mathcal{N}(0, \Sigma^{\omega})$. In general, this execution node can support action-dependant transitions, but in this implementation, only move-action dynamics are considered. The reason for this is that the only other action in our action space - scanning - has the effect of starting a new simulation with initial conditions centered around the agent's true position. The planning components wraps around each of the three planners described in the previous sections. The POMDP planner is implemented in the Julia programming language, leveraging the JuliaPOMDP library [104]. The Basline planner is also implemented in Julia. The MILP planner on the other hand is implemented is C++ and uses Google's or-tools [115] for model definition and solver wrapping. The commercial Gurobi [116] solver is used due to its first-of-class Mixed Integer Program solver.. MILP and Two-Stage are one-shot planners. This means that they either have a solution, or they have nothing. The

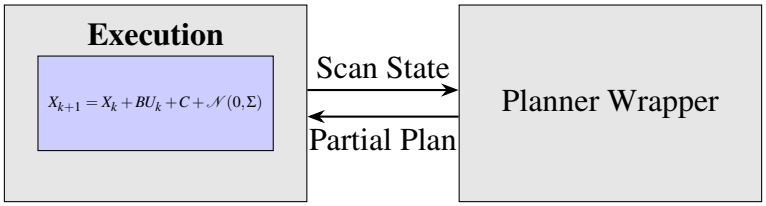


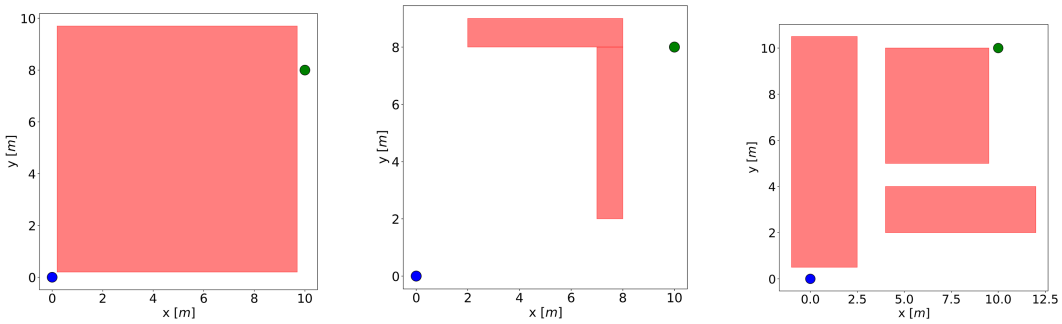
Fig. 5.4 Architecture diagram of Monte Carlo simulation environment

POMDP planner with MCTS is an any-time planner, and a solution can be achieved for any computational cost (at the expense of solution quality). While in principle there is no reason not to use the same thread to run both simulation and planning, this multi-language, multi-planner configuration made it convenient to separate the two functions into separate processes. The Robot Operating Sysem (ROS), was used to manage communication between planning and execution, through a server client architecture (shown from a high-level in Figure 5.4).

The execution node also acts as an exernal observer, by having knowledge about the agent's goal and judging whether the goal has been attained or not. If there is an error between state X and the goal set X^g, the execution node sends a plan request to the wrapper, describing the current state. The planner wrapper receives this request and dispatches it as initial condition to a planner (e.g., POMDP, MILP, Two-Stage) based on a configuration file setting. The wrapper then parses the planner's output and extracts the sequence of commands up to the first scan. This sequence is packaged and sent back to the Execution node as a response. The execution node, in turn, receives that partial plan and executes it, returning back to the start. This process closely follows what is described in Figure 4.19.

The simulation is launched and coordinated through a series of bash script and roslaunch files. For this performance assessment, handcrafted maps were chosen rather than randomly generated terrain, as a way to test specific features of the planners. As seen in Figure 5.5, three 2.5D maps were chosen, the first (Map-1) with a single rectangular obstacle, the second (Map-2) with two obstacles, and the last (Map-3) with three rectangular obstacles. Map-1 represents a worst-case scenario for the two-stage planner, with the start, end, and minimum length path all being very close to an obstacle. Map-2 is similar to Map-1, with an added risk for the agent consisting of a local optimum at the two block's intersection. Map-3 has a narrow

pathway that is technically the shortest path, but requires many stops to be safely navigated.



(a) Map-1 (b) Map-2 (c) Map-3

Fig. 5.5 Environments used to compare the three planner's performance.

5.4 Metrics of interest

Besides qualitative comparisons, when comparing planners, it is important to focus on *solution cost* and *computational cost*. Solution cost is a concept closely related to solution quality - in fact, they are inversely proportional. Given a fixed risk tolerance, the solution's cost can be quantified by measuring the *Execution Time*, which represents the time required to achieve the planning goals under the obtained plan, *excluding* the time spent computing a plan. Excution time can be computed by quantity by multiplying the number of steps in the plan each by the duration of the action selected at that time step.

$$t_E = \sum_{k=1}^{N} \sum_{i=1}^{N_a} a^i y_k \, a^i \Delta t \tag{5.14}$$

The relative weight of scanning-vs-moving cost parameters vastly influences the planner's behavior. As the scanning cost decreases, the optimal task and motion plan will gradually approach the minimum length path. Conversely, as scanning cost increases the solution will approach a robust risk-aware path without scans.

As seen in Table 5.1, scanning is assumed to be much more expensive than moving, with respective time costs of 100 seconds and 0.5 seconds. This number closely matches the hardware system's characteristics, and sufficiently penalizes

Table 5.1 Actions cost

Action	Cost
Move	0.5
Scan	100

information gaining so that optimal policies follow very different paths from the minimum length paths.

Computational cost is the time spent planning for each mission. Assuming that the number of planning and execution iterations needed to reach the goal for a given mission is N and the system clock time that passes during the ith planning step is tp_i, the computational cost can be expressed as:

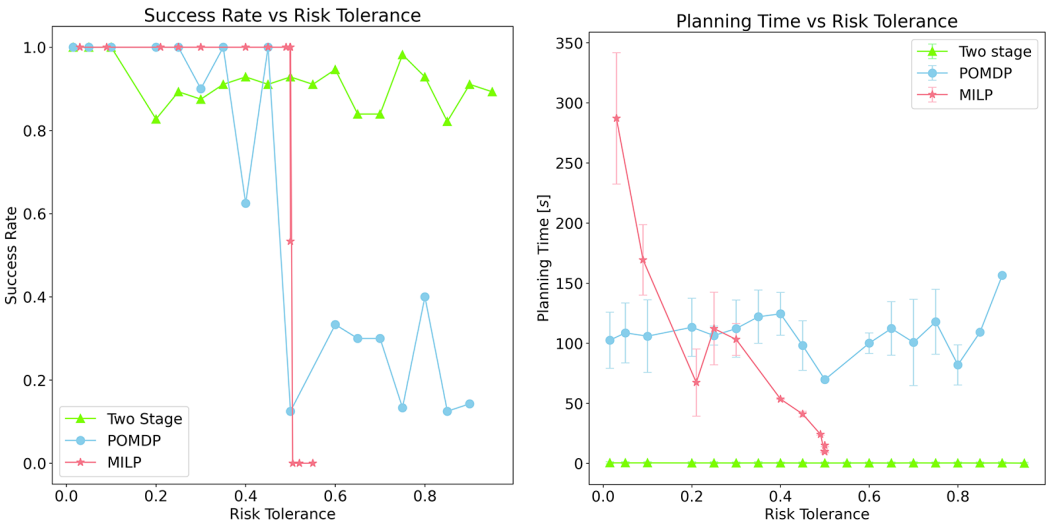
$$t_P = \sum_{i=1}^{N} t_i^p \tag{5.15}$$

5.5 Monte Carlo experiment results discussion

It is important to highlight that there is a fundamental difference between MILP / POMDP and Two-Stage planners that makes the comparison nuanced. The MILP and Two Stage planners output what they believe is the optimal *Plan*, from the initial state to the goal state. The computational cost of determining this plan is fixed in the sense that it is not possible to trade plan optimality for a computational speed-up. The plan is either found, or no solution is available. On the other hand, the POMDP planner uses an anytime solver that will be capable of outputting sub-optimal plans with an arbitrarily small computational cost. For the POMDP planner, increasing the computational budget allocated to searching the solution space will improve solution quality at the expense of a slower planning process. Hence, the only accurate way to compare the planner's performance against each other is to look at performance trends with varying computational resources.

All computational experiments were run on an Intel i7 NUC with a 2.65 GHz processor, and 32-gigabyte memory.

The focus is first on determining the effects of varying *Risk Tolerance* on the planner's results, then on performance metrics.



(a) Success rate vs risk tolerance

(b) Plan time vs risk tolerance

Fig. 5.6 Planner performance comparisons with varying risk tolerances.

These planner's characteristics with varying risk tolerance are visible in Figure 5.6. In these comparisons, the map has been kept constant and corresponds to the One-Obstacle map (Map-1).

In Figure 5.6b, it is visible how risk tolerance has a large effect on the MILP's planner performance, as it modifies the space of feasible solutions. With lower risk tolerances, the MILP planner struggles to search through the solution space, and the time needed to find the optimal plan increases dramatically to several minutes, also with a low number of obstacles. Conversely, risk tolerance can be tuned to find solution times of a few seconds. As expected, POMDP and Two-Stage planner computational costs remain roughly constant with varying risk tolerance, as their underlying solvers are not affected by varying the relative size of feasible vs infeasible solution spaces.

In Figure 5.6a it can be seen how the POMPD planner acts as expected with its plan execution success rate decreasing with increasing risk-tolerance. The Two-Stage planner also shows decreasing success rates, but the trend reaches a lower limit due to the fact that the path planning stage is not coupled with the task scheduling. In fact, given a fixed path, there is an upper bound to the path's risk, given by scheduling only move actions ($\sum_{k=1}^{N} {}^{S}y_k = 0$). In the MILP planner's success rates, the conservative nature of the covariance growth model is clearly visible. Note that the true uncertainty growth for a process where zero-mean Gaussian noise is a

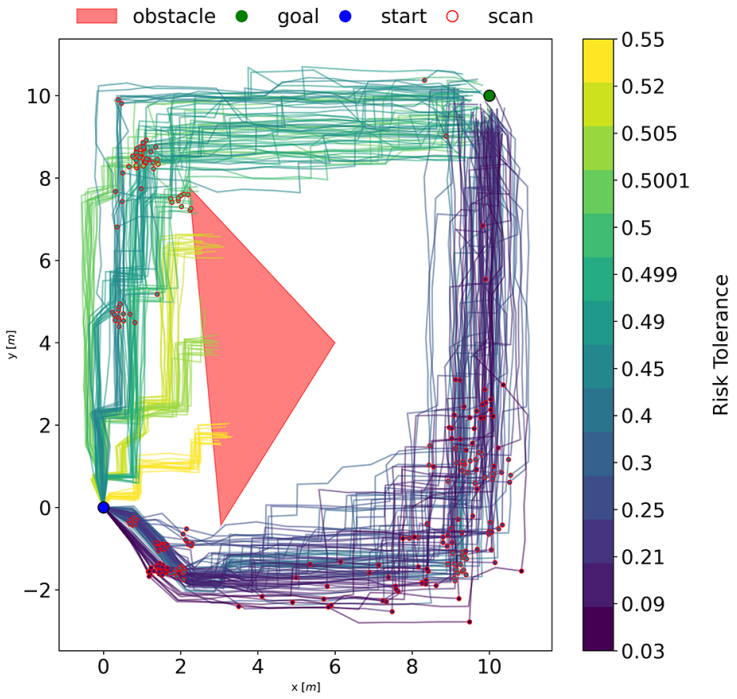
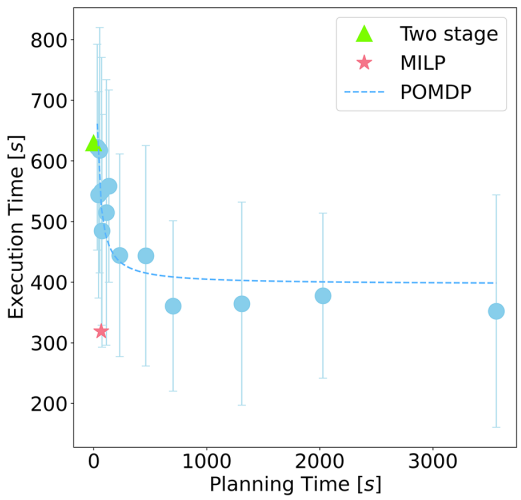


Fig. 5.7 Representation of different Monte Carlo runs of the MILP planner on a triangular obstacle map. Each set of runs has a unique risk tolerance, and it can be seen how high risk tolerance leads to trajectories closer to the obstacle and fewer scans, whereas low risk tolerance leads to an increase in number of scans and more conservative trajectories.

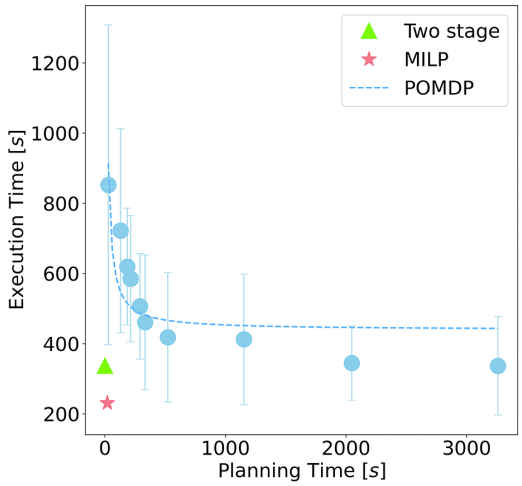
linear *Variance* growth, thus a square root *Covariance* growth. Covariance growth is modeled a *Quadratic*, therefore leading to conservative plans. The success rates of these plans is 100% for risk tolerances up to almost 50%. As covariance growth approaches 50%, the state belief's mean is gradually steered closer and closer to obstacles, and once the tolerance goes beyond 50%, the planner finds solutions where the mean passes *through* the obstacle (Clearly visible in Figure 5.7). For these risk tolerances above 50%, the planner's overestimation of state uncertainty clearly backfires, as the likelihood of colliding with obstacles jumps to 100%. This is not necessarily an issue, as any real-world application would never consider such risk tolerances.

Risk tolerance is a major decision for every mission, and is one of the primary drivers of development, manufacturing and testing costs. Flagship mission, such as that envisioned for EELS, would warrant risk tolerances less than 0.01 (probability of mission success greater than 99%), and a lot of the mission's hardware would have to be designed to be redundant and capable of gracefully degrading. The system level autonomy's risk tolerance should be aligned with the mission's risk posture, but should not be unnecessarily more conservative. The reason for this is that the planner's risk posture has an effect on the plan's cost. Excessively conservative risk tolerances could potentially lead to planning failures when the system is traversing inherently dangerous terrain (no suitable solution satisfies risk constraints), or would lead to unnecessary depletion of mission resources.

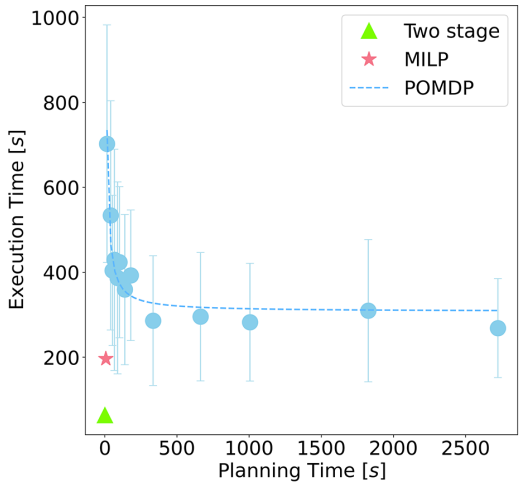
In Figure 5.8,the performance comparison for the three planners with a risk tolerance of 30%, over the three test maps can be seen. As mentioned earlier, MILP and Two-Stage are represented by single points, while the POMDP is a line. To interpret these performance maps, it is worth reminding that a high-quality solution will have low execution time, and vice versa. For all maps, the POMDP planner shows an increase in solution quality as planning time is allowed to increase. The POMDP's stochasticity is visible by the large error bars. A key aspect to note is that, on average the MILP planner finds a higher-quality solution than the POMDP for a fixed computational budget. Another way of seeing this is that by *fixing the solution cost*, the POMDP planner will take *orders of magnitude* longer to find plans of comparable quality to the MILP planner reliably. The comparison between MILP and Two-Stage planners is less straightforward as it is highly map-dependent. The Two-Stage planner will find low-quality solutions when the map has a minimal-length path that runs close to obstacle boundaries for a significant fraction of its



(a) One-Obstacle Map



(b) Three-Obstacles Map

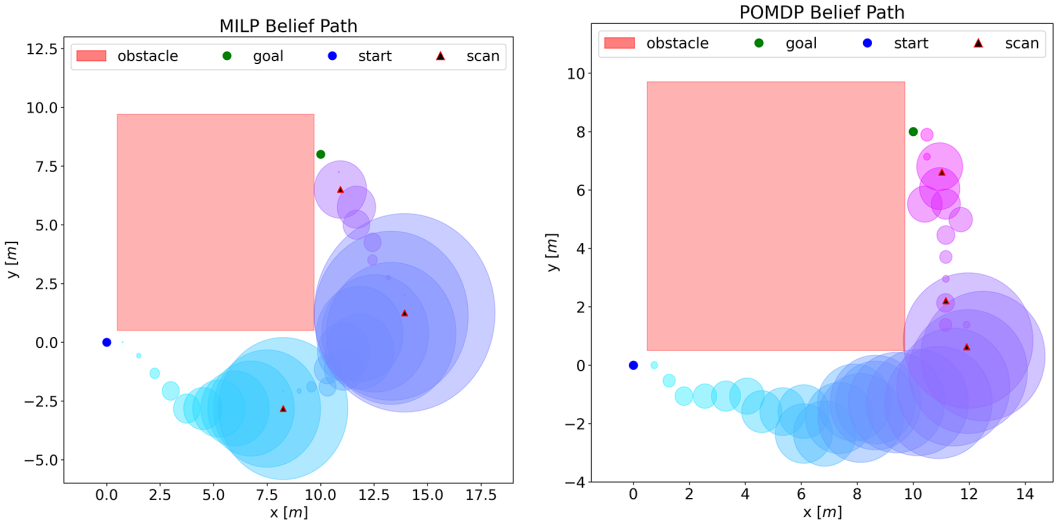


(c) Two-Obstacles Map

Fig. 5.8 Planner performance comparison in three different environments. The x-axis represents planning time (computational cost), whereas the y axis represents execution time (plan cost).

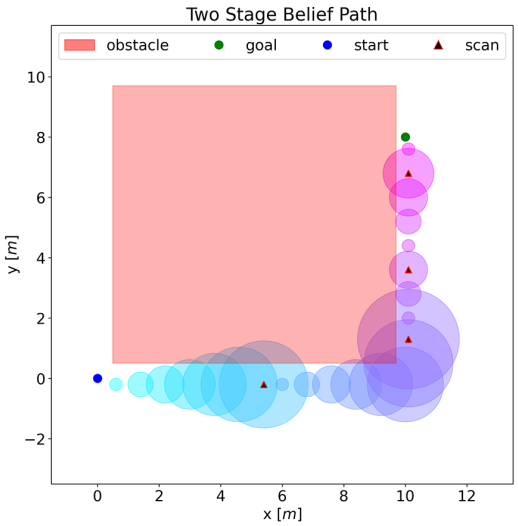
duration. Conversely, terrains with few obstacles will favor the Two-Stage planner, as scans will be scheduled only when in close proximity to obstacles. Note that this relationship is rooted in the MILP planner's modeling of uncertainty growth as quadratic, which yields plan sub-optimality.

The different nature of the three planners can be observed in Figure 5.9, where plans are visualized in belief space. Figure 5.9a shows the MILP planner's output, and it is clear that the planner is finding an optimal solution, but uncertainty growth



(a) MILP

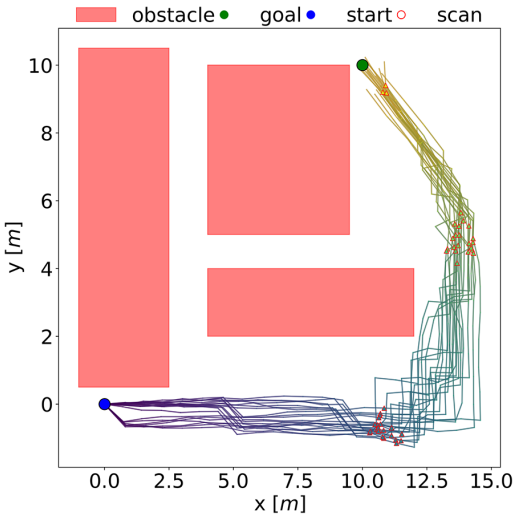
(b) POMDP



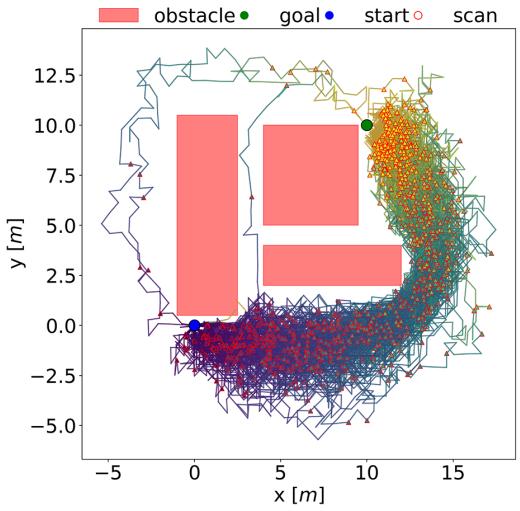
(c) Two-Stage

Fig. 5.9 Comparison of the belief-space plan generated by the three planners on the one-obstacle map. State uncertainty is represented as shaded circles along the planned path, wherars scan behaviors are black triangles with a red outline. A colormap from light blue to magenta displays time-progression along the plan.

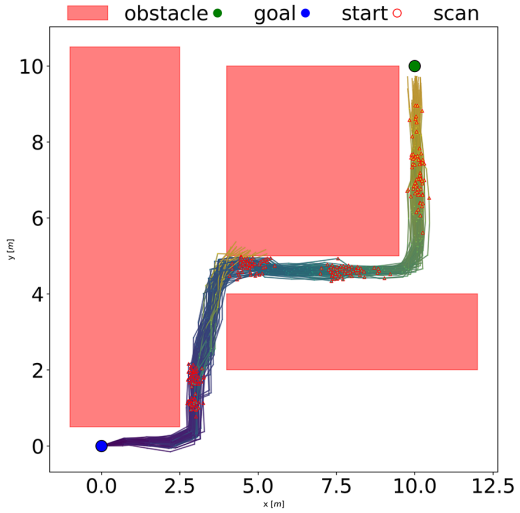
(the growth rate of the shaded area) is overestiamted. Conversely, Figure 5.9 shows how the decoupled approach schedules scans by placing a threshold on the area of intersection between state uncertainty and obstacles. In Figure 5.9b, the POMDP's stochastic execution is once again visible.



(a) MILP



(b) POMDP



(c) Two-Stage

Fig. 5.10 Comparison of trajectories generated by Monte Carlo simulations of the three planners running on the three-obstacle map. Scan behaviors along the trajectory are displayed as red outlined triangles. The progression of time along the trajectory is shown as a colormap going from dark blue (start) to dark yellow (end).

Figure 5.10 shows the trajectories of multiple Monte Carlo runs on Map-3. The Two-Stage planner is tricked into heading into a narrow passage and having to schedule multiple unnecessary scans (Figure 5.10c). The MILP planner avoids the narrow passageway and consistently finds a high-quality path to the goal (Figure 5.10a). The POMPD planner exhibits highly stochastic execution characteristics,

with two runs rarely similar. In general, it follows a path similar to the MILP planner, but a number of stops and specifics of the path are stochastic.

MILP and POMDP planners suffer from scalability issues regarding the number of *planning horizon* and *number of actions*. Mixed Integer Programs are NP-hard problems. Thus, the solution time scales exponentially with the problem's input size. Similar scalability concerns are present for exact POMDP solutions. Yet, the existence of anytime, approximate solution algorithms enables finding solutions at any computational cost - albeit by sacrificing (at times significantly) optimality guarantees.

Tests were performed with planning horizons up to 50 time steps and a number of obstacles less than three obstacles $N^{obs} \leq 3$. Depending on the specifics of risk tolerance and map difficulty, the MILP planner always reached unacceptable solution times before 100.

It is interesting to note the dependence on map difficulty. A primary source of intractability in the MILP planner is decisions of the form "should I go left or right of the obstacle", where either way could be equally appealing. These binary decisions increase the size of the relaxation bounds used by MILP solvers and force the solver to search through a more significant portion of the solution space, thus decreasing the effectiveness of branch-and-bound search techniques.

By formulating the problem as a search through a graph of convex, free-space sets, it has been demonstrated that relaxation bounds can be tightened and the scalability of MILP path planning can be improved [117]. Although their formulations are not directly applicable to our problem, as they are not considering belief-space planning.

The scalability issues of the Mixed Integer Linear Programming (MILP) planner can also be examined through the lens of binary decision variables. As binary variables increase, the problem becomes progressively harder to solve. It is then fully expected that increasing the **number of actions** leads very quickly to computational intractability. Scalability concerns emerged even with a short horizon of less than 50 time steps and a number of actions $N_a \leq 2$. The POMDP formulation is also sensitive to the number of actions, as additional actions further increase the problem's branching factor.

5.6 Hardware integration of MILP planner

The previous sections have outlined how the MILP planner is a better choice than the POMDP planner because of the POMDP's computational cost that turned out to be orders of magnitude larger than the MILP's cost at a given solution quality. Furthermore, the MILP planner proved to be more capable of navigating environments with multiple obstacles than the Two stage planner, because of the MILP's capacity to jointly reason about path and actions. Because of these considerations, the MILP planner was integrated into the robot and tested to understand the limitations of a real-world deployment. This planner can be called when the robot has detected localization failure due to bad measurements without sensor damage. As can be seen in Figure 5.11, several scanning behaviors were implemented, varying by the number of modules involved in the behavior. Due to the robot's high reconfigurability, what information-gaining action to take is part of its action space. By increasing the number of modules involved in the scanning behavior, the robot will achieve a better view, and relocalize better. But, increasing the number of modules will also increase the behavior's time and power costs. For simplicity, the planning problem was solved considering only a single scanning behavior, leaving a multi-scan action space for future work. Specifically, the behavior involving two robot modules on the top right corner of Figure 5.11 is used as a reference.

Figure 5.12, was produced to visualize the impact of scanning on the perception system. The LiDAR's point cloud is shown before and during a scanning behavior and it is immediately clear how scans improve perception capability - especially in cluttered environments with obstacles close by. It is important to acknowledge that the visualization presented in Figure 5.12 is taken in an indoor laboratory environment, so the wide-angle LiDAR mounted on the robot picks up features from the room's walls and ceiling in both scanning and non-scanning configurations. The situation is very different during surface locomotion on the surface of a glacier, where the lack of nearby features leads to degraded navigation performance.

The MILP planner consumes a goal, an initial state, and a set of obstacles described through linear inequality constraints and generates an intermediate waypoint where the robot should scan next.

The goal is expressed as a constraint over state in terms of pose, and a maximum risk level tolerance.

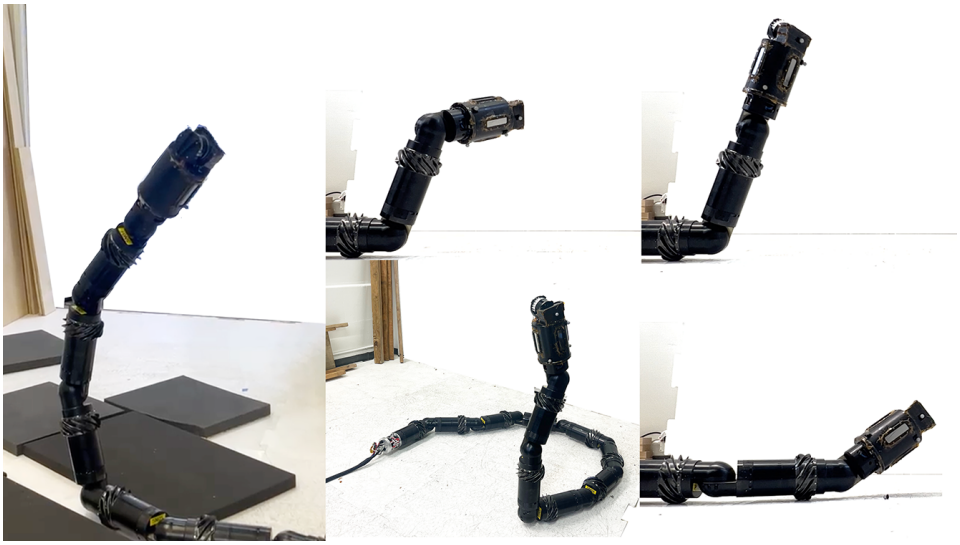


Fig. 5.11 Images of scanning behaviors running on the EELS hardware platform. Several behaviors can be selected depending on the available support polygon, resource and observation needs.

```
MILP_goal:
  pose: [x,y,z]
  risk_bound: 0.1
```

A key assumption for this planner is that the robot is not subjected to non-holonomic constraints. If the system on which the planner is deployed is non-holonomic, this risk-aware planner can still be used, so long as it is planning over a spatial scale much larger than that where the non-holonomic constraint matter. Therefore, the MILP planner is better suited for medium/long-range planning, while planners capable of capturing non-holonomic constraints such as EELS' lower-level motion planner are ideally suited for shorter-range operations.

Another assumption that was made in the planner's development is that perception is not capable of localizing the robot while moving and is capable of localizing the robot when stopping for an information-gaining action. This assumption could not directly be tested on hardware, as the perception was reliably working in the lab environment due to large features being visible from any pose (walls and ceiling). Hence, the perception failure and relocalization assumption was *mocked*. With the robot being asked to execute movements in an open-loop manner, disregarding pose estimates from perception. Pose information is fed to the robot only when planning after a scanning behavior.

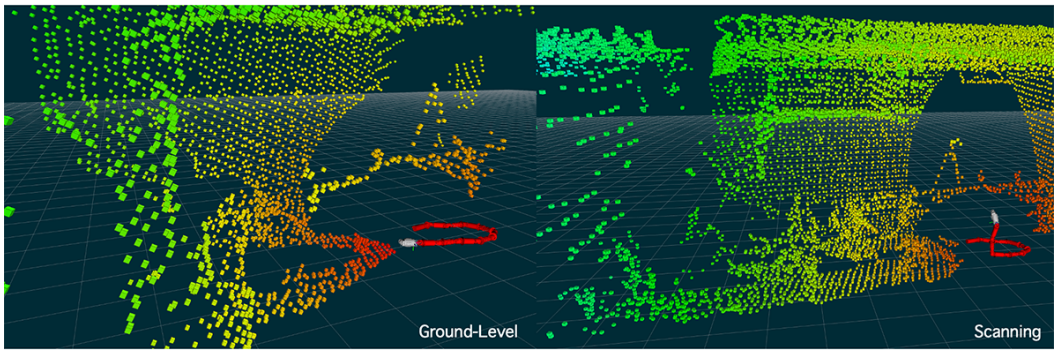


Fig. 5.12 Comparison between ground-level locomotion and Scanning behavior in an indoor laboratory environment. The ground level view is highly constrained and observes only the immediate surroundings, wheras the scanning behavior observes features at a much greater distance from the robot.

5.6.1 From perception to polygonal obstacles

The MILP planner requires convex obstacle representations as a set of line equation coefficients. A large number of perception pipelines (including EELS) generate pose estimates and traversability maps. For surface mobility, the type of map used is a 2.5-dimensional grid-map, that can be interpreted as very similar to an image.

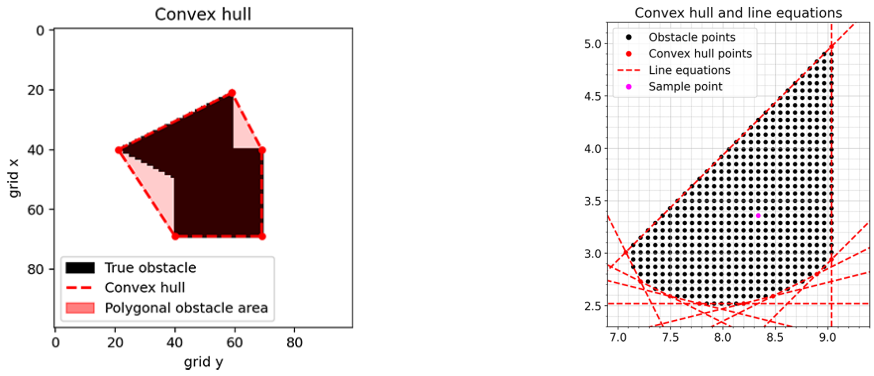
A Grid-map to line-coefficients interface was implemented, by customizing a third party library. It is interesting to trace the process of building this capability step-by-step to illustrate its core concepts. The starting point is a single obstacle, followed by a focus on how the single obstacle case can be extended to an arbitrary number of obstacles.

The first step is to take the Gridmap and transform it into a set of obstacle points. This can be easily done by applying a threshold Δ to the map's traversability estimate layer, and treating any value above the threshold as impassible obstacle:

$$X^{\text{obs}} = \{(x,y) : \ m(x,y) > \Delta\} \tag{5.16}$$

Once the set of obstacle coordinates is computed, there is a large number of efficient algorithms that compute the set's convex hull. An example could be Graham's scan [118] or Quickhull [119]. Implementation details are out of scope, but,

as shown in Figure 5.13a, convex hull algorithms efficiently find the corner points of the smallest convex shape that can be fitted around the obstacle.



(a) Convex Hull (b) Granular representation

Fig. 5.13 Examples of the Quickhull algorithm converting concave obstacles into convex polygonal obstacles.

Convex hull algorithms return an ordered set of points that represent the hull, starting from the corner with $x_{\min}, y_{\min}$ and proceeding either clockwise or counter-clockwise. To make the hull readable by the MILP planner, these points need to be used to derive line coefficients. The first step take all adjacent pairs of points and compute the equation of the line that connects these two points in the form $a_1 x_1 + a_2 a_2 = b$. This is easily done by using the equation of a line passing by two known points:

$$\frac{x_2 - {}^k x_2}{{}^q x_2 - {}^k x_2} = \frac{x_1 - {}^k x_1}{{}^q x_1 - {}^k x_1} \tag{5.17}$$

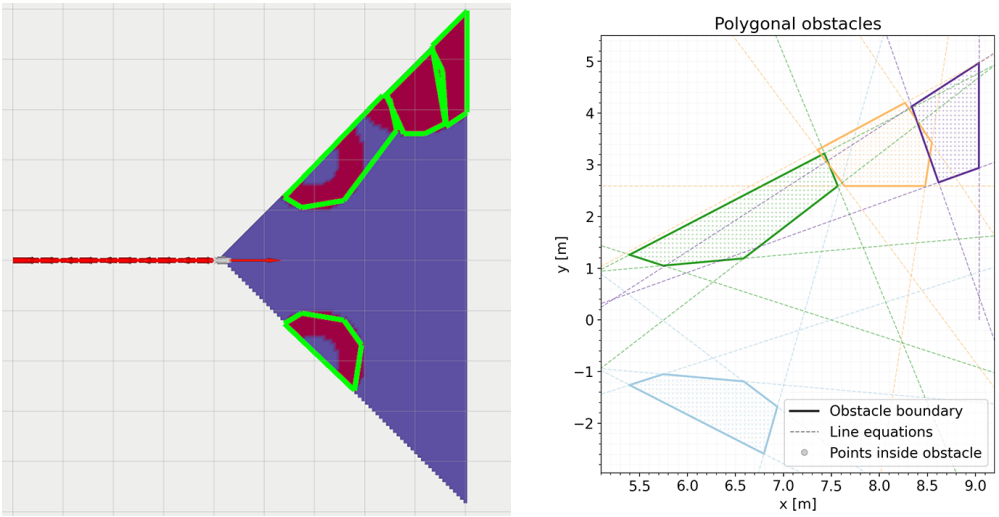
$$\left(x_2 - {}^k x_2\right)\left({}^1 x_1 - {}^k x_1\right) = \left(x_1 - {}^k x_1\right)\left({}^q x_2 - {}^k x_2\right) \tag{5.18}$$

$$x_1 \underbrace{\left({}^k x_2 - {}^1 x_2\right)}_{a_1} + x_2 \underbrace{\left({}^q x_1 - {}^k x_1\right)}_{a_2} = \underbrace{{}^q x_1 \, {}^k x_1 - {}^k x_1 \, {}^q x_2}_{b} \tag{5.19}$$

In the general case, multiple obstacles might be present, and thus there needs to be a preliminary clustering step that identifies groups of adjacent obstacles points that represent an obstacle. A number of clustering effective clustering algorithms have been developed over the years, such as k-means [120] or dbscan [121]. Once clusters of data are generated, the convex algorithm is run over each obstacle individually, and a set of sets of obstacle line coefficients can be fed to the MILP planner.

For clustering and convex hull point computation, the open-source library costmap_converter was used due to its ease of integration with the robot's ROS-1 software system.

An example of the full obstacle processing pipeline, applied to a perception software mock can be seen in Figure 5.14. Here, the Gridmap to convex-hull step is displayed on the left side (5.14a) as a top-down rviz view. The robot's head is displayed at the center of the image. The map is made from blue (safe) or red (unsafe) cells, and the green segments connect the markers that make up the set of convex obstacles. The cost map conversion library conveniently allows modifying in real-time both clustering and convex hull parameters, allowing tuning the obstacle's number, size, and the "resolution" of each obstacle (minimum distance between two convex hull points).



(a) Gridmap to convex hull

(b) convex hull to lines

Fig. 5.14 Conversion between grid-map to set of convex obstacles described by line inequalities. Figure 5.14a shows the output of a mock perception pipeline. The EELS robot is displayed to the center-left and the mock output is the shaded triangular area on the right side of the image. Blue cells is obstacle-free, whereas red cells are obstacles. The green lines are the output of the output of the convex hull algorithm. Figure 5.14b visualizes the line inequalities that make up the obstacle set.

In Figure, 5.14b the obstacle hull markers are converted into line inequalities, and the obstacles - as seen by the MILP planner - are displayed.

There is a tradeoff between obstacle resolution and planner performance, as each additional line/obstacle will increase planning time as it adds inequalities.

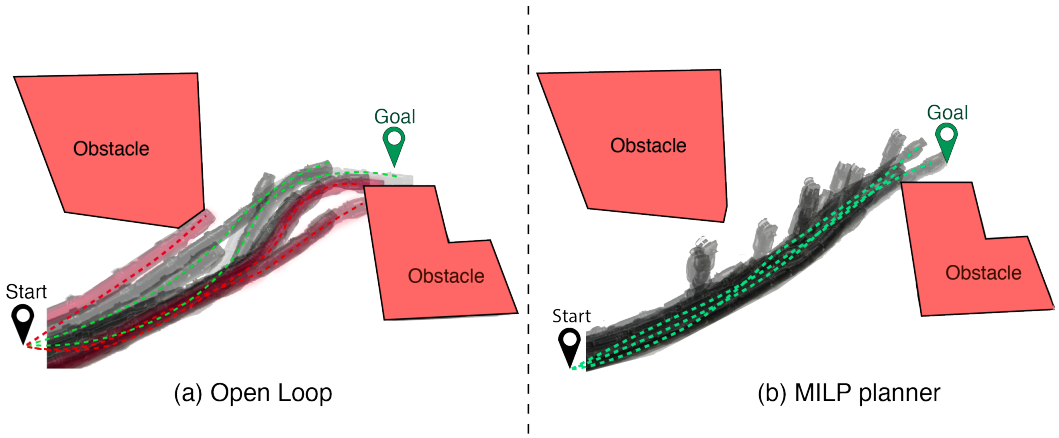
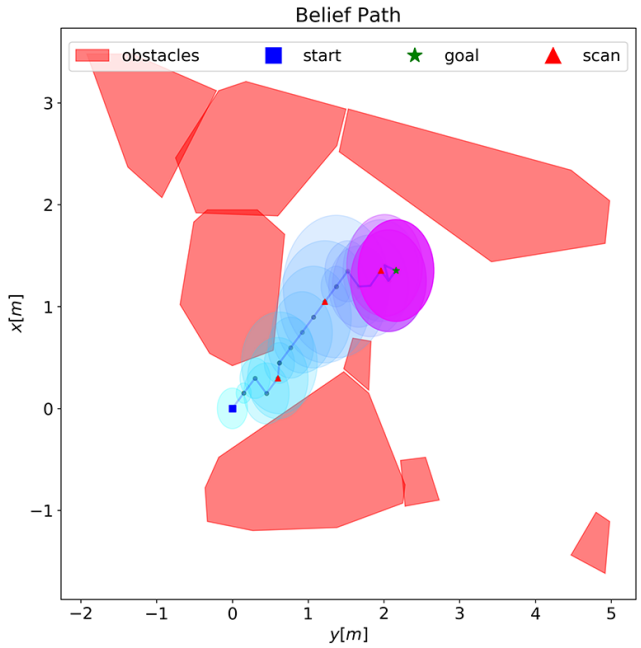


Fig. 5.15 Comparison between open loop behavior and MILP planner running on hardware. Different runs are represented as dashed lines. Runs that have successfully reached the goal are shaded green, whereas runs that resulted in collision with an obstacle are shaded red. The background and obstacles are simplified to improve image readability. Figure (a) shows open loop behavior and shows 3 unsuccessful runs and a 2 succesfull runs, whearas (b) shows 3 succesful runs.

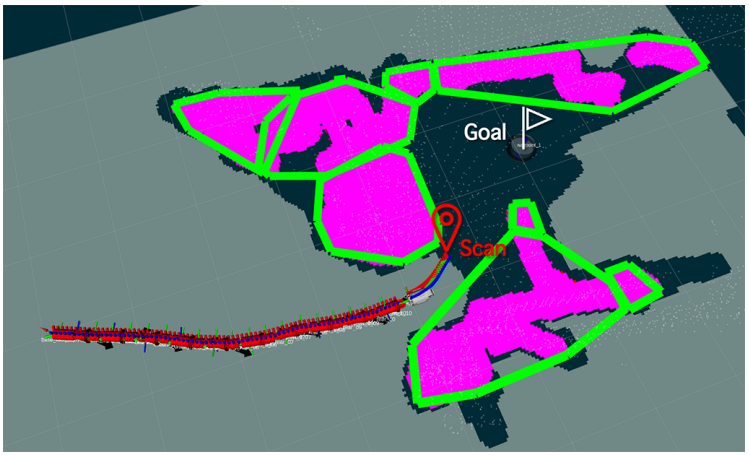
Moreover, the optimal value of the parameter set is environment dependent as open environments with fewer obstacles can afford a polygon with many sides for each obstacle, while more complex environments will require a decrease in obstacle resolution or a grouping of adjacent obstacles.

The MILP planner was tested over multiple runs and compared against open-loop movement execution. The different runs have been overlaid in Figure 5.15, where it is clearly visible how approaching localization failures with open-loop behavior leads to multiple collisions. On the other hand, planning risk-aware trajectories and scheduling information-gaining stops, combined with re-planning leads to a much safer outcome. This is particularly true in obstacle rich environments, where pose uncertainty growth quickly leads to collisions.

It is worth acknowledging that these experiments were conducted in a workspace that is not large enough to ignore the robot's non-holonomic constraints. Thus, the path taken to reach scan waypoints can differ from the expected path over which belief was projected. This is a source of errors, but was deemed acceptable as these results are intended more as a qualitative display of risk-aware planning rather than a quantitative analysis of its performance.



(a) Belief-space plan



(b) RVIZ view.

Fig. 5.16 Software views of the MILP planner running on hardware in a lab environment. 5.16a shows the belief-space representation of the generated plan. 5.16b is an RVIZ view the part of the plan that is dispatched for execution to the mission executive

When the MILP planner is run in the constrained lab environment, multiple scans are scheduled through the robot's mission. As can be seen in Figure 5.16a, these scan points are scheduled far from the obstacles' boundaries, as trajectories that minimize the number of scans tend to be less costly than trajectories with more scans.

It can also be said that Figure 5.16a shows the internal representation of the MILP planner. Moving one step closer to hardware, Figure 5.16b shows the MILP planner's output viewer from the rest of the software stack. This rviz view shows how the goal of reaching a point in space was broken down into intermediate scanning waypoints that were reached thanks to the lower level planners and controllers.

5.6.2 Limitations of the hardware experiments

These hardware experiments are only intended to be a proof of concept of the integration between the MILP planner, the system level autonomy framework outlined in section 2.4, and the EELS hardware platform. Detailed data logging was not set up, and quantitative experiments were explicitly left for future work. The primary reason for this is the author's involvement in EELS' field campaign [122], and the consequent requirement to focus on lower level control to achieve subsurface mobility. The EELS platform required a considerable research and development effort in the locomotion layer because of its novel surface and subsurface mobility strategies. A detailed experimental analysis of the mission planning layer, and a unification between surface and subsurface planners was left for after the first field campaign, where the primary test was for the system's locomotion capabilities. The author left the EELS project soon after the field campaign.

As a consequence, no detailed analysis and design went into accommodating the MILP planner in the system's computational architecture. As a way to minimize the MILP solver's impact on the rest of the software stack, the planner process was fully contained on the machine that was used to host the computational experiments outlined in the previous sections. This machine was connected to the EELS network, and planning requests and responses were implemented as network calls. Therefore, the solver's computational performance between hardware and non-hardware experiments remained unchanged. This is in line with the initial testing philosophy applied to EELS, where the purpose of tests was to demonstrate the feasibility and functionality of the algorithms, without focusing on algorithm optimization. Future

work will be centered around refining the portability of these algorithms, paying attention to compute resource usage.

Chapter 6

Conclusions and future work

In conclusion, it is worth reviewing the primary results and lessons learned through this work and outlining future promising research directions.

This research began by outlining the capabilities needed to operate autonomously in extreme, uncertain environments and argues in favor of including elements of motion planning and risk awareness in system-level autonomy. Subsequently, the EELS system was described alongside a robotic software architecture capable of supporting high-level autonomy functionality. The architecture presented focuses on modularity and proved well-suited for fast-paced research projects where planning requirements often change. However, the main purpose of this architecture was to provide a minimal scaffolding needed to run and test the main contributions of this work, which are the decision making under uncertainty algorithms developed for the EELS surface locomotion problem. Therefore, the architecture description does not outline in detail the inclusion of risk and fault management components, and is not evaluated quantitatively in the experimental section. In the development of decision-making algorithm, emphasis was placed on task and motion planning under uncertainty and approached through three families of planners.

- A classical two-stage approach that decouples motion from task planning and solves them separately in stages.
- A planner based on a Partially Observable Markov Decision Process (POMDP) formulation coupled with an MCTS solver and several different reward-shaping strategies.

- An entirely novel planner based on a Mixed Integer Linear Program (MILP) formulation capable of jointly optimizing task and motion planning and providing guarantees on solution existence and quality.

These planners were quantitatively compared against each other in a computational framework, and the MILP planner was implemented on hardware and tested in a laboratory setting because of its promising characteristics. The main limitation of these planners is scalability. Both the POMDP and MILP planners exhibited the capacity to find good solutions, but with increases in planning horizon, action space size, and number of obstacles, these belief-space planners fail scale due to the fundamentally NP-hard nature of the problem they are trying to solve. An inclusion of true risk-awareness - defined as the capacity to reason jointly about probability of failures, and the consequences of these failures - is left for future work, as the planners treated in this work reason with a fixed consequence level.

An immediate direction of research for future work is seeking ways to improve these algorithm's scalability in terms of the number of obstacles and actions they can handle. Belief-space planning is very challenging and generally scales poorly, but quality solutions can be found by sacrificing optimality guarantees. Perhaps, there is a broader challenge of cultural nature at play here. The aerospace industry often requires algorithms with provable mathematical guarantees of safety and performance, which disqualifies a large set of algorithms that could nevertheless be good enough, and have much improved scalability. The question of how build trust and verify algorithms that do not come with mathematical guarantees is likely going to be central to the future of system-level autonomy research.

A particularly promising technical research direction, that does not require significant modifications to the MILP algorithm, is seeding the planner with a feasible solution generated with a faster algorithm, and terminating the search early. This strategy would forsake optimality guarantees, but could lead to both high-quality solutions and low computational cost. Additional ways of dealing with the scalability issues introduces by belief-space planning could be (a) remove belief space planning in favor of integrating Task and Motion Planning into system-level autonomy and deal with uncertainty implicitly through the planner's constraints, or (b) explore data-driven approaches. Achieving safe behavior when neglecting beliefs could be achieved similarly to the two-stage approach, which could be vastly improved with iterative planning techniques. Another interesting research direction is to further

improve the proposed robotics software architecture to include fault management and mesh reactive system-level behaviors with deliberative planning. Ensuring that the deliberative planning is aware of localize reactive behaviors is another interesting architectural challenge. Additionally, system-level reactive behaviors might be necessary to recover from certain failures, and there is an opportunity to design a unified architecture for high-level autonomy capable of both reactive behaviors and deliberative planning. It is also necessary to work on finding ways to formulate risk-aware planners capable of assessing consequences and finding scalable ways of solving these formulations. A work focused more on architectural changes would benefit from quantitative experiments analyzing the effect on mission-level performance metrics of architectural changes.

True system-level autonomy will likely only be achieved through artificial general intelligence (AGI), whose future prospects are fundamentally unknowable. However, even in the absence of AGI, there is still a lot of value that can be found in furthering the state of the art of system-level autonomy. Many decision-making technologies have proven their value in Operations Research, Field Robotics, and Space Exploration, and their usefulness is only bound to grow as autonomous systems become more widespread.

With an appropriate set of simplifying assumptions, and incremental technological improvements, planetary subglacial access could soon be within humanity's reach - if a political choice is made to prioritize it.

References

[1] Edwin S Kite and Allan M Rubin. Sustained eruptions on enceladus explained by turbulent dissipation in tiger stripes. *Proceedings of the National Academy of Sciences*, 113(15):3972–3975, 2016.

[2] Richmond Kristof, Hogan Bartholomew, Lopez Alberto, Harman John, Myers Krista, Guerrero Veronica, Lanford Ephraim, Ralston James, Tanner Neal, Siegel Victoria, et al. Prometheus: Progress toward an integrated cryobot for ocean world access. *Authorea Preprints*, 2022.

[3] Steven Oleson, J Michael Newman, Andrew Dombard, D'Arcy Meyer-Dombard, Kate Craft, James Sterbentz, Anthony Colozza, Brent Faller, James Fittje, John Gyekenyesi, et al. Compass final report: Europa tunnelbot. Technical report, 2019.

[4] Kris Zacny, Juergen Mueller, Tighe Costa, Tom Cwik, Andrew Gray, Wayne Zimmerman, Paul Chow, Fredrik Rehnmark, and Grayson Adams. Slush: Europa hybrid deep drill. In *2018 IEEE Aerospace Conference*, pages 1–14. IEEE, 2018.

[5] Bernd Dachwald, Jill Mikucki, Slawek Tulaczyk, Ilya Digel, Clemens Espe, Marco Feldmann, Gero Francke, Julia Kowalski, and Changsheng Xu. Icemole: a maneuverable probe for clean in situ analysis and sampling of subsurface ice and subglacial aquatic ecosystems. *Annals of Glaciology*, 55(65):14–22, 2014.

[6] Jonathan I Lunine. Ocean worlds exploration. *Acta Astronautica*, 131:123–130, 2017.

[7] E Camprubí, JW De Leeuw, CH House, F Raulin, MJ Russell, Anja Spang, MR Tirumalai, and F Westall. The emergence of life. *Space Science Reviews*, 215:1–53, 2019.

[8] Kevin Peter Hand and Christopher R German. Exploring ocean worlds on earth and beyond. *Nature Geoscience*, 11(1):2–4, 2018.

[9] Robert T Pappalardo, Michael JS Belton, HH Breneman, MH Carr, Clark R Chapman, GC Collins, T Denk, S Fagents, Paul E Geissler, B Giese, et al. Does europa have a subsurface ocean? evaluation of the geological evidence. *Journal of Geophysical Research: Planets*, 104(E10):24015–24055, 1999.

[10] Ronald Greeley, Robert Sullivan, James Klemaszewski, Kim Homan, James W Head III, Robert T Pappalardo, Joseph Veverka, Beth E Clark, Torrence V Johnson, Kenneth P Klaasen, et al. Europa: initial galileo geological observations. *Icarus*, 135(1):4–24, 1998.

[11] Cynthia B Phillips and Robert T Pappalardo. Europa clipper mission concept: Exploring jupiter's ocean moon. *Eos, Transactions American Geophysical Union*, 95(20):165–167, 2014.

[12] L Esposito, AIF Stewart, J Colwell, A Hendrix, W Pryor, D Shemansky, and R West. Enceladus' water vapor plume. *Science*, 311(5766):1422–1425, 2006.

[13] Luciano Iess, DJ Stevenson, M Parisi, D Hemingway, RA Jacobson, JI Lunine, F Nimmo, JW Armstrong, SW Asmar, M Ducci, et al. The gravity field and interior structure of enceladus. *Science*, 344(6179):78–80, 2014.

[14] Carolyn C Porco, Paul Helfenstein, PC Thomas, AP Ingersoll, J Wisdom, Robert West, Gerhard Neukum, Tilmann Denk, Roland Wagner, Thomas Roatsch, et al. Cassini observes the active south pole of enceladus. *science*, 311(5766):1393–1401, 2006.

[15] PC Thomas, R Tajeddine, MS Tiscareno, JA Burns, J Joseph, TJ Loredo, P Helfenstein, and C Porco. Enceladus's measured physical libration requires a global subsurface ocean. *Icarus*, 264:37–47, 2016.

[16] William B McKinnon. Effect of enceladus's rapid synchronous spin on interpretation of cassini gravity. *Geophysical Research Letters*, 42(7):2137–2143, 2015.

[17] Frank Postberg, Sascha Kempf, Jürgen Schmidt, N Brilliantov, Alexander Beinsen, Bernd Abel, Udo Buck, and Ralf Srama. Sodium salts in e-ring ice grains from an ocean below the surface of enceladus. *Nature*, 459(7250):1098–1101, 2009.

[18] J Hunter Waite, Christopher R Glein, Rebecca S Perryman, Ben D Teolis, Brian A Magee, Greg Miller, Jacob Grimes, Mark E Perry, Kelly E Miller, Alexis Bouquet, et al. Cassini finds molecular hydrogen in the enceladus plume: evidence for hydrothermal processes. *Science*, 356(6334):155–159, 2017.

[19] Frank Postberg, Yasuhito Sekine, Fabian Klenner, Christopher R Glein, Zenghui Zou, Bernd Abel, Kento Furuya, Jon K Hillier, Nozair Khawaja, Sascha Kempf, et al. Detection of phosphates originating from enceladus's ocean. *Nature*, 618(7965):489–493, 2023.

[20] GL Villanueva, HB Hammel, SN Milam, V Kofman, S Faggi, CR Glein, R Cartwright, L Roth, KP Hand, L Paganini, et al. Jwst molecular mapping and characterization of enceladus' water plume feeding its torus. *Nature Astronomy*, pages 1–7, 2023.

[21] KL Mitchell, M Ono, C Parcheta, and S Iacoponi. Dynamic pressure at enceladus' vents and implications for vent and conduit in-situ studies. In *48th Annual Lunar and Planetary Science Conference*, number 1964, page 2801, 2017.

[22] Miki Nakajima and Andrew P Ingersoll. Controlled boiling on enceladus. 1. model of the vapor-driven jets. *Icarus*, 272:309–318, 2016.

[23] Andrew P Ingersoll and Miki Nakajima. Controlled boiling on enceladus. 2. model of the liquid-filled cracks. *Icarus*, 272:319–326, 2016.

[24] National Academies of Sciences Engineering, Medicine, et al. Origins, worlds, and life: A decadal strategy for planetary science and astrobiology 2023-2032. 2022.

[25] Morgan L Cable, Carolyn Porco, Christopher R Glein, Christopher R German, Shannon M MacKenzie, Marc Neveu, Tori M Hoehler, Amy E Hofmann, Amanda R Hendrix, Jennifer Eigenbrode, et al. The science case for a return to enceladus. *The planetary science journal*, 2(4):132, 2021.

[26] Mark Chodas, Masahiro Ono, Jessica Weber, Laura Rodriguez, Michel D Ingham, Ben Hockman, Karl L Mitchell, Morgan L Cable, and Jason Rabinovitch. Enceladus vent explorer mission architecture trade study. In *2023 IEEE Aerospace Conference*, pages 1–16. IEEE, 2023.

[27] J. W. Weber, L. E. Rodriguez, M. Ono, M. Cable, M. Chodas, and M. Ingham. A general stm for the enceladus vent explorer mission concept and a case for vent exploration. In *54th Lunar and Planetary Science Conference 2023*, page LIV 1119, 2023.

[28] Kalind Carpenter, Morgan L Cable, Mathieu N Choukroun, Hiro Ono, Rohan A Thakker, Michel D Ingham, Patrick McGarey, Ansel Barchowsky, Saverio Iacoponi, Joseph J Bowkett, et al. Venture deep, the path of least resistance: Crevasse-based ocean access without the need to dig or drill. *Bulletin of the American Astronomical Society*, 53(4):356, 2021.

[29] Stephan Ulamec, Jens Biele, Oliver Funke, and Marc Engelhardt. Access to glacial and subglacial environments in the solar system by melting probe technology. *Life in Extreme Environments*, pages 1–24, 2007.

[30] Wayne Zimmerman, Robert Bonitz, and Jason Feldman. Cryobot: an ice penetrating robotic vehicle for mars and europa. In *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*, volume 1, pages 1–311. IEEE, 2001.

[31] Pavel G Talalay. *Thermal ice drilling technology*. Springer, 2019.

[32] M Brandt, W Zimmerman, D Berisford, J Mueller, M Barry, M Durka, R Kristof, B Hogan, and W Stone. Modeling of cryobot melting rates in cryogenic ice. In *2019 IEEE Aerospace Conference*, pages 1–17. IEEE, 2019.

[33] Scott Bryant. Ice-embedded transceivers for europa cryobot communications. In *Proceedings, IEEE Aerospace Conference*, volume 1, pages 1–356. IEEE, 2002.

[34] Guglielmo Daddi and Fernando Mier-Hicks. Thermal probe enhanced with pulsed plasma discharges for efficient ice penetration. *Journal of Thermophysics and Heat Transfer*, 36(1):81–88, 2022.

[35] Cameron Adkins, Mirza Akhter, Fernando Mier-Hicks, and David Staack. Pulsed plasma discharge effects on efficiency and rate of penetration of melt probes in ice. *Icarus*, 401:115600, 2023.

[36] William Stone, Bart Hogan, Vickie Siegel, John Harman, Chris Flesher, Evan Clark, Omkar Pradhan, Albin Gasiewski, Steve Howe, and Troy Howe. Project valkyrie: laser-powered cryobots and other methods for penetrating deep ice on ocean worlds. *Outer Solar System: Prospective Energy and Material Resources*, pages 47–165, 2018.

[37] William C Stone, Vickie Siegel, Kristof Richmond, and James Ralston. Deep cryogenic vacuum-ice testing of cryobots: Project prometheus. In *2023 IEEE Aerospace Conference*, pages 1–17. IEEE, 2023.

[38] Masahiro Ono, Karl Mitchel, Aaron Parness, Kalind Carpenter, Saverio Iacoponi, Ellie Simonson, Aaron Curtis, Mitch Ingham, Charles Budney, Tara Estlin, et al. Enceladus vent explorer concept. *Outer Solar System: Prospective Energy and Material Resources*, pages 665–717, 2018.

[39] Issa AD Nesnas, Lorraine M Fesq, and Richard A Volpe. Autonomy for space robots: Past, present, and future. *Current Robotics Reports*, 2(3):251–263, 2021.

[40] Joseph A Starek, Behçet Açıkmeşe, Issa A Nesnas, and Marco Pavone. Spacecraft autonomy challenges for next-generation space missions. *Advances in Control System Technology for Aerospace Applications*, pages 1–48, 2016.

[41] Andrew E Johnson, Yang Cheng, James F Montgomery, Nikolas Trawny, Brent Tweddle, and Jason X Zheng. Real-time terrain relative navigation test results from a relevant environment for mars landing. In *AIAA Guidance, Navigation, and Control Conference*, page 0851, 2015.

[42] Daniel Dvorak, Robert Rasmussen, Glenn Reeves, and Allan Sacks. Software architecture themes in jpl's mission data system. In *2000 IEEE Aerospace Conference. Proceedings (Cat. No. 00TH8484)*, volume 7, pages 259–268. IEEE, 2000.

[43] TA Estlin, Rich Volpe, Issa Nesnas, Darren Mutz, Forest Fisher, Barbara Engelhardt, and Steve Chien. Decision-making in a robotic architecture for autonomy. 2001.

[44] Rashied Amini, Lorraine Fesq, Ryan Mackey, Faiz Mirza, Robert Rasmussen, Martina Troesch, and Ksenia Kolcio. Fresco: A framework for spacecraft systems autonomy. In *2021 IEEE Aerospace Conference (50100)*, pages 1–18. IEEE, 2021.

[45] Jorge Ocón, Francisco Colmenero, Joaquín Estremera, Karl Buckley, Mercedes Alonso, Enrique Heredia, Javier Garcia, Andrew Ian Coles, Amanda Jane Coles, Moises Martinez Munoz, et al. The ergo framework and its use in planetary/orbital scenarios. In *Proceedings of the 69th International Astronautical Congress (IAC)*, 2018.

[46] Malik Ghallab, Félix Ingrand, Solange Lemai, and Frédéric Py. Architecture and tools for autonomy in space. *ISAIRAS, Montreal*, 2001.

[47] Eric M Timmons, Marlyse Reeves, Benjamin J Ayton, Brian Williams, Michel D Ingham, Julie Castillo-Rogez, William Seto, Klaus Havelund, Ashkan Jasour, Benjamin Donitz, et al. Information-driven and risk-bounded autonomy for scientist avatars. In *ASCEND 2021*, page 4023. 2021.

[48] Douglas E Bernard, Gregory A Dorais, Chuck Fry, Edward B Gamble, Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, P Pandurang Nayak, Barney Pell, et al. Design of the remote agent experiment for spacecraft autonomy. In *1998 IEEE Aerospace Conference Proceedings (Cat. No. 98TH8339)*, volume 2, pages 259–281. IEEE, 1998.

[49] Kanna Rajan, Douglas Bernard, Gregory Dorais, Edward Gamble, Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, Pandurang Nayak, Nicolas Rouquette, et al. Remote agent: An autonomous control system for the new millennium. In *ECAI*, volume 14, pages 726–730, 2000.

[50] Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davis, Dan Mandl, Stuart Frye, Bruce Trout, et al. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication*, 2(4):196–216, 2005.

[51] Martina Troesch, Faiz Mirza, Kyle Hughes, Ansel Rothstein-Dowden, Amanda Donner, Robert Bocchino, Martin Feather, Benjamin Smith, Lorraine Fesq, Brian Barker, et al. Mexec: An onboard integrated planning and execution approach for spacecraft commanding. 2020.

[52] Martin S Feather, Brian Kennedy, Ryan Mackey, Martina Troesch, Cornelia Altenbuchner, Robert Bocchino, Lorraine Fesq, Randall Hughes, Faiz Mirza, Allen Nikora, et al. Demonstrations of system-level autonomy for spacecraft. In *2021 IEEE Aerospace Conference (50100)*, pages 1–18. IEEE, 2021.

[53] Jo Bermyn. Proba-project for on-board autonomy. *Air & Space Europe*, 2(1):70–76, 2000.

[54] R Steel, M Niezette, A Cesta, S Fratini, A Oddi, G Cortellessa, R Rasconi, G Verfaillie, C Pralet, MICHÈLE Lavagna, et al. Advanced planning and scheduling initiative: Mrspock aims for xmas in the space domain. In *The 6th International Workshop on Planning and Scheduling for Space, IWPSS-09, July 19th-July 21st*, 2009.

[55] Massimo Tipaldi and Luigi Glielmo. A survey on model-based mission planning and execution for autonomous spacecraft. *IEEE Systems Journal*, 12(4):3893–3905, 2017.

[56] Jeremy Straub. A review of spacecraft ai control systems. In *Proc. 15th World Multi-Conference on Systemics, Cybernetics and Informatics*, 2011.

[57] Corrina Gibson, Michael Bonnici, and Jean-Francois Castet. Model-based spacecraft fault management design & formal validation. In *2015 IEEE Aerospace Conference*, pages 1–12. IEEE, 2015.

[58] Arturo Rankin, Tyler Del Sesto, Pauline Hwang, Heather Justice, Mark Maimone, Vandi Verma, and Evan Graser. Perseverance rapid traverse campaign. In *2023 IEEE Aerospace Conference*, pages 1–16. IEEE, 2023.

[59] Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, and Larry Matthies. Autonomous navigation results from the mars exploration rover (mer) mission. In *Experimental Robotics IX: The 9th International Symposium on Experimental Robotics*, pages 3–13. Springer, 2006.

[60] Max Bajracharya, Mark W Maimone, and Daniel Helmick. Autonomy for mars rovers: Past, present, and future. *Computer*, 41(12):44–50, 2008.

[61] Vandi Verma, Mark W Maimone, Daniel M Gaines, Raymond Francis, Tara A Estlin, Stephen R Kuhn, Gregg R Rabideau, Steve A Chien, Michael M McHenry, Evan J Graser, et al. Autonomous robotics is driving perseverance rover's progress on mars. *Science Robotics*, 8(80):eadi3099, 2023.

[62] Adham Atyabi, Somaiyeh MahmoudZadeh, and Samia Nefti-Meziani. Current advancements on autonomous mission planning and management systems: An auv and uav perspective. *Annual Reviews in Control*, 46:196–215, 2018.

[63] I Woodrow, C Purry, A Mawby, and J Goodwin. Autonomous auv mission planning and replanning-towards true autonomy. In *14th International Symposium on Unmanned Untethered Submersible Technology, Durham, NH*, 2005.

[64] Benjamin Ayton, Nikhil Bhargava, Tiago Vaquero, Eric Timmons, Brian Williams, and Richard Camilli. Risk-bounded goal-directed mission planning for ocean exploration. In *IJCAI'2017 Workshop on Artificial Intelligence in the Oceans and Space*, 2017.

[65] Fletcher Thompson and Damien Guihen. Review of mission planning for autonomous marine vehicle fleets. *Journal of Field Robotics*, 36(2):333–354, 2019.

[66] Bing Zhu, Lihua Xie, Duo Han, Xiangyu Meng, and Rodney Teo. A survey on recent progress in control of swarm systems. *Science China Information Sciences*, 60:1–24, 2017.

[67] Michael T Wolf, Amir Rahmani, Jean-Pierre de la Croix, Gail Woodward, Joshua Vander Hook, David Brown, Steve Schaffer, Christopher Lim, Philip Bailey, Scott Tepsuporn, et al. Caracas multi-agent maritime autonomy for unmanned surface vehicles in the swarm ii harbor patrol demonstration. In *Unmanned systems technology XIX*, volume 10195, pages 218–228. SPIE, 2017.

[68] Li A Zhang, Jia Xu, Dara Gold, Jeff Hagen, Ajay K Kochhar, Andrew J Lohn, and Osonde A Osoba. Air dominance through machine learning: A preliminary exploration of artificial intelligence-assisted mission planning. Technical report, RAND Corporation Santa Monica, 2020.

[69] Stuart Young and Alexander Kott. A survey of research on control of teams of small robots in military operations. *arXiv preprint arXiv:1606.01288*, 2016.

[70] Hans Röthlisberger. Water pressure in intra-and subglacial channels. *Journal of Glaciology*, 11(62):177–203, 1972.

[71] Tiago Stegun Vaquero, Guglielmo Daddi, Rohan Thakker, Michael Paton, A Jasour, Marlin P Strub, R Michael Swan, Rob Royce, Matthew Gildner, P Tosi, et al. Eels: Autonomous snake-like robot with task and motion planning capabilities for ice world exploration. *Science Robotics*, 9(88):eadh8332, 2024.

[72] R Thakker M Paton M Strub M Swan G Daddi R Royce M Gildner T Vaquero P Tosi M Veismann P Gavrilov E Marteau J Bowkett D Loret de Mola Lemus Y Kumar Nakka B Hockman A Orekhov T Hasseler C Leake B Nuernberger P Proença W Reid W Talbot N Georgiev T Pailevanian A Archanian E Ambrose J Jasper R Etheredge C Roman D Levine K Otsu H Melikyan R Rieber K Carpenter J Nash A Jain L Shiraishi A Agha-mohammadi M Travers H Choset J Burdick M Ono. Eels: Towards autonomous mobility in extreme environments with a novel large-scale screw driven snake robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2023.

[73] Shigeo Hirose and Makoto Mori. Biologically inspired snake-like robots. In *2004 IEEE International Conference on Robotics and Biomimetics*, pages 1–7. IEEE, 2004.

[74] Kristin Y Pettersen. Snake robots. *Annual Reviews in Control*, 44:19–44, 2017.

[75] Xuesu Xiao and Robin Murphy. A review on snake robot testbeds in granular and restricted maneuverability spaces. *Robotics and Autonomous Systems*, 110:160–172, 2018.

[76] Pål Liljebäck and Richard Mills. Eelume: A flexible and subsea resident imr vehicle. In *Oceans 2017-Aberdeen*, pages 1–4. IEEE, 2017.

[77] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. A review on modelling, implementation, and control of snake robots. *Robotics and Autonomous systems*, 60(1):29–40, 2012.

[78] James K Hopkins, Brent W Spranklin, and Satyandra K Gupta. A survey of snake-inspired robot designs. *Bioinspiration & biomimetics*, 4(2):021001, 2009.

[79] Bruce C Jayne. What defines different modes of snake locomotion? *Integrative and comparative biology*, 60(1):156–170, 2020.

[80] Florian Richter, Peter V Gavrilov, Hoi Man Lam, Amir Degani, and Michael C Yip. Arcsnake: Reconfigurable snakelike robot with archimedean screw propulsion for multidomain mobility. *IEEE Transactions on Robotics*, 38(2):797–809, 2021.

[81] Tal Dachlika and David Zarrouk. Mechanics of locomotion of a double screw crawling robot. *Mechanism and Machine Theory*, 153:104010, 2020.

[82] Kenji Nagaoka, Masatsugu Otsuki, Takashi Kubota, and Satoshi Tanaka. Terramechanics-based propulsive characteristics of mobile robot driven by archimedean screw mechanism on soft soil. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4946–4951. IEEE, 2010.

[83] Mohd Zamzuri Ab Rashid, Mohd Fitri Mohd Yakub, Sheikh Ahmad Zaki bin Shaikh Salim, Normaisharah Mamat, Sharifah Munawwarah Syed Mohd Putra, and Shairatul Akma Roslan. Modeling of the in-pipe inspection robot: A comprehensive review. *Ocean Engineering*, 203:107206, 2020.

[84] Jorge Villacrés, Martin Barczyk, and Michael Lipsett. Literature review on archimedean screw propulsion for off-road vehicles. *Journal of Terramechanics*, 108:47–57, 2023.

[85] Ansel Barchowsky, Ahmadreza Amirahmadi, Kyle Botteon, Gregory Carr, Curtis Jin, Patrick McGarey, Shelly Sposato, and Summer Yang. A high voltage tethered power system for planetary surface applications. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–8. IEEE, 2022.

[86] William Talbot, Jeremy Nash, Michael Paton, Eric Ambrose, Brandon Metz, Rohan Thakker, Rachel Etheredge, Masahiro Ono, and Viorela Ila. Principled icp covariance modelling in perceptually degraded environments for the eels mission concept. 2023.

[87] M. Troesch, F. Mirza, K. Hughes, A. Rothstein-Dowden, R. Bocchino, A. Donner, M. Feather, B. Smith, L. Fesq, B. Barker, and B. Campuzano. Mexec: An onboard integrated planning and execution approach for spacecraft commanding. In *Workshop on Integrated Execution (IntEx) / Goal Reasoning (GR),*

International Conference on Automated Planning and Scheduling (ICAPS IntEx/GP 2020), October 2020. Also presented at International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS 2020) and appears as an abstract.

[88] Kyohei Otsu, Scott Tepsuporn, Rohan Thakker, Tiago Stegun Vaquero, Jeffrey A Edlund, William Walsh, Gregory Miles, Tristan Heywood, Michael T Wolf, and Ali-Akbar Agha-Mohammadi. Supervised autonomy for communication-degraded subterranean exploration by a robot team. In *2020 IEEE Aerospace Conference*, pages 1–9. IEEE, 2020.

[89] Marta Cialdea Mayer, Andrea Orlandini, and Alessandro Umbrico. Planning and execution with flexible timelines: a formal account. *Acta Informatica*, 53(6-8):649–680, 2016.

[90] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. The agile manifesto, 2001.

[91] Allan Gut and Allan Gut. *Probability: a graduate course*, volume 200. Springer, 2005.

[92] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[93] Daniel Kahneman. Prospect theory: An analysis of decisions under risk. *Econometrica*, 47:278, 1979.

[94] Amos Tversky and Daniel Kahneman. The framing of decisions and the psychology of choice. *science*, 211(4481):453–458, 1981.

[95] Peter M Todd and Gerd Gigerenzer. Précis of simple heuristics that make us smart. *Behavioral and brain sciences*, 23(5):727–741, 2000.

[96] Gerd Gigerenzer and Daniel G Goldstein. Reasoning the fast and frugal way: models of bounded rationality. *Psychological review*, 103(4):650, 1996.

[97] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[98] Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Value iteration in continuous actions, states and time. *arXiv preprint arXiv:2105.04682*, 2021.

[99] Josep M Porta, Nikos Vlassis, Matthijs TJ Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. 2006.

[100] Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2):193–208, 2002.

[101] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[102] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[103] Johan Wahlström and Isaac Skog. Fifteen years of progress at zero velocity: A review. *IEEE Sensors Journal*, 21(2):1139–1151, 2020.

[104] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[105] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 433–445. Springer, 2011.

[106] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[107] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[108] Sung-Kyun Kim, Rohan Thakker, and Ali-Akbar Agha-Mohammadi. Bi-directional value learning for risk-aware planning under uncertainty. *IEEE Robotics and Automation Letters*, 4(3):2493–2500, 2019.

[109] Arthur George Richards. *Trajectory optimization using mixed-integer linear programming*. PhD thesis, Massachusetts Institute of Technology, 2002.

[110] Lars Blackmore, Masahiro Ono, and Brian C Williams. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, 2011.

[111] Egon Balas. Disjunctive programming. *Annals of discrete mathematics*, 5:3–51, 1979.

[112] Masahiro Ono et al. Robust, goal-directed plan execution with bounded risk. 2012.

[113] Marcio da Silva Arantes, Claudio Fabiano Motta Toledo, Brian Charles Williams, and Masahiro Ono. Collision-free encoding for chance-constrained nonconvex path planning. *IEEE Transactions on Robotics*, 35(2):433–448, 2019.

[114] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[115] Laurent Perron and Vincent Furnon. Or-tools.

[116] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[117] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. Motion planning around obstacles with convex optimization. *arXiv preprint arXiv:2205.04422*, 2022.

[118] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Proc. Lett.*, 1:132–133, 1972.

[119] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[120] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[121] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[122] Michael Paton, Richard Rieber, Sarah Cruz, Matt Gildner, Chantelle Abma, Kevin Abma, Sina Aghli, Eric Ambrose, Avak Archanian, Elizabeth A Bagshaw, et al. 2023 eels field tests at athabasca glacier as an icy moon analogue environment. In *2024 IEEE Aerospace Conference*, pages 1–18. IEEE, 2024.