Human-in-the-loop Planning of Mobile Robots

Xuning Yang

CMU-RI-TR-22-04

January 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Robotics.*

Abstract

Human-in-the-loop control for mobile robots is an important aspect of robot operation, especially for navigation in unstructured environments or in the case of unexpected events. However, traditional paradigms of human-in-the-loop control have relied heavily on the human to provide precise and accurate control inputs to the robot, or reduced the role of the human to providing supervisory task specifications. In this thesis, we explore a new paradigm of human-robot collaboration called human-in-the-loop planning, where the robot can act semi-autonomously according to the human's intention while having the human directly inform motion planning and trajectory generation. The proposed paradigm maximizes the strengths of the human and robot such that the human-robot system can perform with optimal efficiency.

To this end, we first abstract away complex vehicle dynamics by way of motion primitive teleoperation, which allows an operator to control a vehicle via input reparameterization into trajectories. We then build upon motion primitive teleoperation and present a method for reactive collision avoidance. We then propose a novel method of local trajectory generation without end goal specifications for human-in-the-loop control. The method, called Biased Incremental Action Sampling, is a sample based approach to build motion primitive trees that optimize for non-goal based cost functions. We then introduce hierarchical human-in-the-loop planning, which incorporates intended motions as global paths such that generated local trajectories can follow the paths autonomously. Lastly, we introduce continuous dynamic autonomy by generating path predictions on semantic topological navigation maps. By incorporating environment contexts into human-in-the-loop control, this allows us to reason about the human's intentions over the path space and generate predictions to assist navigation in unstructured, constrained environments.

Acknowledgments

First, I'd like to thank my committee for supporting my research throughout the years. I would like to thank my primary advisor, Nathan Michael, for your guidance and encouragement throughout my graduate school journey and many of the opportunities that you've provided me with along the way. I would also like to thank many other advisors I've had during my time at CMU, Koushil Sreenath and Jean Oh, for your guidance, support, and belief in me. It is through iterating ideas with you and through your feedback that I learned to grow as a researcher. To Henny Admoni, whom I have spent years discussing research with as a pseudo-advisor, I am grateful for your insights and support, as well as our coffee chats. I am fortunate to also have Sanjiban Choudhury and Helen Oleynikova on my committee, whose insights and expertise have been invaluable to developing this thesis, as well as many of the life advice that our conversations would inevitably evolve into. There are many other faculties and admins in the RI community whom I have received a tremendous amount of support from over the years that I am truly grateful for. Thank you to Tommy Liu, who encouraged me to come to CMU RI to begin with.

Most of my time at RI was spent at RISLab, and I would like to thank its past and present members: working together made the day-to-day of graduate school fulfilling and fun. Thank you to those who sat in the Master's suite, Aditya Dhawale, Arjav Desai, Cormac O'Meadhra, Jasmine Cheng, John Yao, Kshitij Goel, Lauren Lieu, Logan Ellis, Mike Lee, Tabitha Lee, Vibhav Ganesh, Wennie Tabib, and Yves Georgy Daoud, and its frequent visitors, Kumar Shaurya Shankar, Alex Spitzer, Derek Mitchell, Ellen Cappo, Micah Corah. Thank you to my first and always mentor, Vishnu Desaraju, for fielding relentless questions from me during my first two years (and many thereafter). Thank you to Curtis Boirum for being a steady presence for hardware and field experiments.

My time at CMU would have not been as fun without the camaraderie of many of the RI friends that have come and went over the years. To Achal Dave, Anirudh Vemula, Cara Bloom, Dhruv Saxena, Rosario Scalise, you guys are some of the very first and oldest friendships from RI. I am forever grateful that we met and became friends. Since then, we've only strengthened our friendships over time. To Shaurya, who has been a mentor, a friend, a voice of reason, and a calming presence since day one. To Aditya Dhawale, Jerry Hsiung, Mike Lee, thank you for all the nonstop laughter and so many great memories; it is very unfortunate that I will not be able to forget any of them. To Cherie Ho, thank you for all the great food adventures and many productivity hacks. To Ada Taylor, Adithya Murali, Ankit Bhatia, Christine Baek, Jaskaran Grover, Nadine Chang, Pragna Mannam, Roberto Shu, Rogerio Bonatti, Shushman Choudhury, Shohin Mukherjee, Stefanos Nikolaidis, Suddhu Suresh, Thomas Weng, Zhi Tan, thank you for all the fun times and many profound conversations we've had.

Each one of those conversations have impacted me and I am deeply grateful for our friendship. To Aditya Martowirogo and Aaron Lam, thank you for your friendship and tremendous support from afar throughout the years. There are many, many others whom I have interacted with over the years (Robohaus! PSRs! RoboOrg! Conferences!) that made this experience a life changing one; I am grateful that our paths crossed.

Lastly, thank you to Alex, who is always there to support and encourage me with eternal optimism and silly jokes. Thank you for all the candor, honest discussions on work, life, and everything in between. Thank you for being an anchor for me through all the ups and downs that this journey inevitably comes with. I have learned so much from you.

Finally, I'd like to thank my parents for their unconditional love, support, and sacrifice. It is through them that I understand what it means to have determination, resiliency, and perseverance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

While autonomous mobile robots have exploded in consumer and industrial applications in recent years such as exploration, autonomous driving, delivery robots, and entertainment, the scope of fully autonomous operation has been limited to structured environments with well-defined task objectives. Advances of mobile robotics in unstructured scenarios for arbitrary tasks are still reliant on human control due to several challenges. These challenges include encoding task objectives, interpreting scene into semantics, and making critical operating decisions for edge cases.

Humans on the other hand, can more efficiently reason about arbitrary scenes. This allows the human to identify goals, process the environment context and recognize its effects and implications faster and more accurately than autonomous platforms. The human intuition is invaluable and should be leveraged to improve robot operation in environments where full robot autonomy cannot be relied upon. For many navigation tasks, human-in-the-loop has been a key paradigm for robot control. However, the traditional paradigm has its host of inefficiencies which leads to a surge in interest for fully autonomous operations. In this thesis, we return to the human-in-the-loop control problem, and look to increase the efficiency of the human-robot system in navigation. This thesis focuses on the question of enabling seamless human-in-the-loop navigation with semi-autonomy, such that arbitrary navigation tasks can be effortlessly completed with both the human and the robot's strengths maximized.

1.1 Motivation

To begin this thesis, consider the two scenarios in Fig. 1.1. The first task is to perform search and rescue in a wooded area with rough terrain using an all-terrain vehicle (ATV) as shown in Fig. 1.1. The second task is a search and inspect operation in a potentially hazardous warehouse using a multirotor aerial vehicle (MAV) to identify potentially dangerous objects of interest.

Figure 1.1: Motivating scenarios for dynamic autonomy. (Left) a warehouse with unstructured obstacles. (Right) Satellite images of a rough terrain with traversability unknown.

Both of these scenarios follow the same task structure: Given an unstructured environment, the primary task is to navigate through the environment, and return to some arbitrary goal location. While the vehicle navigates, the operator must complete a secondary objective; for example, to look for an object of interest or to inspect a location of interest. The toy problem is illustrated in Fig 1.2.

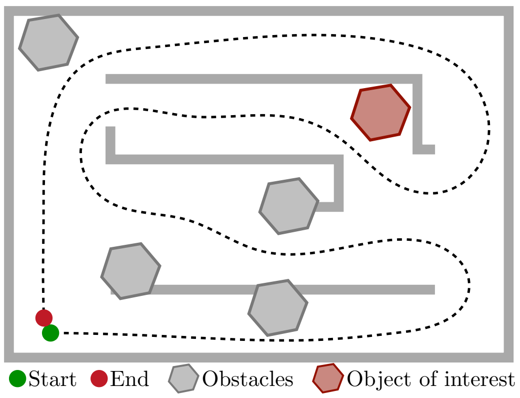


Figure 1.2: Simplified representation of a search/inspection navigation task. Given an unstructured environment, the task is to navigate through the environment while searching for an object of interest, and return to the starting location.

The task structure is highlighted by a few characteristics:

- The end goal configuration is irrelevant; often, it could be the same as the starting location.

- The choice of path is dependent on the underlying task objective.

Applications that follow this task structure are good examples of when to involve human-in-the-loop. Autonomous navigation algorithms that can directly accomplish the underlying objective are usually not readily available. Humans are recognizably superior as intuitive path planners given the semantics of the scene.

1.2 Human-in-the-loop Navigation

The HITL control problem can be simplified into a feedback control loop: The human observes the robot motion and provides inputs that will steer the robot to achieve the intended motion. Fig. 1.3 illustrates this continuous feedback control process.

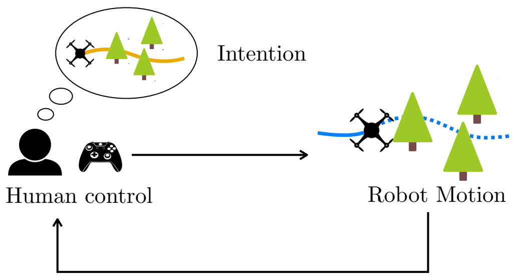


Figure 1.3: Illustration of the human-in-the-loop control problem: The human issues a command via an interface to the robot based on what they envision the motion that the robot should undertake. The robot executes motion based on the provided input. Based on the observed outcome, the human provides a new input, and the cycle repeats until the human's intention is achieved.

Such a feedback control problem involves two agents: The human pilot, and the robot. The common objective for both agents is to follow the intention of the human. In order to achieve that, the human needs to communicate to the robot via the input interface. The first key question is, *How does the human interface with the robot?*

Once an input is received from the human, the robot needs to interpret the input and translate it into motion. The robot needs to execute the best possible motion that it believes the human wishes to achieve. Therefore, the second key question becomes, *How should the robot use the human input to generate motion?*

For the most efficient human-robot system, the objective needs to be clearly understood by the robot, such that the robot can execute the motion that the human imagines. Possibilities of the human's intention is vast given the navigation context. Although task information may not be immediately or easily communicated to the robot a priori, the human's actions are clearly observable. The third question then becomes, *How should intention be represented and predicted for navigation tasks?*

1.3 Thesis Objectives

The roles of the human and robot is disproportionate in traditional HITL navigation: The human is responsible for most of the autonomy of the robot, including planning, safety, dynamic feasibility, and disturbance rejection. The human then provides a system-level control input to the robot, i.e., an input that corresponds to the control dynamics of the system. The robot will simply set the reference for its low-level controllers to the provided control input.

The ideal human-robot system should play to the strengths of both: While humans are recognizably superior as path planners, they are less capable at computing precise control inputs while focusing on multiple objectives such as planning, safety, and responding to unanticipated scene changes.

Autonomy in robotics can compensate for these weaknesses. When humans are in control of the robot, "the intelligence of the system being controlled determines how involved the human becomes in the process."[1]. In other words, the maximum level of autonomy of the robot determines the minimum level of required control from the human. This gives us a key insight: *If we can shift the control responsibility from the human to the robot wherever possible, then the human-robot system becomes more efficient.* However, this comes with a caveat: the autonomy assumed by the robot must be aligned with the expectation of the human.

The thesis statement is as follows.

The goal of this thesis is to enable seamless human-in-the-loop control of a mobile robot, such that the robot navigates in unstructured environments according to the human's intention.

To achieve this goal, this thesis addresses the three areas of HITL introduced previously in detail. This thesis proposes to increase the efficiency of the human-robot navigation system via increasing the autonomy of the robot in general robot navigation, motivated by the examples in unstructured scenarios. In achieving this objective, we look at some of the deficiencies in current HITL paradigms, and identify methodologies that help alleviate these inefficiencies.

The human-in-the-loop navigation should satisfy the following three criteria:

1. **Intuitive:** The operator's inputs should result in expected motions of the robot.
2. **Safe:** The robot should remain safe in unstructured environments in the presence of unanticipated obstacles.
3. **Efficient:** The robot should navigate according to the human's intended path without requiring precise human instructions.

1.4 Challenges

We begin addressing the thesis problem by inspecting three areas of challenges corresponding to HITL navigation as outlined in Section 1.2.

Table 1.1: Challenges associated with the HITL problem.

Human control	**Robot motion**	**Intention**
▶ Complex robot dynamics ▶ Timescales ▶ Multitasking	▶ Plan in the absence of goals ▶ Legibility of robot motion	▶ Intention representation ▶ Incorporating environment priors

1.4.1 Challenges associated with Human Control

Challenge 1: Reduce complexities for the human operator to control the robot.

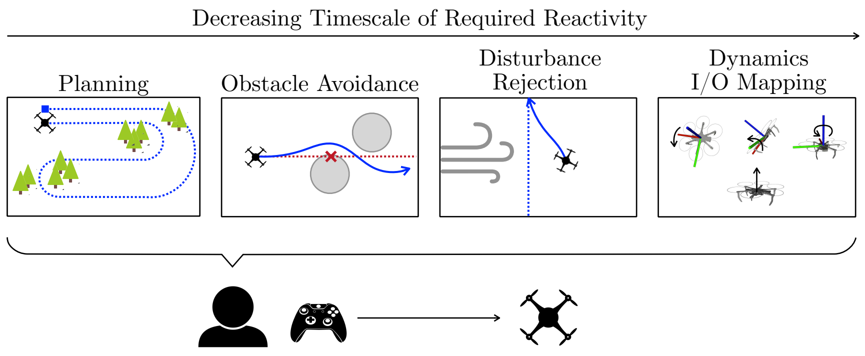


Figure 1.4: Illustration of multitasking requirements on human-in-the-loop control: Multiple objectives on various timescales needs to be taken into consideration of computing a single control input that is provided to the robot.

1.4.1.1 Complex robot dynamics

It is difficult for a human operator to control the robot precisely given the complex dynamics of the system with little to no prior experience. Vehicles such as MAV's have high degrees-of-freedom, which is controlled by giving inputs as angular velocities, as illustrated in Fig. 1.5. The system dynamics that maps these inputs to the output state space is unintuitive to the human.

For any dynamical system, the human learns the system dynamics by exploring the available actions and observing feedback of the change in the robot state. This process allows the operator to implicitly learn the mapping from the input to the resulting motions. By

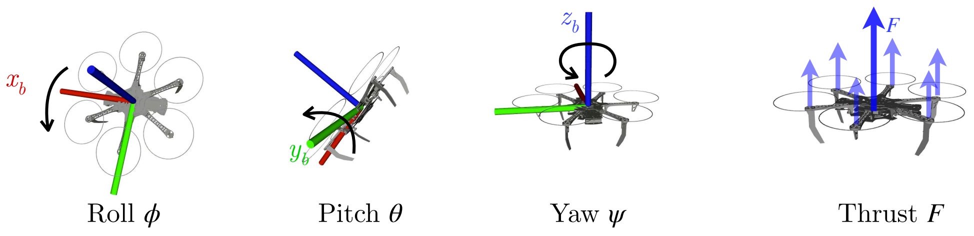


Figure 1.5: Multirotor Aerial Vehicle (MAV) Dynamics.

doing so, the operator is implicitly learning an approximate model of the robot's dynamics, which allows the operator to generate an appropriate input given a desired output. The learning time for any operator is correlated with the complexity of the system, thereby limiting teleoperation to skilled operators. The complexity of the system dynamics is a barrier to allowing the average human to be able to control a system with ease.

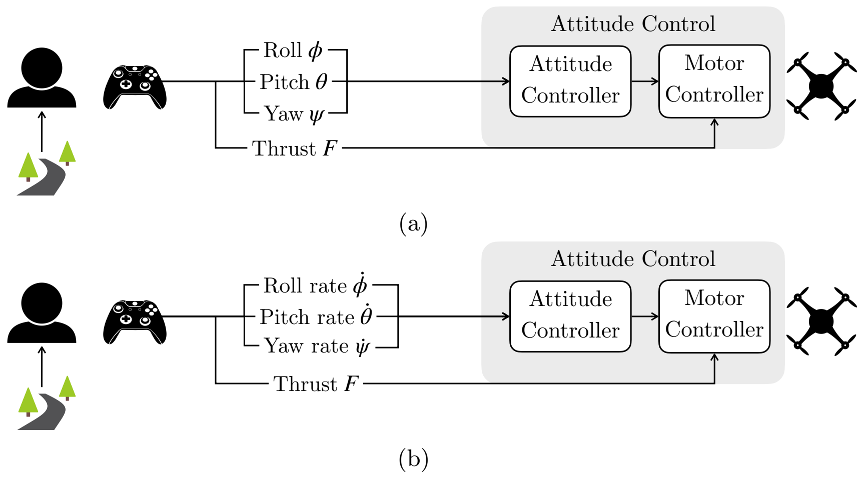
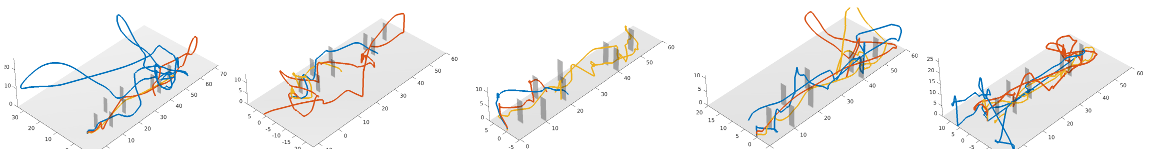
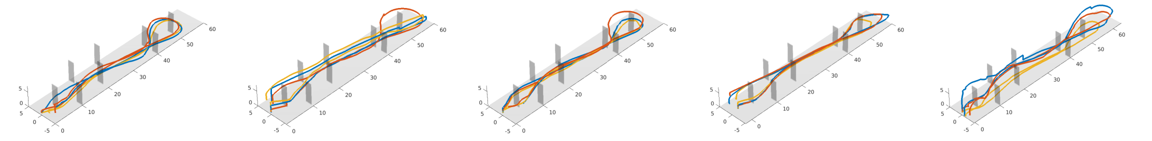


Figure 1.6: Traditional RC control for Multirotor Aerial Vehicles (MAV) with (a) orientation and (b) angular velocities.

An example is shown in Fig. 1.7, where a set of naive operators was asked to complete a navigation task that requires weaving through a series of pillars without given a specific trajectory to follow. It is clear that many naive operators fail to complete the navigation task by providing low-level system commands (RC inputs) (Fig. 1.7a). With an intuitive teleoperation method (Fig. 1.7b), it is clear that without specifying the exact trajectory for the task, the naive operators are able to control the vehicle with ease and with resulting trajectories that look very similar.



(a) Resulting trajectories using traditional RC teleoperation.



(b) Resulting trajectories using motion-primitives based teleoperation, introduced in Chapter 3.

Figure 1.7: A comparison of trajectories of teleoperating a MAV using traditional RC and a more intuitive, motion primitives method in a pillar navigation task with naive operators. The RC method proves to be unintuitive and results in suboptimal performance, whereas the more intuitive method allows the operator to complete the task easily regardless of their skill level.

1.4.1.2 Timescales of tasks

Humans, on average, have reaction times[1] between 200-250 ms, and for elite trained vehicle operators, 140-200 ms [2]. This corresponds to an average reaction time of 3-7 Hz. Local replanning typically happens around 1 Hz and up to 10 Hz depending on the environment density and trajectory horizon [3], with global plans generated at a much slower rate. However, control inputs are required at a much faster rate: for a multirotor vehicle, control frequency can range around 100 Hz to 500 Hz, depending on the level of control that the operator is responsible for. This timescale is illustrated in Fig. 1.8. Lastly, visual scene changes at a rate that is dependent on the vehicle's speed and the environment structure; in constrained environments, this may result in unanticipated obstacles appearing at faster rates than the human can process. While the human is capable of planning and handling environment changes that happen at a lower rate, computing precise control inputs given vehicle system rates and responding to unpredictable external disturbances require fast response rates. The slow response time of the human is ultimately inefficient in handling these objectives.

1.4.1.3 Multitasking

The human operator is overloaded with multitasking between planning, reacting to unexpected scene changes, estimating disturbances and computing the appropriate control inputs (Fig. 1.4). This is highly inefficient for the human operator. While humans are recognizably superior as path planners, they are less capable at computing precise control inputs while focusing on multiple objectives.

[1]A visual-based reaction test can be found at `https://humanbenchmark.com/tests/reactiontime`, with the median being 215 ms over 81 million test points.

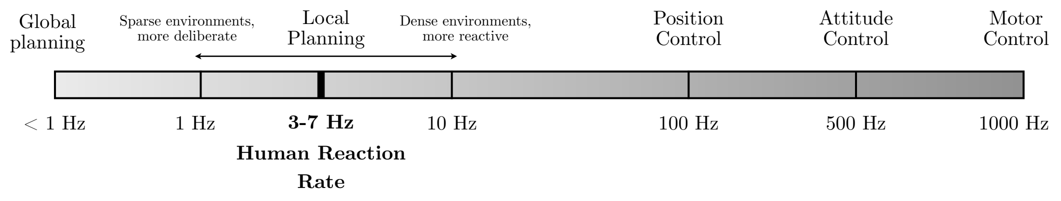


Figure 1.8: Timescale between system frequency vs. human reaction. Depending on the level of control that the human-robot system interfaces at, the human cannot react fast enough to provide precise inputs at a rate that is comparable to an autonomous control system. However, human operator can accommodate lower rate tasks such as planning and semantic understanding, given that a scene does not visually change faster than the human's processing time.

Studies have shown that by adding one additional task to a primary task, the human's processing bandwidth on the primary task reduces by approximately 37% even if the human does not engage in the secondary task [4]. Simply paying attention to a secondary focus leads to a deterioration in the primary task performance. Further, satisfying objectives with various time scales causes physiological stress and fatigue [2, 5], leading the operator to provide erroneous control inputs that could introduce instability in the system. Therefore, to increase the efficiency of the human-robot system, the robot should enable the operator to focus solely on the navigation task. This requires shifting the responsibilities of safety, dynamic feasibility, reactivity to environment to the robot.

1.4.2 Challenges associated with Robot Motion Planning

Challenge 2: Plan dynamically feasible and collision-free trajectories that follow the human operator's desired path.

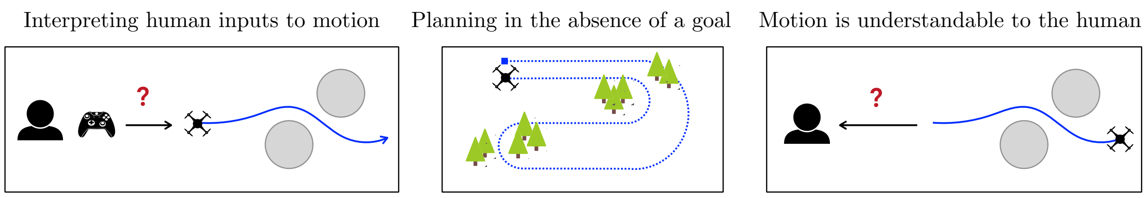

Interpreting human inputs to motion
?
Planning in the absence of a goal
Motion is understandable to the human
?

Figure 1.9: Illustration of challenges associated with HITL motion generation

1.4.2.1 Motion planning without state-space goals

The general problem of motion planning in robotics has traditionally referred to the problem of computing collision-free paths from a start to a final goal configuration. For human-in-the-loop navigation, it is not often that a goal configuration can be clearly defined with respect to the task. The example tasks described in Sect 1.1 illustrates this point: For many tasks, the task objective is defined by the navigation path rather than the start and end goal configuration, which in many cases, could be the same.

Instead of planning from a starting configuration to a goal configuration, human-in-the-loop requires generating plans that follow desired motions. Much of modern literature on motion planning focuses on planning to a goal. This thesis aims to address planning and trajectory generation that optimize for motions in the absence of state-space goals.

1.4.2.2 Legibility of Robot Motion

An additional constraint that must be considered for HITL control is the legibility of the robot motion. *Legible motion* [6] is defined as motion that expresses its intent without obfuscation. The motion that the robot performs needs to be clearly understandable to the operator, such that the human can assess the trajectory of the robot and provide feedback. An example is shown in Fig. 1.10. As a robot approaches an obstacle, it must remain collision-free. Although both trajectories are collision-free, the trajectory on the right does not express it in a way that is clearly understandable to the human.

This constraint, while perhaps not a significant concern for autonomous operation, is a key concern especially for mobile robot teleoperation. When direct communication of intent is unavailable or infeasible, providing passive feedback in the form of motion is an implicit way to communicate understanding of intention.

1.4.3 Challenges associated with Intention Representation for Human-in-the-loop navigation

Challenge 3: Represent and predict the operator's intention in navigation tasks.

1.4.3.1 Intention as paths

The HITL feedback control problem involves two agents: The human operator, and the robot. The common task objective for both agents is the intention of the human, which is non-observable to the robot.

In a human-robot collaboration scenario, *intention* can be represented in various ways. Notably, the intention can be represented either as a goal [7–9] in the state space (e.g., a door, an address, a specific object), or they can be represented as a path [10–12] (e.g., a sequence of streets to take to reach an address). However, the operator's intention can also be represented at a lower level, either as a specific motion [13, 14] (e.g., a left turn, a right turn or perform an in-place yaw), or as a system set-point [15, 16] (e.g., drive a vehicle at a fixed velocity). These various levels of intention are dependent on the task objective and interface design.

This thesis primarily focuses on intention representation for navigation tasks as *paths.* The operator typically has a specific path that the vehicle should take. Representing intentions as goals may overlook some of the granular details of human preference, such as trajectory behavioral characteristics and shape profile.

1.4.3.2 Incorporating Known Priors for Predicting Human Intention

Contextual information provides rich information regarding the human's intention. While task descriptions and/or goals communicated to the robot a priori can be a definitive model of human intention, contextual cues such as environment models can provide expository semantics. Understanding such semantics via inference allows the robot to understand the operator's intention, which is non-observable.

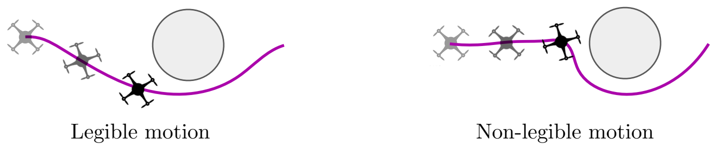


Figure 1.10: Legible vs. non-legible motion of a quadrotor approaching an obstacle. The legible motion clearly indicates a desire to evade the obstacle by going right. The non-legible motion approaches the obstacle straight on, and it is not immediately clear during the approach that the vehicle will go left, right, stop or crash into the obstacle.

1.5 Thesis Outline

To address these challenges, this thesis explores *human-in-the-loop planning* (Fig. 1.11). The idea of human-in-the-loop planning is as follows: Instead of using the operator's input directly to control the robot, the input is used to inform the generation of a trajectory. This thesis presents a variety of work related to human-in-the-loop planning, with increasing planning horizon and autonomy.

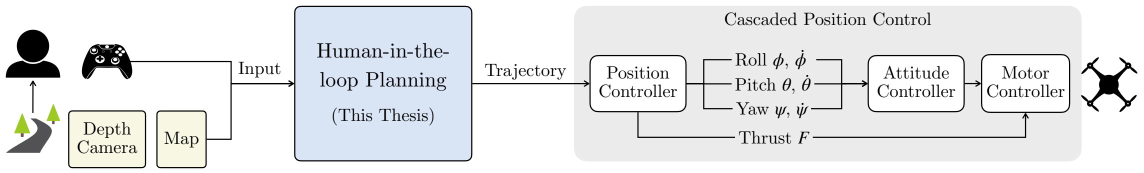


Figure 1.11: Proposed HITL framework for a MAV. Compared to traditional HITL frameworks such as Fig. 1.6, this thesis explores human-in-the-loop planning, such that the operator's input is used to generate a trajectory, which is then used to generate references for the vehicle controller.

In **Chapter 2**, we define the domain of human-in-the-loop control and formalize the teleoperation problem, and provide mathematical preliminaries required in order to contextualize subsequent chapters. Additionally, related works that pertain to various aspects of our problem domain are presented and discussed.

In **Chapter 3**, we begin addressing the thesis problem by moving from providing control-level inputs to specifying state-space motions by way of motion primitives. By doing so, we abstract away complex dynamics. Sections of this chapter first appear in [17–19].

In **Chapter 4**, we build upon the previous chapter and discuss reactive autonomous collision avoidance based on motion primitives. We present fast collision checking with respect to two types of map representation: a KD-Tree Voxel based representation, and a Gaussian mixture model (GMM) based map representation. Sections of this chapter first appear in [19, 20].

In **Chapter 5**, we move from one-step reparameterization into local trajectory generation, which extends the planning horizon by generating trajectories that actively avoid obstacles. To do so, we propose generating motion primitive trees computationally efficiently via an Biased Incremental Action Sampling (BIAS). The contents of this chapter first appear in [21].

In **Chapter 6**, we introduce hierarchical HITL, which incorporates both global paths and local trajectories. The proposed framework incorporates the operator's intended path into the planning stack in addition to generating locally dynamically feasible and collision-free trajectories. The contents of this chapter first appear in [22].

In **Chapter 7**, we introduce continuous dynamic autonomy given a compact representation of a constrained environment. The environment context is incorporated into the HITL framework as a semantically topological navigation graph. Path predictions are generated on the navigation graph, which is used to assist navigation via continuous dynamic autonomy.

We conclude the thesis in **Chapter 8** with some of the key insights obtained throughout this work, and lay out potential future directions. A summary of the presented work is shown in Table 1.2.

Table 1.2: Table of Contributions

Human control	Ch. 3: Motion Primitives for HITL navigation
	Ch. 4: Reactive collision avoidance
Robot motion	Ch. 5: Local trajectory generation for HITL navigation
	Ch. 6: Hierarchical HITL navigation
Intention	Ch. 7: Continuous dynamic autonomy

Chapter

2

Background

2.1 Preliminaries

Unless otherwise specified, we will use the following notation and definitions below:

Operator input. The input from the operator is are received as a continuous stream of real values coming from a continuous input device. Each *input* or *action* from the operator is represented by $\mathbf{a} = \{a_1, ...a_q\}$, for q input dimensions. To identify unique inputs from the continuous stream, a *novel/unique/new input* from the operator is defined as an input that leads a zero-order hold over at least 100ms. Unless otherwise noted, $\mathbf{a}$ will identify a unique input from the operator.

Each of the input dimension space $\mathcal{A}_i$ is assumed to be a convex closed set. The action space is given by $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_q$.

For the multirotor, the input will be $\mathbf{a} = \{v, \omega, v_z\}$, where $v \in \mathcal{V}_x$ is the forward linear velocity, $\omega \in \varOmega$ is the yaw velocity (or the angular velocity about yaw), and $v_z \in \mathcal{V}_z$ is the velocity along the world-aligned z-axis. The action space is consequently $\mathcal{A} = \mathcal{V}_x \times \varOmega \times \mathcal{V}_z$.

State. The *planning state* or *state* is given by $\mathbf{x} = \{x, y, z, \theta\}$, where x, y, z is a position in Euclidean space, and θ is the heading (or yaw) of the vehicle aligned with respect to the world-aligned z-axis. The state at a given time is denote by $\mathbf{x}_t$. The *state space* is given by $\mathcal{X} = \mathbb{R}^3 \times [-\pi, \pi]$. The state space is *safe* if $\mathcal{X}_{\text{safe}} = \mathcal{X} \backslash \mathcal{O}$, where $\mathcal{O}$ is the obstacle space.

The time derivatives with respect to the state are given by $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$, $\dddot{\mathbf{x}}$, $\mathbf{x}^{(4)}$, representing the velocity, acceleration, jerk and snap respectively.

Note that the state is not to be confused with the *control state*, which for a multirotor would include the Euclidean position, the orientation using Euler angles, and the associated angular velocities: $\mathbf{x}_{\text{control}} = \{x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r\}$.

Path. A *path* is a sequence of waypoint states that specifies the vehicle's position: $\rho = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathcal{X}$. The path is *safe* if $\forall\, \mathbf{x}_i,\ i \in \{1, ..., n\}, \mathbf{x}_i \in \mathcal{X}_{\text{safe}}$.

Trajectory. A *trajectory* is a time-parameterized function $\xi(t)$, defined over a time interval $t \in [0, T]$ that maps a given time t to a state[1] $\mathbf{x}$, i.e., $\xi\ :\ [0, T] \rightarrow \mathcal{X}$, where the duration $T \in [0, \infty)$. A trajectory is *safe* if $\xi_{\text{safe}}\ :\ [0, T] \rightarrow \mathcal{X}_{\text{safe}}$. A *trajectory space* is the span of the trajectory space $\xi \in \varXi$. The trajectory space is safe if every trajectory in the trajectory space is a safe trajectory.

Intention Model. The operator model $\varPhi$ outputs a trajectory given the operator's input $\mathbf{a}$ and environment variables $\mathcal{M}$: $\varPhi(\mathbf{a}, \mathcal{M}) = \xi$.

2.2 Problem Setup

Recall an earlier insight, *the maximum level of intelligence of the robot determines the minimum level of required control from the human.* This implies that by sequentially increasing the intelligence of the robot, operator engagement can be reduced accordingly.

We propose to address the thesis problem by using trajectory-based teleoperation. The focus is to generate trajectories that allow the operator to focus solely on the navigation task by increasing the autonomy of the robot.

The thesis problem is as follows:

Trajectory-based Teleoperation Problem. Given an operator input $\mathbf{a}$ at any time t and an environment representation $\mathcal{M}$, ensure that the vehicle's trajectory ξ satisfies the following:

$$\min\ s(\xi, \hat{\xi}) \tag{2.1}$$

$$\text{s.t.}\ \ \xi \in \varXi_{\text{safe}} \tag{2.2}$$

$$\hat{\xi} = \varPhi(\mathbf{a}_{0,\dots,t}, \mathcal{M}) \tag{2.3}$$

where $\hat{\xi}$ is a trajectory obtained from an intention model $\varPhi(\mathbf{a}, \mathcal{M})$ and $s(\xi, \hat{\xi})$ is a similarity metric evaluated between trajectories ξ and $\hat{\xi}$.

This document breaks down Problem (2.1) to be addressed in parts. First, the completed work demonstrates an appropriate *trajectory representation* such that teleoperation can be performed without operator's explicit expertise, resulting in expected motions that are natural to the operator. Second, we ensure that the trajectories generated satisfy *collision avoidance* as the operator teleoperates. Then, we demonstrate *trajectory generation* in

[1]If the trajectory function is continuously differentiable, then instead mapping to the state $\mathbf{x}$, the trajectory function maps a given time t to the *control state* $\mathbf{x}_{\text{control}}$, i.e., $\xi\ :\ [0, T] \rightarrow \mathcal{X}_{\text{control}}$.

unstructured environments under a known intended trajectory or intention model. Lastly, we infer the human's *intention* in the space of trajectories and utilize it to guide navigation.

These works aim to increase the efficiency of the human-robot navigation. Throughout this thesis, we define the efficiency of the human-robot system measured in terms of human operator engagement[2]– that is, if the human interacts with the robot frequently to provide inputs. We hypothesize that, if the system is performing as expected, the human does not need to intervene to provide corrective inputs.

2.3 Related Works

The related works will cover human-in-the-loop control and assistance, human intention representation, motion planning and trajectory generation, as well as trajectory similarity measures.

2.3.1 Human-in-the-loop Control of Mobile Robots

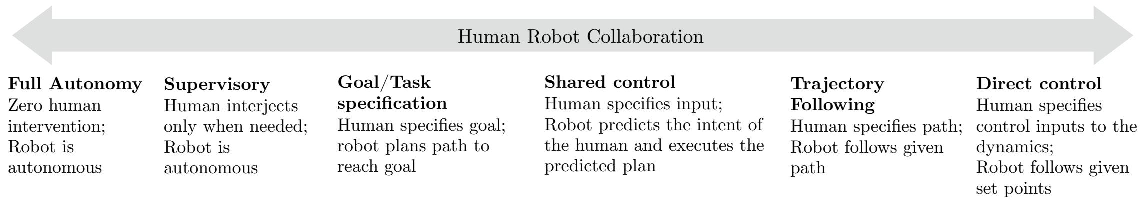

Human Robot Collaboration
Full Autonomy
Zero human intervention; Robot is autonomous
Supervisory
Human interjects only when needed; Robot is autonomous
Goal/Task specification
Human specifies goal; robot plans path to reach goal
Shared control
Human specifies input; Robot predicts the intent of the human and executes the predicted plan
Trajectory Following
Human specifies path; Robot follows given path
Direct control
Human specifies control inputs to the dynamics; Robot follows given set points

Figure 2.1: A scale of human-in-the-loop control and assistance for mobile robots.

2.3.1.1 Human-in-the-loop Control

One of the earliest works on human-robot interaction describes the type of interactions between human and robots as *supervisory control*, where the supervisor gives directives, which are in turn understood and translated into detailed actions by the subordinate [1]. These earlier ideas interpret supervisory control to include the span of autonomy between the extremes of full direct control and fully autonomous control. In recent literature, the involvement between human and robots has evolved from supervisory to collaborative, leading to the human and robot sharing responsibilities.

In Fig. 2.1 we provide an evolved scale of human-in-the-loop control [23]. We focus our discussion on teleoperation and mobile robots, and describe related literature on various subdomains outlined in this scale.

Direct control. In traditional multirotor UAV teleoperation, the operator typically issues low-level inputs to the vehicle according to the vehicle dynamics, requiring experience

[2]Operator *engagement* is defined to be the number of times that an operator has to provide distinctive inputs in order for the vehicle to move as the operator intended.

in teleoperation to maintain stability [24]. Haptic feedback informs and prevents collisions at the interface level [25–27]. Low-level control inputs from the operator are augmented in order to avoid the immediate obstacles [24, 28].

Trajectory specification. The operator can also directly provide a trajectory shape for a vehicle to follow [29]. In this form of control, while high level, the operator is responsible for concatenating trajectories in order to achieve the task while the onboard automation is responsible for following the trajectory and guaranteeing stability.

Shared control. For shared control methods, an intended policy is usually predicted, and the final system input is a function of both the predicted policy and the user's actual policy. The type of policy differs depending on the system. If the policy is in the form of an *input*, linear arbitration [30] is a widely used method to allocate control between the system and the user given a particular arbitration function [8, 31–36]. Some examples of arbitration functions are based on uncertainty [37], a max function over probability [33], or manually tuned [34]. For policies modeled as a POMDP, the choice of action can either be influenced by the user input [38, 39], or it could be used to indicate whether to follow the user's policy or the predicted policy [40]. If the predicted policy is a *trajectory*, [41] generates safe mini-trajectories for each incremental waypoint while [42] uses a cooperative motion planner to optimize the robot's trajectory to the forecasted one. [43] describes three Bayesian approaches for providing navigational assistance: a Maximum Likelihood (ML) approach that chooses the trajectory that maximizes the user model, a Maximum A Posteriori (MAP) approach that maximizes the posterior probability, and a greedy POMDP approach for multi-model estimation. Assistance approaches vary in terms of the level of control abstraction from the actual user input, from distinct arbitration to the user input acting as a prior in Bayesian methods.

Goal/Task specification. This is the highest level of teleoperation that the operator can engage in without having a fully autonomous system. In this case, the operator teleoperates a robot by providing high level commands such as selecting sequential sub-goals or specifying sub-tasks. The robot is responsible for the autonomy to complete the sub-task, computing the relevant trajectories and translating them into low level control inputs that are safe and stable [33, 44].

2.3.1.2 Human Decision Models

The Boltzmann noisily-rational decision model [45] is a commonly accepted model for modeling human decision making. This model stems from theories of human decision making, in particular, Luce's axiom of choice [46]. In Luce's choice model, the humans's decision making process is modeled as a reward function, maximizing the weighting (or reward) of one particular choice over a set of choices. The Boltzmann model extends the model to utilize exponential rewards, thereby formulating a soft-max function over the weights associated with each choice [47].

The reward-based intent model has been popularized in many different applications [48–51]. Model learning and incorporating feedback from the human has been widely studied within the context of this framework [52].

2.3.1.3 Human Intent Representation

Intention can represent different quantities given particular scenarios: In [53], a direct robot *input* is inferred using a physics-based model. For intent modeled as a *latent variable*, [54] constructs a probabilistic representation for predicting measurement and transition models using Gaussian Processes (GP), whereas [55] employs a single latent variable to represent user behavior trained using an artificial neural network (ANN). *Task*-based intent can be inferred using pre-defined task features [33] or probabilistically using Gaussian Mixture Autoregression with statistical features [42]. For *trajectory* inference, a mixture distribution over a set of composite trajectories for multiple agents is used in [56], whereas a Bayesian trajectory recognition framework that provides a probabilistic distribution over a set of possible trajectories to possible goals is used in [38, 57, 58]. If intent is described as a *cost function*, [59] defines a joint user-robot cost function which is refined iteratively using Kullback-Liebler Divergence based on the principle of minimum cross-entropy. Intent defined as *goals* can be inferred using a Voronoi diagram combined with local Gaussian Mixture Models [60], using inverse models of system states [41], using artificial potential fields [61], or using Maximum Entropy inverse optimal control (MaxEnt IOC) to construct a distribution of all possible goal states, assuming the user is approximately optimizing some cost function for their intended goal [8, 30, 36]. In controlled settings, intent representation trends toward goal-based methods, whereas dynamic environments typically utilize task or trajectory based methods.

2.3.2 Planning and Trajectory Generation

2.3.2.1 Motion Planning

Given an objective and an environment, motion planning methods involves two separate methods: 1) Optimization methods, or 2) using sample-based motion planning coupled with smoothing using optimization.

Optimization-based methods. Optimization methods require computing the explicit gradient of the objective with respect to its parameters, usually a cost formulation over an Euclidean Signed Distance Field (ESDF) [62–64]. CHOMP [63] is a well-known optimization based approach, generating paths in the global space with respect to the gradient of the cost. However, CHOMP does not produce a time-parameterized trajectory; In [65], instead of optimizing for the path points, the optimization computes the end-point derivatives, which can be used to compute a time-parameterized polynomial.

Sample-then-smooth methods. Sample-based motion planning usually separates the trajectory generation problem into a bi-level kinodynamic planning problem: 1) generate a geometric path in the state-space to seed an initial path, and then 2) refined using an optimization based method to generate a dynamically feasible trajectory from the geometric path [66–71]. [66, 68, 70] search directly in the space of discretized higher order derivatives instead of points in the state space, then refine the resulting path via bspline optimization using an ESDF of the environment. [72] generates a topological roadmap using a visibility Probabilistic Roadmap (PRM) by sampling directly in the free Euclidean space, which is then refined using bspline optimization.

Alternatively, an alternate approach is to directly sample from the space of trajectories [73] and output a distribution of parameters that represent the minimum cost trajectories. Monte Carlo Tree Search (MCTS) has been applied in single and multi-robot motion planning for mobile robots [74, 75]

Motion primitives. Instead of geometric paths, one can compute motion primitives on discretized lattice grids for ground [76, 77] and aerial [78] robots. By concatenating motion primitives, one can build a tree of trajectories [79], although a naive tree construction could face combinatorial explosion in the number of resulting nodes, which can quickly become intractable to evaluate in terms of the number of trajectories.

Global-local planning architectures. Global-local planning architectures have been deployed in autonomous systems for global navigation and local collision avoidance for both UAVs [80, 81] and ground robots [82–85]. These methods usually plan a global path to a pre-specified goal location using a global planner such as A* or RRT variants, which is then used as a guiding path for local trajectory generation and selection that satisfies dynamic constraints and safety requirements.

2.3.2.2 Trajectory Representation

Polynomial. The most common trajectory parameterization is a polynomial specified by the coefficients of the polynomial. The coefficients can be uniquely determined by fixed initial and final control states and a fixed duration. This representation is well used in MAV trajectory generation as continuity up to jerk is required in order to fully determine the control inputs to the quadrotor model [86, 87].

B-spline. A B-spline is a piecewise polynomial parameterized by a set of control points, a knot vector. The key to the b-spline representation is that it has the convex hull property, which allows the higher order derivatives (velocity, acceleration, jerk, etc) to be checked against dynamic feasibility subject to higher order derivative constraints. B-spline representations implicitly guarantee continuity between higher order derivatives between subsequent control points [69, 70, 88].

2.3.2.3 Trajectory Similarity Measures

Common trajectory similarity measures include Dynamic Time Warping (DTW), Fréchet distance, and Hausdorff distance. Dynamic time warping aligns two sequences of points non-linearly in the time dimension to find similarities in the space dimension of two matching paths. Hausdorff distance evaluates the similarity between two sets of paths by computing the supremum of the set of point-wise infimum distance between discrete points. The Fréchet distance is similar in definition to the Hausdorff distance, but differs in that the two sequences are *ordered*, therefore the infimum lengths computed must be evaluated by following the order of the points such that the pairs may not backtrack or skip beyond its sequence neighbors.

2.4 Assumptions

There are many system and design considerations associated with teleoperation that may be of interest. We limit the scope to follow the assumptions listed below.

Latency. There are three broad types of latency inherent in teleoperation systems: 1) The physiological latency between the time that the operator receives feedback and when an operator issues a command, and 2) the latency between the time that an operator command has been issued until the vehicle is able to execute that command, and lastly, 3) the time between the vehicle executing that command and the time feedback has reached the operator. In this work, we assume that the operator command is able to be executed without delay with immediate visual feedback, therefore ignoring delays caused by (2) and (3), acknowledging that (1) is inherent in all human operators. While the methods presented may indirectly alleviate issues with (2) and (3), its effects are not studied extensively as a part of this work.

Visual feedback. Systems with degraded visual feedback may greatly affect the operator's decision making process. This work does not investigate the effects of visual information degredation caused by occlusions. It is assumed that the operator has sufficient visual information in order to make an informative decision. Therefore, the operator is presented with a third-person-observer view that follows the level frame of the vehicle in simulation. For MAVs, this perspective view is often not achievable on hardware without extensive mechanical and system engineering in stabilizing and transmitting vehicle-mounted camera views. Therefore, the hardware experiments are carried out with the operator directly observing the vehicle from ground.

Haptic feedback. Haptic or input-related feedback can be an useful tool in informing the operator of the vehicle's surroundings. However, we exclude this consideration in our work and does not assume dependency or use of haptic feedback devices.

Human interface design. The physical interface used by the human operator can greatly influence how the operator interact with the robot. An interface can be physical (e.g., a control panel) or virtual (e.g., a mouse-based interface to select buttons on a screen); as well as discrete (e.g., buttons) or continuous (e.g., knobs, joystick, steering wheels). Typically, continuous inputs are associated with lower-level control whereas buttons are associated with high-level control such as mode selection. In our work, we assume that the operator is controlling the robot via a standard continuous stream input device, such as a joystick or a gamepad with continuous knobs.

Engagement. It is assumed that the human operator is fully engaged with the robot, and that the human operator does not have any distractions that could result in periods of negligence.

Collaboration. It is assumed that the human operator is teleoperating the robot is not an adversarial agent. For example, if the robot is heading into an wall that extends infinitely, the operator does not continue to drive the vehicle into the wall. Instead, the operator is expected to steer it away to avoid a collision.

CHAPTER

3

Motion Primitives based HITL Control

One of the challenges in easing teleoperation for general operators is to allow the operator to focus on the motions of the vehicle in the state space. This chapter introduces motion primitives based teleoperation, which allows a direct parameterization of a system's inputs to its state space outputs by way of forward propagating dynamic models. While motion primitives based teleoperation of ground vehicles can directly invert and propagate a unicycle model, this is difficult and unintuitive to do for a complex system such as the multirotor UAV. The key insight of this chapter is that, we show that the same unicycle model can be translated to multirotor UAVs by leveraging differentially flat properties of multirotors. This strategy facilitates teleoperating complex systems that may have difficult dynamics by using a familiar model that may already be known to operators. Further, dynamical models such as multirotor UAVs require continuity of higher order differentials in order to execute aggressive and smooth motions; we show that this can be achieved using the selected model while maintaining continuity of the trajectory up to snap.

Sections of this chapter first appeared in [17–19].

3.1 Introduction

For high dimensional nonlinear robotic systems, the choice of using teleoperation has been the preferred method of control and planning for navigating robots in unfamiliar scenarios. In direct teleoperation of mobile robots, operators issue inputs, such as linear and angular velocities to the robot. As mobile robots are often operating in dynamic environments with disturbances, the operator is required to be vigilant, reactive, and precise in issuing correct inputs in order to keep the robot safe. To mitigate human error and to allow the operator plan the robot trajectories with certainty, this chapter presents a novel abstraction of system-specific control inputs, or *actions*, into the state space for teleoperation. Given an action and a kinematic or dynamic model, the input is mapped using some given dynamics into a time-parameterized trajectory. Thus, motion primitives represent temporally extended actions in the state-space. Motion primitives are well-known tools in planning for manipulation, gait, and mobile systems [89–91]. In this work, we leverage this technique for teleoperation.

For systems with a known kinematic or dynamic model, the motion primitives are, in fact, equivalent to input-based teleoperation as one can forward propagate the model for the selected action; however, any kinematic model that does not violate non-holonomic assumptions may be used instead of the full dynamics of the robot. *By construction, the direct correspondence between the action space and the state space allows teleoperation to occur directly in the state space of the robot.* Motion primitive based teleoperation allows the operator to solely act in the role of the planner, alleviating the operator from having to provide high frequency and reactive inputs in the presence of disturbances. In this chapter, we begin with motion primitives for ground vehicles that are continuous up to acceleration, and present snap-continuous motion primitives for aerial vehicles, ensuring smooth teleoperation at high speeds.

Second, this chapter introduces a novel assistance methodology by moderating the available range of motion according to the predicted directional intent via sampling. We model the operator as an optimal controller [92]. Consequently, the operator's action selection policy is modeled as a reward function. We assume that the operator optimizes a reward function over time, and issues an action that most closely reflects the optimal reward. This assumption allows assistance to be formulated for perpetual tasks, such as navigation and exploration for ground and aerial vehicles. Contrary to other methods, limitation of the available motion primitive set still allows the operator to retain control over the robot. *The implicit assumption here is that a human's criterion of optimality will be satisfied with some bound around the quantifiably optimal value; thus restriction of the allowable set of motion primitives will actively aid the operator.* By subsampling the available range of motions, we ensure that the operator's choice of motion follows the probabilistic distribution over the action space. We compare our approach to two baseline tests: a naïve filtering method on previous inputs, and direct velocity-based teleoperation without adaptation.

We evaluate the proposed and baseline approaches using behavioral entropy techniques and show provably better performance of the resulting trajectories and adherence to the operator's intended direction than baseline methods.

3.2 Approach

We begin with a discussion of teleoperation using motion primitives in Section 3.2.1 and discuss motion primitive generation for ground and aerial vehicles.

The adaptive teleoperation framework is presented in Section 3.2.2 given a set of known motion primitives. The operator model is provided in Section 3.2.2.1. Adaptive teleoperation using the model is presented in Sect. 3.2.2.4.

3.2.1 Motion Primitives for Mobile Robots

3.2.1.1 Control input parameterization

Inputs from the operator are received as a continuous stream of real values coming from a continuous input device, such as a joystick or gamepad.

We define an *action* to be a set of inputs provided via an external input device. For q input dimensions, an action is denoted as $\mathbf{a} = \{a_1, \dots, a_q\}$. Furthermore, we assume that each of the input domain a_i is convex. For the continuous input, we discretize the input values in each dimension to obtain the action space, which consists of q sets of N finite actions.

A motion primitive $\gamma(t)$ is a parameterized function defined over a time interval $t \in [0, T]$ which generates a unique sequence of states given an initial state $\mathbf{x}_0 \in \mathcal{X}$, and an action $\mathbf{a}$:

$$\gamma_{\mathbf{a},T} : [0, T] \to \mathcal{X} \qquad \mathbf{a} \in \mathcal{A},\ T \in [0, \infty) \tag{3.1}$$

where $\mathcal{A} = \mathbb{R}^m$ is the action space. The span of the action set generates a motion primitive library (MPL), which is a discrete set of motion primitives created using the discretized action set:

$$\mathit{\Gamma} = \{\gamma_{\mathbf{a}_i,T}\},\ i = 1, \dots, N \tag{3.2}$$

with N actions, $\{\mathbf{a}_i\}, i = 1, \dots, N$. Hence, an MPL will contain $q \times N$ motion primitives, and is treated as an indexed set of motion primitives parameterized by the action space.

Between sequential inputs, continuity and smoothness are enforced in the transition from one motion primitive to another. If the operator provides an input at time t_f, where $0 < t_f \leq T$, then $\gamma_t(t_f) = \gamma_{t+1}(0)$ for position and higher derivatives of the sequential motion primitives. As unsmooth transitions between motion primitives may saturate motor limitations, the design of the transitions is crucial for providing teleoperation safety. In the following sections, we discuss *forward-arc motion primitives* for ground robots in Section 3.2.1.2, and extending it to dynamically feasible aerial motion primitive in Section 3.2.1.3.

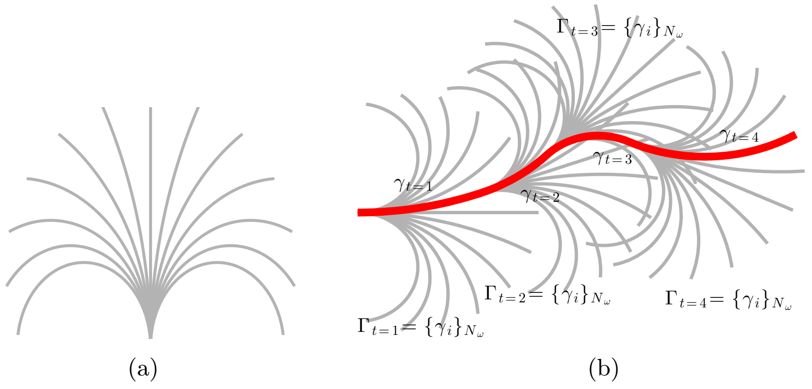

$\Gamma_{t=3} = \{\gamma_i\}_{N_\omega}$
$\gamma_{t=4}$
$\gamma_{t=3}$
$\gamma_{t=1}$
$\gamma_{t=2}$
$\Gamma_{t=2} = \{\gamma_i\}_{N_\omega}$
$\Gamma_{t=4} = \{\gamma_i\}_{N_\omega}$
$\Gamma_{t=1} = \{\gamma_i\}_{N_\omega}$
(a)
(b)

Figure 3.1: (a) A motion primitive library generated for a 2D unicycle model. For easier visualization, only discretizations in angular velocity ω are shown. (b) A trajectory formulated over four time steps. The selected motion primitives at each time step (in red) form a single trajectory.

3.2.1.2 Motion Primitives for Ground Vehicles

The simplest class of motion primitives are obtained by forward propagating the kinematics (or dynamics) of the robot. If the kinematic model is known, then a motion primitive can be defined as the position and higher order derivatives obtained by propagating the selected action by some constant amount of time, T.

Forward-arc motion primitives are an example of parameterized motion primitive for ground vehicles by by propagating the dynamics of a unicycle model with a constant linear and angular velocity for a specified amount of time, T [93]. The motion primitives are given by the solution to the unicycle model (3.3):

$$\dot{\mathbf{x}}(\mathbf{a},\tau) = \begin{bmatrix} v_x \cos(\omega\tau) \\ v_x \sin(\omega\tau) \\ \omega \end{bmatrix} \tag{3.3}$$

The solution to Eq. (3.3) is:

$$\mathbf{x}_{t+T} = \mathbf{x}_t + \begin{bmatrix} \frac{v_{xt}}{\omega_t}(\sin(\omega_t T + \theta_t) - \sin(\theta_t)) \\ \frac{v_{xt}}{\omega_t}(\cos(\theta_t) - \cos(\omega_t T + \theta_t)) \\ \omega_t T \end{bmatrix}, \tag{3.4}$$

where $\mathbf{x}_t = [x_t, y_t, \theta_t]^\top$ represents the pose of a ground vehicle at time t in the body frame, and v_{xt}, ω_t are the linear and angular velocities of the vehicle at time t in the body frame, respectively. The action space is given by uniformly dense sets of actions, denoted as the following: $\mathcal{V}_x = \{v_{xi}\}, i = 1, \ldots, N_{v_x}$, $\varOmega = \{\omega_j\}, j = 1, \ldots, N_\omega$. A single motion primitive at each time t is denoted by $\gamma_t = \{\mathbf{a}_t, T\}$ where $\mathbf{a}_t = \{v_{xt}, \omega_t\}$ and T is some fixed duration indicating the length of the motion primitive. Note that the operator has the ability to generate a new input at any time t_f for $0 < t_f \leq T$. Thus if no new input is received, then $\mathbf{a}_{t+1} = \mathbf{a}_t$ and $\varGamma_{t+1} = \varGamma_t$.

For differential drive ground vehicles, these motion primitives are equivalent to input-based teleoperation. An example of the forward-arc MPL in 2D is shown in Fig. 3.1a. Figure 3.1b depicts the MPL selected at each time t with the initial condition matching that of the current robot state. The set of each selected motion primitive at each time t forms a smooth trajectory in a fixed frame.

3.2.1.3 Snap-Continuous Motion Primitives for Multirotor Aerial Vehicles

The dynamics of a MAV are cannot easily be forward propagated. However, multirotors are differentially flat systems [86], implying that exact control inputs can be computed such that the vehicle follows a specified trajectory in the flat outputs x, y, z, and yaw θ. This property can be exploited to generate motion primitives using a choice of a dynamic model, since time-parameterized trajectories generated in the flat outputs can be satisfied with respect to dynamic feasibility bounds.

For ground vehicles, the heading of the vehicle is fixed to the yaw of the vehicle by nature. Although aerial platforms such as quadrotors can independently control heading from yaw, we maintain the use of a unicycle model by ensuring that the heading is equivalent to the yaw of the vehicle, as humans naturally optimize for curved trajectories in robot control [94] with heading aligned with the zero yaw angle.

From Eq. (3.4), it is evident that forward-arc motion primitives preserve continuity up to velocity in concatenating consecutive motion primitives, as it is the solution to the kinematic model of ground vehicles. To apply these motion primitives for quadrotor vehicles, we need to preserve the smoothness between consecutive motion primitives. For quadrotors, smoothness usually requires continuity up to jerk [86]. Thus, we extend Eq. (3.4) to generate motion primitives that retain differentiability up to jerk and continuity up to snap. From the resulting motion primitives, we can calculate desired vehicle attitude, angular velocity, and angular acceleration for use as feedforward terms in the controller.

The motion primitives are parameterized as follows. We define a *local frame* $\mathcal{L}$ to be a fixed z-axis aligned frame, taken at a snapshot in time. The motion primitive definition will be provided in the local frame at the time at which an input is issued, and can be freely transformed into a fixed global frame or body frame for control purposes. The unicycle model in 3D is:

$$\dot{\mathbf{x}}(\mathbf{a}, \tau) = \begin{bmatrix} v_x \cos(\omega\tau) \\ v_x \sin(\omega\tau) \\ v_z \\ \omega \end{bmatrix}, \tag{3.5}$$

where $\tau \in [0, T]$ and $\mathbf{x}_t = [x_t, y_t, z_t, \theta_t]^\top$ is the pose of the aerial robot with θ_t being the yaw of the vehicle. The additional input dimension has action space $\mathcal{V}_z = \{v_{zk}\}, k = 1, \ldots, N_{v_z}$ for vertical velocity. The action is defined as $\mathbf{a}_t = \{v_{xt}, v_{zt}, \omega_t\}$ in the local frame at which the input is issued, where v_x is the x-velocity (i.e., the forward velocity of the vehicle), v_z is the z-velocity, and ω is the yaw rate.

To generate snap-continuous motion primitives, we constrain the initial and final higher order derivatives of the motion primitive, with the endpoint velocities provided by Eq. (3.5). The position endpoints are unconstrained and we enforce all higher order derivatives above

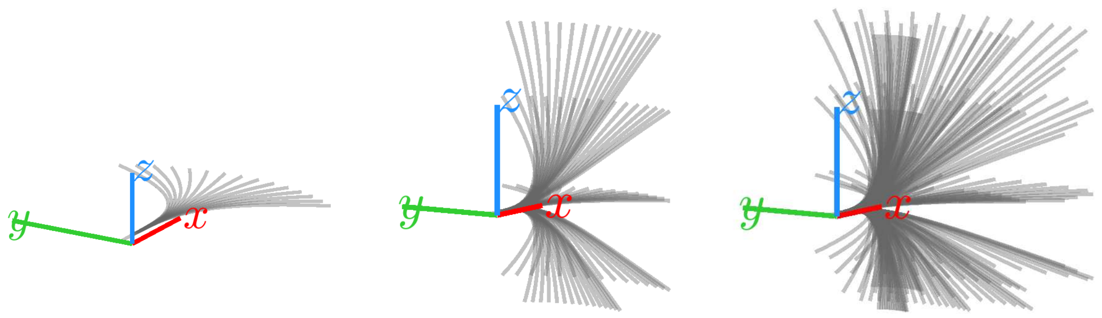
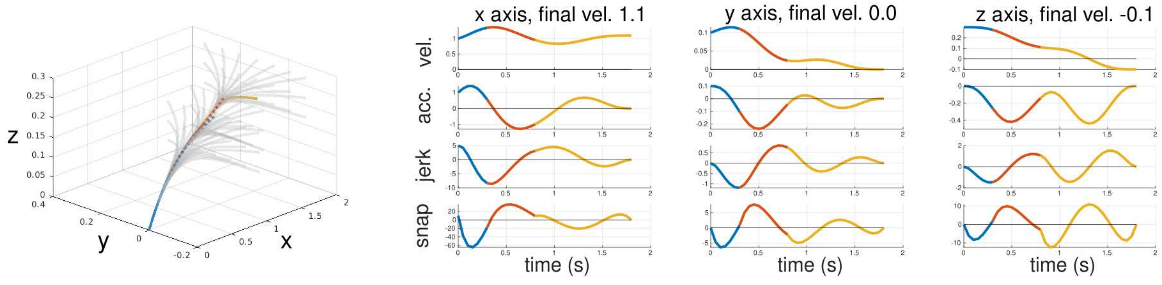


Figure 3.2: Motion primitive library constructed with forward arc primitives. The variations in angular velocity, z velocity, and linear velocity are added incrementally for maximum clarity.



(a) (b)

Figure 3.3: (a) A trajectory composed of 3-segments of motion primitives that switches to a new motion primitive at arbitrary points along the trajectory that have non-zero higher-order-derivative terms. The discarded trajectory is shown in dotted lines. (b) Higher-order time derivatives (velocity, acceleration, jerk, and snap) of the three segment trajectory, showing that the trajectories are differentiable up to jerk and continuous in snap at the switching points. At the end of the trajectory, all higher order derivatives are zero except for the operator specified velocity.

velocity to be zero.

Regeneration step k occurs at time $t_k \in \mathbb{R}^+$ when a new input is received from the joystick, or the previous trajectory $\gamma_{\mathbf{a}_{k-1},T}$ finishes executing. Alternatively, a fixed regeneration rate can be chosen in order to accommodate changes in the environment for collision avoidance. A library of dynamically feasible motion primitives is generated in the local frame $\mathcal{L}_{t_k}$ specified by the reference state at time t_k, i.e. $\mathbf{x}_{\text{ref}}(t_k) = \gamma_{\mathbf{a}_{k-1}}(t_k - t_{k-1})$, given a set of discretized actions $\mathbf{a}_k$. Each motion primitive γ is a vector of four 8th order polynomials that specify the trajectory along the position coordinates x, y, z and yaw coordinate θ. Given an action $\mathbf{a}_k$, each motion in the motion primitive library is generated in frame $\mathcal{L}_{t_k}$ according to

$$\begin{aligned}
\gamma_{\mathbf{a}_k,T}(t) &= \sum_{i=0}^{8} \mathbf{c}_i t^i \\
\text{s.t. } \gamma_{\mathbf{a}_k}^{(j)}(0) &= \mathbf{x}_{\text{ref}}^{(j)}(t_k) \;\; \text{for } j = 0,1,2,3,4 \\
\dot{\gamma}_{\mathbf{a}_k}(T) &= \dot{\mathbf{x}}_{\mathbf{a}_k}(T) \\
\gamma_{\mathbf{a}_k}^{(j)}(T) &= 0 \;\; \text{for } j = 2,3,4
\end{aligned} \tag{3.6}$$

where $\{\cdot\}^{(j)}$ specifies the jth time derivative. Note that all constraints are appropriately transformed into $\mathcal{L}_{t_k}$.

The result of having snap-continuous motion primitives (see Fig. 3.3) ensures that we have smoothness in error dynamics, thus minimizing instabilities and tracking error while the vehicle travels at high speeds.

3.2.2 Adapting Motion Primitives According to an Operator Model

In this section, we discuss a novel methodology for adapting the set of available motions to the operator according to an optimal control based operator model. We define an operator intent model over the space of motion primitives with the operator acting as an optimizing controller. Using the inference model, we provide assistance in the form of adaptation by subsampling the set of available motion primitives. However, a key assumption of the following framework is that the operator is *not adversarial.* This is to say, the operator will always act in favor of their intended motion and will not attempt to circumvent assistance.

3.2.2.1 Operator Intent Model and Inference

We assume that the operator inherently optimizes a reward function, but the action selected at each time step does not optimally reflect this function. This is the notion of "good-enough" – that humans operate within some region of optimality but do not always select a single optimal action [95]. In this particular problem, the operator issues action $\mathbf{a}$

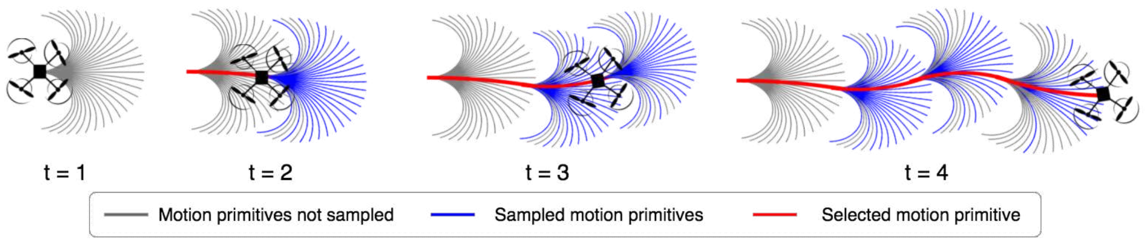

t = 1
t = 2
t = 3
t = 4
Motion primitives not sampled
Sampled motion primitives
Selected motion primitive

Figure 3.4: An illustrative graphic of the proposed approach. The operator begins each trial with access to an uniformly dense motion primitive library (MPL) that varies only in angular velocity. Over time, our algorithm updates the belief distribution over the set of motion primitives with respect to the operator's intent. At each input time t, a subset of the motion primitives is sampled with respect to the belief distribution. The operator input maps to a motion primitive from the subsampled set via a selector function.

at each input time t, which is in some neighborhood of $\mathbf{a}^\star$, the optimal action that satisfies:

$$\mathbf{a}^\star = \underset{\mathbf{a}}{\operatorname{argmax}}\, R_t(\gamma_{\mathbf{a}}) \approx \underset{\mathbf{a}}{\operatorname{argmax}} \sum_i^Q \alpha_t^i \phi_t^i(\gamma_{\mathbf{a}}), \tag{3.7}$$

where ϕ^i's are basis functions defined with respect to quantifiable natural human behavior, for a total of Q basis functions. We assume the reward function is composed of linear basis terms. Thus, the inference problem is the prediction of the underlying reward function, $\hat{R}_t = \sum_i^Q \hat{\alpha}^i \phi_t^i$, from the series of noisy operator inputs $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{t-1}\}$.

Using this model, we infer the operator's motion selection policy as the solution to the following optimization problem:

$$\begin{aligned} \hat{\gamma}_{t+1} &= \underset{\gamma_{\mathbf{a},t+1}}{\operatorname{argmax}}\, R_t(\gamma_{t-m:t}, \gamma_{\mathbf{a},t+1}) \\ &= \underset{\gamma_{\mathbf{a},t+1}}{\operatorname{argmax}} \sum_i^Q \alpha^i \phi_t^i(\gamma_{t-m:t}, \gamma_{\mathbf{a},t+1}), \end{aligned} \tag{3.8}$$

where $\gamma_{t-m:t}$ represents a trajectory formed by the past m motion primitives at time t, and $\gamma_{t+1} \in \mathit{\Gamma}_{t+1}$. Equation (3.8) is the key assumption that reflects the notion of "good-enough," which allows us to define assistance that inherently reflects this property.

Given an estimate of the operator's reward function, we can iteratively update the probability of a motion primitive being selected at the next time iteration. The prediction update is provided in Eq. (3.9). Given an initial uniform distribution over the set of motion primitives, one can update the probability of the n-th motion primitive being chosen at the next time step given the probability of the reward based on a segment of the previous

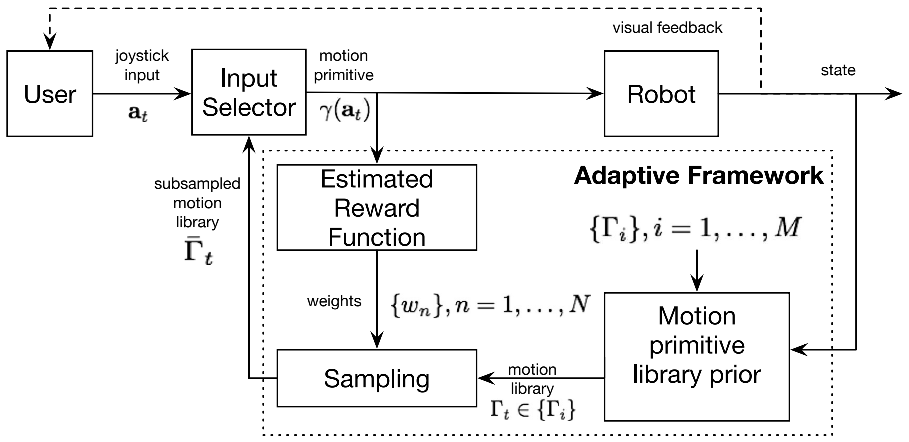


Figure 3.5: System diagram of the proposed adaptive framework. The operator issues an input from a joystick and selects a motion primitive as described by the input selector. While the robot follows the motion primitive, the choice of motion primitive is used in the adaptation framework to generate a subsampled set of the full underlying motion primitive library that tailors better towards the operator's intent. At the next time step, the joystick action is mapped to a motion primitive in the subsampled set.

trajectory:

$$\begin{aligned} p(\gamma^n_{t+1}|\gamma_{t-m:t}, \hat{R}_t) &= \frac{p(\hat{R}_t|\gamma_{t-m:t}, \gamma^n_{t+1})p(\gamma^n_{t+1}|\gamma_{t-m:t})}{p(\hat{R}_t|\gamma_{t-m:t})} \\ &= \eta\, p(\hat{R}_t|\gamma_{t-m:t}, \gamma^n_{t+1})p(\gamma^n_{t+1}|\gamma_{t-m:t}) \end{aligned} \tag{3.9}$$

where the past trajectory is denoted by $\gamma_{t-m:t}$, which is the past m selected motion primitives as observed at time t, and the distribution $p(\hat{R}|\gamma_{t-m:t}, \gamma_{t+1})$ estimates the reward function of the operator, as introduced in Sect. 3.2.2.1, with η being the normalization weight.

3.2.2.2 Reward Bases

For mobile robots, the operator acts as a high level trajectory planner. To this end, we define several bases that are intrinsic to human motion planning, inspired by [56, 96]. As the operator issues inputs over time, the inputs are made to follow a trajectory that is more conducive to human motions; thus a natural choice of bases is to optimize for what humans deem to be natural trajectories.

It is entirely possible for bases to only be functions of past trajectories (called *hindsight bases*) for the purposes of intent prediction. However, it is possible to incorporate prior knowledge, either in the form of a known trajectory or other environment models such as disturbance or obstacles fields. Here, we present three hindsight bases for the quadrotor air vehicle (smoothness, orthogonality, and time), and an additional distance metric given a desired trajectory to follow (distance error). These are defined as follows:

Smoothness We define smoothness as the magnitude of change in the input:

$$\phi_{\text{smoothness}}(\gamma_{t+1}, \gamma_{t-m:t}) = \sum_{j=t-m+1}^{t+1} \|\mathbf{a}_j - \mathbf{a}_{j-1}\|_1 \tag{3.10}$$

Orthogonality We penalize any drastic deviation in heading from the previous trajectory. This is defined using a simple ratio for three points as follows:

$$\phi_{\text{orthogonality}}(\gamma_{t+1}, \gamma_{t-m:t}) = \frac{\|p_t - p_{t-m}\| - \|p_{t+1} - p_t\|}{\|p_{t+1} - p_{t-m}\|} - 1 \tag{3.11}$$

where each point $p_\tau \in \mathbb{R}^3$ is the position at time step τ at time T obtained from the motion primitive: $p_\tau = \gamma_\tau^{\mathbf{a}}(t = T)$.

Time We define the cost of time as the inverse of the linear body velocity, which allows for inference over the desired speed of motion:

$$\phi_{\text{time}} = \frac{1}{v_x} \tag{3.12}$$

Distance Error If a desired trajectory is provided, the distance error between the motion primitive and the trajectory can be calculated by approximating the area or volume for 2D and 3D trajectories respectively [97]. Given two paths $\{p_1, ..., p_n\}$ and $\{q_1, ..., q_n\}$, the distance error between the two can be defined as:

$$\phi_{\text{distance}} = \sum_{i=2}^{n} \frac{1}{2} \left(\|p_i - p_{i-1}\|_2 + \|q_i - q_{i-1}\|_2\right) \|p_i - q_i\|_2 \,. \tag{3.13}$$

3.2.2.3 Reward Function Estimation

The reward function is inferred from a past window of m motion primitives. The belief distribution of the reward $p(\hat{R}|\gamma_{t-m:t}, \gamma_{t+1}^n)$ is computed over the entire motion primitive library. For this study, we assume that each input dimension is conditionally independent. The belief distribution is computed using an online function approximation to estimate the reward function, $\hat{R}_t = \sum_i^Q \hat{\alpha}^i \phi_t^i$. We employ Locally Weighted Projection Regression (LWPR), a computationally efficient online method for local approximations of high dimensional nonlinear functions [98] to estimate the reward function. The incremental algorithm performs global function approximation by taking a weighted sum of the local regressors that influence the region.

The regression over the reward bases is defined with respect to a linear global reward function, which is estimated using LWPR. A global LWPR model is sufficient for tasks with a single intent. However, tasks that may require dynamic changes in intent – for example, high-speed driving or flight with inference over the angular velocity – such tasks require a more temporally local prediction in order to adjust to these changes. To continue leveraging the speed of LWPR, we keep a rolling queue of l LWPR models. At each input time, a model M_t is popped off the queue and a new model M_{t+l} is added to the queue. Each model in the queue is updated with the data received at that time. The prediction at $p(\hat{R}|\gamma_{t-m:t}, \gamma^n_{t+1})$ is then generated with model M_t. This is a local batch estimation method for time-varying intent functions that has shown to be computationally tractable over the time span of interest. More succinct online regression methods that are amenable to time varying models and conducive to a small number of data points are an area of interest and will be addressed in future work.

3.2.2.4 Adaptation Using the Operator Intent Model

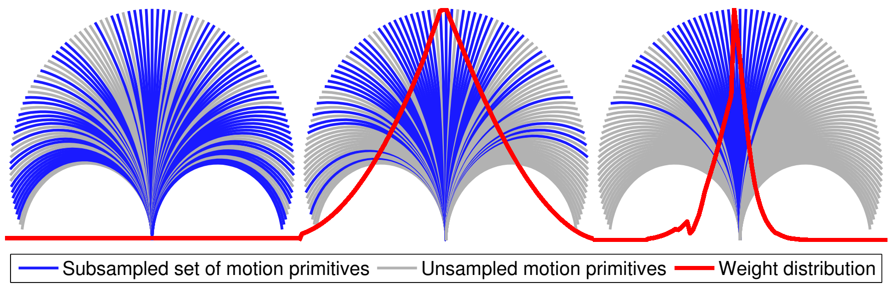

Subsampled set of motion primitives
Unsampled motion primitives
Weight distribution

Figure 3.6: Motion primitives and distribution over the motion primitives at selected times along a racetrack at timesteps $t = 0$, $t = 5$, and $t = 150$ respectively. The prediction becomes more peaked near the mean of the predicted motion primitive.

Given a dense set of motion primitives and a probabilistic distribution over these motion primitives, assistance is provided by limiting the set of available motion primitives to a subset adhering to the operator's intent. By defining a "good-enough" region of interest around the optima, the set of motion primitives within the region remains safe and feasible, and most closely reflect the operator intent. We assume that within some short duration, the operator's intent does not fluctuate widely and single instances of input that are outside of the interest region are unintended by the operator.

We adaptively modify the subset of available motion primitives from an underlying set of motion primitives such that the density of the subsampling reflects the reward function distribution $p(\hat{R}|\gamma_{t-m:t}, \gamma_{t+1})$. By use of motion primitives, a particular choice of action $\mathbf{a}_t$ at time t is represented by its parameterized motion primitive γ_t for some fixed duration T. The key insight here is that we have removed the dependency of trajectories on the continuous input space, thus allowing inference to be made over a set of motion primitives, which is the set of local trajectories that is safe and feasible.

To construct the set of available motion primitives, we subsample motion primitives using importance sampling. As such, within the region of interest, fine-grained control of the action is preserved by the density of motion primitives. The subsampled set closely adheres to the operator's underlying region of optimality and circumvents misaligned motions to the operator's interest.

Let the weight of the n^{th} motion primitive be $w_n = p(\hat{R}|\gamma_{t-m:t}, \gamma_{t+1}^n)$. Given a motion primitive library $\mathit{\Gamma}$ of size N, we sample K motion primitives using the weights $\{w_n\}, n = 1, \dots, N$ with replacement such that we obtain a subsampled set:

$$\bar{\mathit{\Gamma}} = \{\gamma^k\} \subseteq \mathit{\Gamma}, k = 1, \dots, K, \tag{3.14}$$

which is the available set of motion primitives provided to the operator.

To map the operator input to a specific motion primitive in the subsampled set, a selector function (3.15) is used to select the motion primitive with the closest parameterization of the actual operator input $\mathbf{a}_{\text{input}}$ in the continuous input case:

$$\gamma_{\text{selected}} = \gamma\{\operatorname*{argmin}_{\mathbf{a}} \mathbf{a} - \mathbf{a}_{\text{input}}\} \in \bar{\mathit{\Gamma}}. \tag{3.15}$$

The adaptation methodology is summarized in Algorithm 1. A visualization of this algorithm is provided in Fig. 3.6.

Algorithm 1: Algorithm for updating construction of the subsampled set

```
1 $\mathit{\Gamma}_{t+1} \leftarrow \{\gamma^n\};\ n = 1, \dots, N;\ \mathit{\Gamma}_{t+1} \sim U(0,1)$;
2 $\bar{\mathit{\Gamma}}_{t+1} \leftarrow \emptyset$;
3 for $n = 1 : N$ motion primitives do
4     Calculate weights for each motion primitive $w_n = p(\hat{R}|\gamma_{t-m:t}, \gamma_{t+1}^n)$;
5 for $k = 1 : K$ do
6     Sample $\gamma^k \in \mathit{\Gamma}_t$ with probability $w_k$ with replacement;
7     $\bar{\mathit{\Gamma}}_{t+1} \leftarrow \bar{\mathit{\Gamma}}_{t+1} + \gamma^k$;
8 $\bar{\mathit{\Gamma}}_{t+1}$;
```

3.3 Experiments and Results

We present simulation and experiment results for a quadrotor micro-air vehicle. We introduce a method validation criterion in Section 3.3.2. Vehicle experiment results are presented in Section 3.3.3.

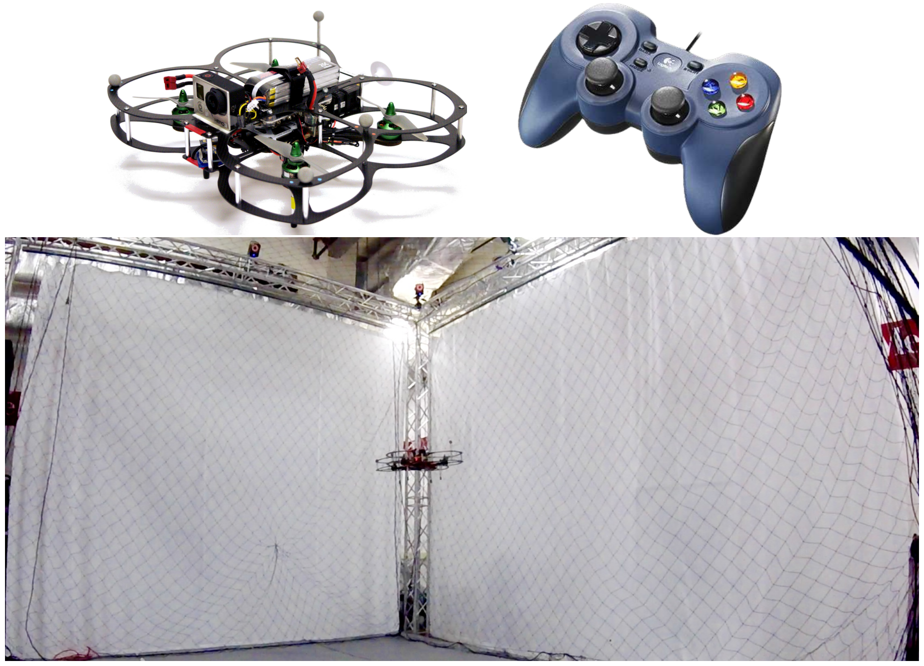


Figure 3.7: The quadrotor and joystick used in the experiment (top) and the quadrotor in flight (bottom).

Table 3.1: LWPR parameters for estimating the reward function for both scenarios

D_{init}	7	w_{gen}	0.3
α_{init}	250	w_{cutoff}	0.5
meta	false	penalty	1.0

Table 3.2: Motion primitive library parameters used in the experiments.

$v_{x\text{max}}$	0.5 m/s	$v_{x\text{min}}$	-0.5 m/s	N_{v_x}	101
$v_{z\text{max}}$	0.5 m/s	$v_{z\text{min}}$	-0.5 m/s	N_{v_z}	101
ω_{max}	3 rad/s	ω_{min}	-3 rad/s	N_{ω}	101

3.3.1 Implementation

The LWPR parameters used in this study are provided in Table 3.1. All parameters that are not listed here take on default values provided by [98]. For both scenarios, we bound the rate at which inputs are received to 10 Hz.

The MPL was generated using the discretization and range values provided in Table 3.2. We select velocity bounds that are conducive for non-aggressive maneuvers for this study and leave adaptation for aggressive maneuvers as future work. We choose a discretization of 0.01 m/s for the linear and vertical velocities, and 0.06 rad/s (approximately 3.44 deg/s) as the discretization in angular velocity. We find this discretization to be sufficient for the purposes of this study and does not interfere with the operator's intentions. One could choose an arbitrarily large number of motion primitives with finer discretizations, however, we find that this adds to the computational complexity and our choice of granularity does not pose a hindrance to the teleoperation performance.

3.3.2 Metrics

We validate the efficacy of our method versus baseline approaches using *Behavioral Entropy* [99], an online, nonintrusive measure of workload that characterizes operator efficiency. Behavioral entropy characterizes the efficiency of an operator's interaction with a robot and measures the consistency of an operator with respect to a predictive model based on the observed behavior. The implication that skilled or desired behavior is consistent becomes quantifiable in the form of entropy.

For joystick-based continuous-input systems, Joystick Steering Entropy (JSE) [100] is a behavioral entropy technique that uses a Taylor series approximation model. We evaluate our assistive teleoperation approach as compared to no adaptation using this metric, and briefly describe JSE below.

At time t, the error between a second order Taylor approximation and the actual input is evaluated:

$$e_t = u_t - \hat{u}_t$$
$$\hat{u}_t = u_{t-1} + (u_{t-1} - u_{t-2}) + \frac{1}{2}((u_{t-1} - u_{t-2}) - (u_{t-2} - u_{t-3}))$$

where $u \in \mathbb{R}$ is the continuous input. A frequency distribution of the error e_t is constructed and divided into 9 bins. The total steering entropy, Hp, for each trial is given by:

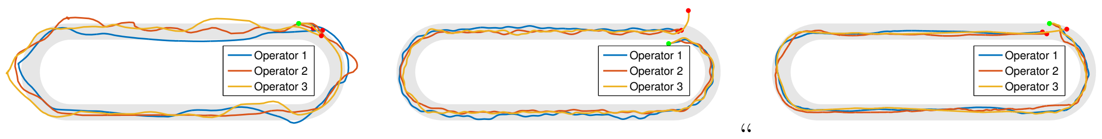
$$Hp = \sum_i -P_i \log_9 P_i. \tag{3.16}$$

A slight modification to the algorithm is made by padding the proportion P_i of each bin by $\epsilon \approx 1e\text{-}6$ in order to avoid asymptotes:

$$P_i = \frac{n_i}{\sum_i n_i} + \epsilon,\ i = 1, \ldots, 9. \tag{3.17}$$

As efficiency increases, the steering entropy decreases accordingly.

We further evaluate the correctness of prediction by computing the *frequency of inputs* over time. We assert that once the performance is sufficient as deemed by the operator, the operator will provide inputs less frequently as the generated motion primitive accurately follows their intent. For mobile vehicles, we compute the frequency of inputs as number of inputs received per second, and for the humanoid robot, we compute the frequency of inputs as the number of inputs received per walking step.



(a) (b) (c)

Figure 3.8: Illustrative odometry results for three operators teleoperating a quadrotor one lap around a simulated racetrack (a) without adaptation, (b) with a low pass filter with $\alpha = 0.5$, and (c) with adaptation. Adaptation results in the smoothest trajectory while low pass filter causes the operator to overcorrect due to smoothing effects.

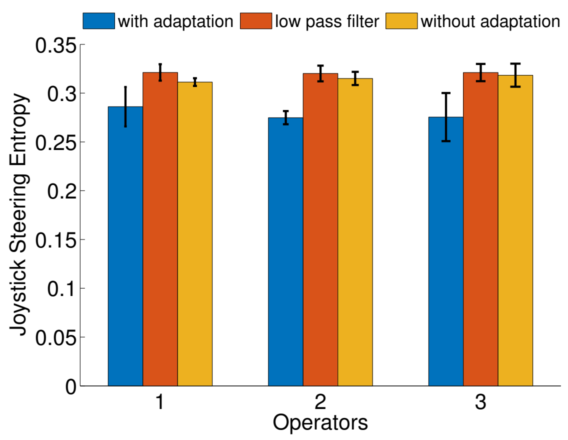
3.3.3 Results

Two scenarios are used to test the proposed framework using a quadrotor vehicle: a racetrack of size $30\,\mathrm{m} \times 10\,\mathrm{m}$ at a height of $1\,\mathrm{m}$ and a lemniscate motion with length of $5\,\mathrm{m}$. The racetrack scenario is used to validate single-intent long-duration adaptation. Due to the size of the racetrack, we perform this task in simulation only. To demonstrate our framework for dynamic intent and the fidelity of our simulation framework, we evaluate the lemniscate motion in both simulation and in the flight arena (Fig. 3.7). For simplicity, we only perform inference over the heading of the robot for all of the subsequent experiments. This is to say, the motion primitive library only contains variations in ω.

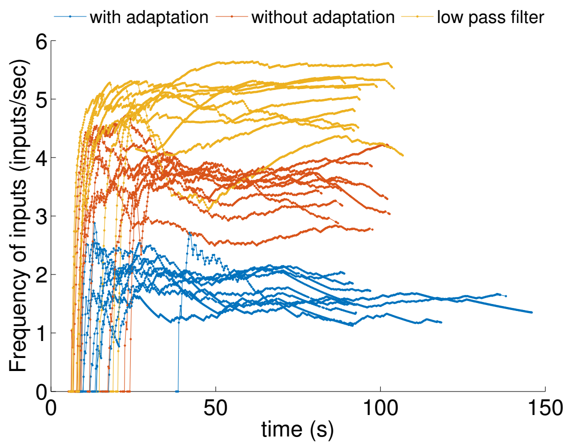
For all of the subsequent experiments, we compare teleoperation results using our adaptation framework to that without adaptation, and we perform additional trials using a exponential moving average filter with weight $\alpha = 0.5$.

Racetrack Results. Operators are asked to teleoperate a simulated quadrotor vehicle using a joystick and follow the racetrack to the best of their ability. As the trajectory that the operator is trying to follow is known, this information can be incorporated into the distance error basis as discussed in Sect. 3.2.2.2. The racetrack is tested with 15 trials (five with adaptation, five without, and five with a low pass filter). All of the trials are randomized in arbitrary order, and are anonymized to the operator. For consistency, we perform the same experiment with three colleagues at Carnegie Mellon University, not including the authors. All operators do not have any prior experience in teleoperating a quadrotor aerial vehicle using this joystick setup.

The resulting trajectory with adaptation, without adaptation, and with the low pass filter is shown in Fig. 3.8. We observe that teleoperation with adaptation produces very smooth trajectories, much smoother than the trajectories without adaptation. Filtered inputs produce trajectories that are much more controlled than without adaptation, but we observe that the trajectories demonstrate periodic behaviors. This is likely due to the lagging effect of the filter: as the inputs are smoothed over time, this creates a lagging

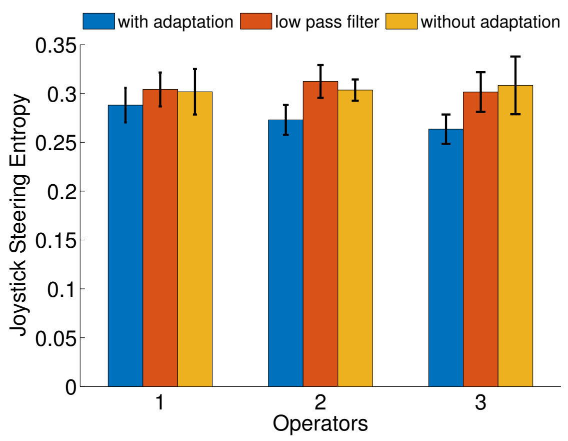


(a)

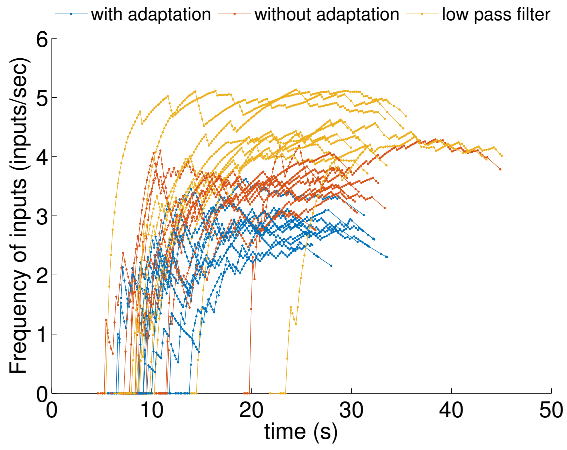


(b)

Figure 3.9: Results of a quadrotor completing one lap around a racetrack, where adaptive teleoperation is compared with a low pass filter with $\alpha = 0.5$ and with no adaptation. (a) Joystick steering entropy averaged over 5 trials for each test case. (b) Frequency of inputs over time.



(a)



(b)

Figure 3.10: Results of a quadrotor completing a lemniscate motion, where adaptive teleoperation is compared with a low pass filter with $\alpha = 0.5$ and with no adaptation. (a) Joystick steering entropy averaged over 5 trials for each test case. (b) Frequency of inputs over time.

effect which causes the operators to over-correct their inputs. We assert that filtering the operator's inputs over time misrepresents their true intent and causes them to provide control-level actions instead of providing only navigational inputs.

We further evaluate our method using Joystick Steering Entropy and frequency of inputs over time, as shown in Fig. 3.9. While filtered inputs sometimes exhibit higher entropy due to the operator's overcorrection, adaptation consistently produced lower entropy than without adaptation, and with a low pass filter, with an average reduction in entropy of

13%. While a reduction in entropy indicates that the operator's inputs are smoother, and consequently result in smoother trajectories, we notice that frequency of inputs with adaptative teleoperation is consistently lower than with filtered input or without adaptation as observed in Fig. 3.9b. This is a key improvement in adaptation as operators provide *less* frequent inputs when the behavior of the robot is aligned with the intended behavior.

Lemniscate results. For this scenario, a specific trajectory was not provided to the operator. Instead, the operator is asked to teleoperate a quadrotor vehicle via joystick to perform a free-hand lemniscate motion in simulation and in the flight arena. As no prior trajectory is provided, predictions are based purely on the hindsight bases as defined in Sect. 3.2.2.2. This experiment uses the incremental adaptive approach outlined in Sect. 3.2.2.3.

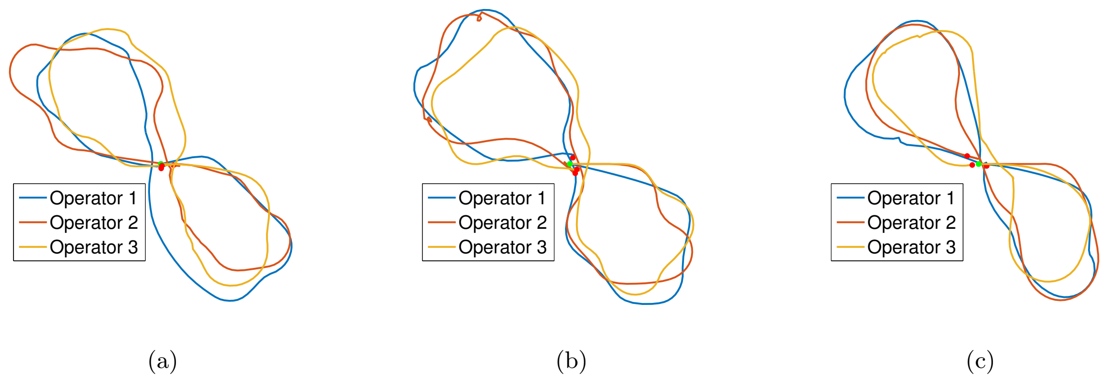


Figure 3.11: Illustrative odometry results for three operators teleoperating a quadrotor with a free-form (i.e without pre-defined trajectory) lemniscate motion (a) without adaptation, (b) with a low pass filter with $\alpha = 0.5$, and (c) with adaptation. Adaptation results in the smoothest trajectory while low-pass filtering causes the operator to overcorrect due to smoothing effects.

We first observe the resulting trajectory as shown in Fig. 3.11. We observe slightly smoother performance with adaptation (Fig. 3.11c) than without (Fig. 3.11a), and similar overcorrecting behavior for the filtered input is again evident. However, we observe the performance with adaptation is not as smooth as what was observed with the racetrack scenario. We attribute this to the fast changes in directional intent that may cause high variance in the incremental prediction.

Joystick steering entropy and frequency of inputs are also evaluated, as shown in Fig. 3.10a. As with the racetrack scenario, we notice that while filtered inputs and teleoperation without adaptation show fluctuating entropies, teleoperation with adaptation shows slightly lower entropy but the reduction is less than expected. We posit that this is a result of the high variance with the incremental prediction, which we will address as future work in Sect. 3.4. Frequency of inputs also is reduced over time and is lower than that with filtered input and without adaptation, however the margin between the average frequency of inputs

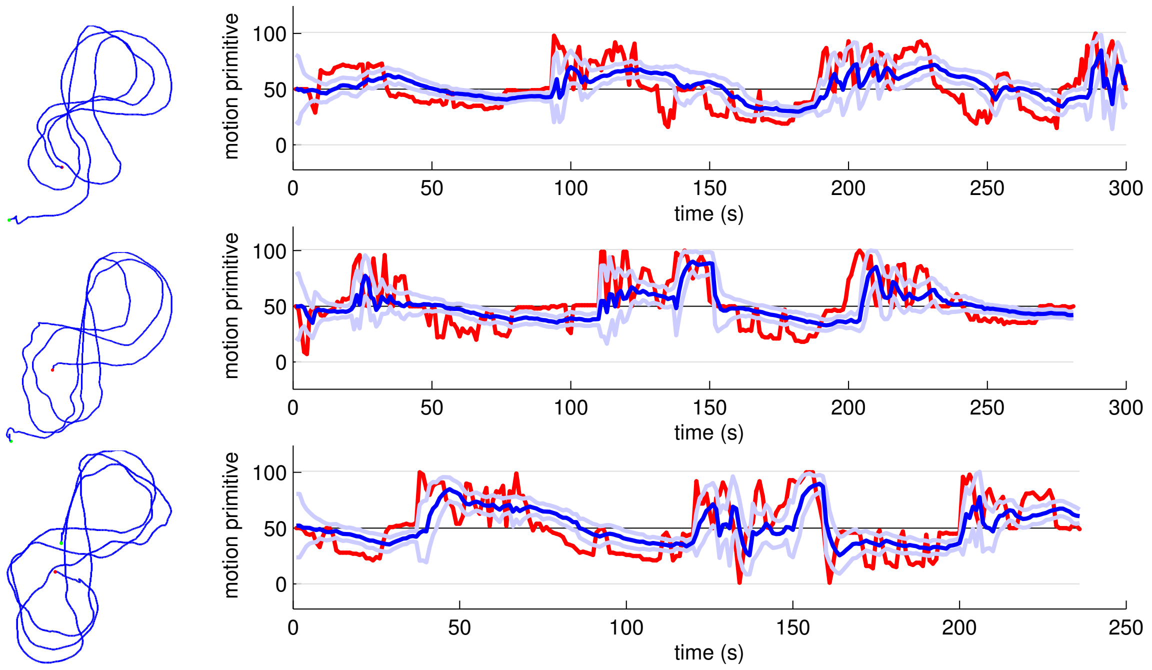


Figure 3.12: Odometry (left) and mean of prediction (right) for three of the experimental trials for the lemniscate experiment. The actual input (red) is compared to the mean of predicted distribution (blue) with light blue highlighting upper and lower bounds based on the covariance. Near regions of rapid operator input changes, variance increases and the mean adjusts to the new prediction.

of adaptive teleoperation and non-adaptive teleoperation is much smaller than that of the racetrack scenario.

The adaptation process is shown for three different runs of the lemniscate with adaptation in Fig. 3.12. The raw input is compared to the predicted mean of the input, where we observe a smoothing of the operator input. In addition, we notice that near regions of rapid operator input changes, variance of the prediction increases and decreases accordingly as the mean adjusts over time.

We now validate the number of motion primitives. From the trajectories in the above scenarios, the choice of discretizing 101 motion primitives seemed reasonable and conducive to good performance. Furthermore, we visualize the sampled motion primitives over time for several trials, as shown in Fig. 3.13a and Fig. 3.13b. For the racetrack scenario, we observe that the number of subsampled motion primitives quickly decreases over time. The lemniscate result is more interesting. We see that as directional intent changes, the number of subsampled motion primitives increases suddenly and converges again.

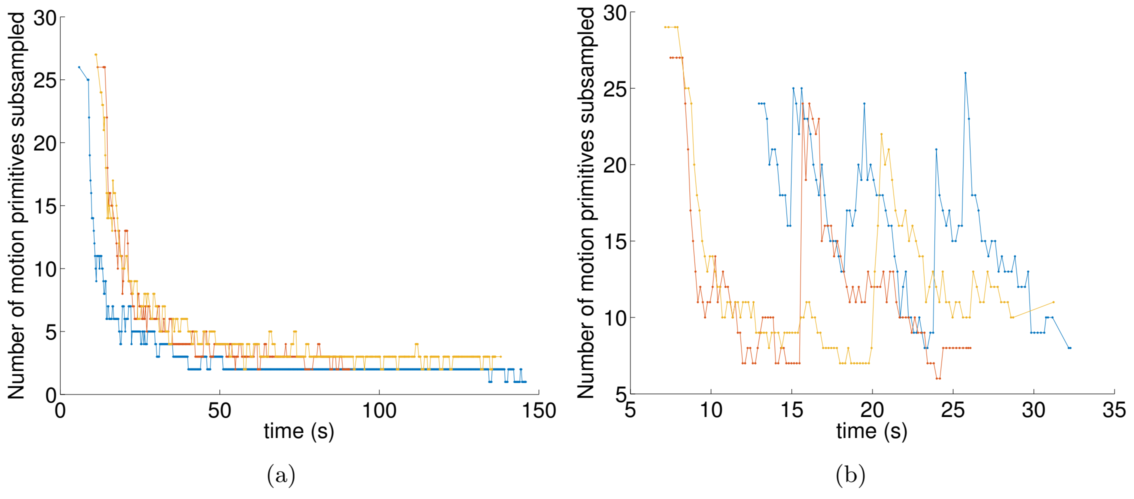


Figure 3.13: Subsampling of motion primitives over time for quadrotor teleoperation in the (a) racetrack and (b) lemniscate scenario. Three random trials are shown for adaptation over angular velocity. This illustrates that a reasonable and non-restricting discretization of the motion primitive is sufficient for constructing the motion primitive library.

3.4 Discussion

This chapter sets the foundation for trajectory-based teleoperation by introducing motion-primitives based teleoperation for ground and aerial vehicles. By moving mobile teleoperation from providing control-level inputs to teleoperation with state-space motions, we eliminate the need for an average operator to require expertise in teleoperating such a vehicle.

Further, this chapter explores an adaptation method for teleoperation that allows assistance of the operator without prior knowledge of an operator model. We experimentally test our framework via teleoperation of a quadrotor air vehicle. The proposed approach demonstrated lower behavioral entropy, indicating increased performance. Furthermore, we demonstrated the correctness of our prediction algorithm to the underlying operator intent by showing a reduction in the frequency of inputs over time.

Teleoperation safety can further be guaranteed by performing obstacle avoidance by pruning motion primitives given a local map. In the next chapter, we introduce reactive collision avoidance for motion primitives based teleoperation.

Chapter 4

Motion Primitives with Reactive Collision Avoidance

This chapter addresses *safety* with respect to motion primitives based HITL control. Given a motion primitive library, the parameterized action set allows efficient collision avoidance by reactively pruning sets of the motion primitive library that leads the vehicle into collision. This chapter discusses and evaluates reactive collision avoidance with motion primitive libraries with respect to two mapping approaches.

Sections of this chapter first appear in [19, 20].

4.1 Introduction

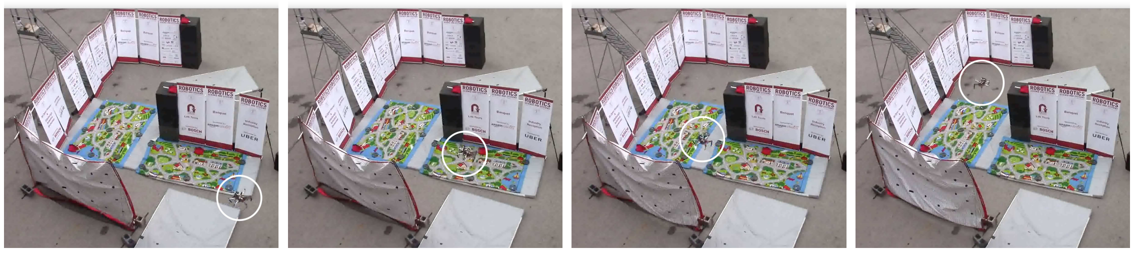
In unknown, cluttered environments, online collision checking approaches enable fast obstacle avoidance. Mobile robots have moved from a paradigm of operating in well-defined environments to complex environments, and require online, real-time mapping and collision checkinng that can operate at a minimum of 10Hz. State-of-the-art methods utilize laser range finders, monocular cameras, and depth cameras to populate a local map for collision avoidance [101–107]. Many prior works in high speed flight exploit the field-of-view (FOV) of stereo cameras for fast collision checking for autonomous flights, including [106, 107], where trajectories are constrained to be inside the FOV of the depth sensor with a max range of 10 m. In [102], trajectories generated by RRT* are checked for collisions directly in the disparity space. Lopez and How [108] presents aggressive flight on a 1.2 kg MAV, achieving a velocity of 3 m/s. These methods achieve fast collision checking by circumventing the need to construct a local map and checking for collisions in the sensor's FOV. This however, limits the range of motions the vehicle can perform. Approaches with local map generation using a laser range finder [105] and a monocular RGB camera [103] have been shown to achieve maximum velocities of 1.8 m/s and 1.5 m/s respectively, but data processing limits the update rate of the local maps.

In this chapter, we present methods of fast reactive collision avoidance by pruning motion primitive libraries with respect two different local map representations: 1) a voxel-based map representation using KD-Trees, and 2) a probabilistic representation using Gaussian Mixture Models, which approximates the underlying distribution from which sensor measurements are sampled and provides an environment model that scales in fidelity with minimal information loss [109].

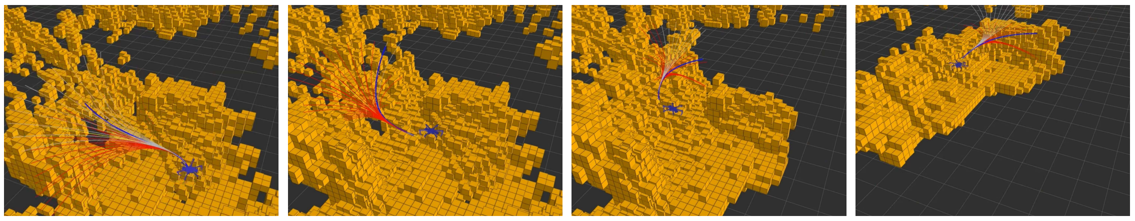
In the experiments for each of the methods, a minimally trained operator is given full control of the vehicle. We show that fast and agile flights can be achieved with a human-in-the-loop while maintaining safety in real time. We perform a series of high speed collision avoidance hardware trials with respect to online-generated KD-Tree local map in both indoor and outdoor environments with untrained operators. In our experiments, the hexarotor attains speeds exceeding 12 m/s and accelerations exceeding 12 m/s^2. We are able to safely avoid obstacles at speeds up to 10 m/s and accelerations of 8 m/s^2. We then demonstrate collision avoidance results of a quadrotor teleoperated through through a cluttered environment at 2m/s with respect to online-generated GMM local map at > 40Hz. Both methods exhibit fast collision checking.

4.2 Approach

We describe reactive teleoperation with respect to teleoperation using motion primitive libraries, first described in Chapter 3. We assume that the state estimates of the vehicle do not drift significantly, such that errors due to state estimation can be considered negligible



(a)



(b)

Figure 4.1: A trial run of a teleoperated vehicle through a tunnel at $2m/s$ with collision avoidance. (a) The vehicle in the tunnel environment (b) The motion primitive library and pruning visualized in the local map generated of the environment. The operator selects infeasible primitives (in yellow), and it is mapped to the closest feasible primitive (in blue). The reactive collision avoidance allows the vehicle to traverse the tunnel without requiring the operator specifying the exact inputs.

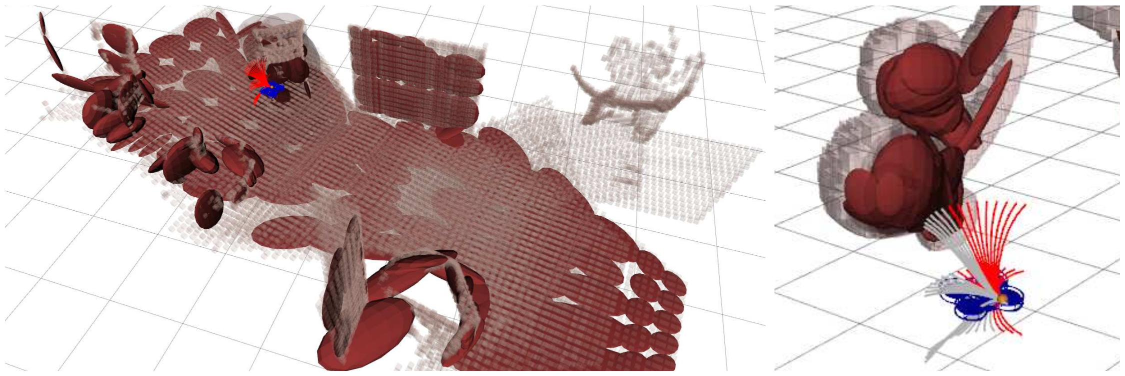


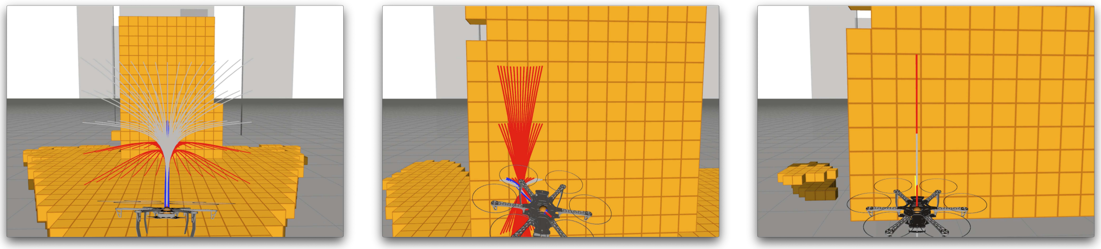
Figure 4.2: A quadrotor navigates a cluttered environment via motion primitive based teleoperation. The dark red ellipsoids represent the Gaussian mixture model components that represent a local map (overlayed over a dense voxel grid representation for visualization). As the quadrotor navigates through the environment, trajectories that intersect with the Gaussian components are pruned (red lines) leaving only the safe, feasible trajectories (grey lines).

in local map generation and collision avoidance and leave considerations of inconsistent state estimates as future work.

Recall, at each regeneration time, a motion primitive library is constructed by discretizing the continuous input along each action dimension, such that each action $\mathbf{a}_i \in A$ is selected from a convex set $A := \{\mathbf{a} \in [\mathbf{a}_{\min}, \mathbf{a}_{\max}]\}$ with size $N_1 \times N_2 \times \cdots \times N_n$ where N_i is the dimension of the space of each input.

At every time step, the operator input is mapped to the closest input in the action space, as defined by the Euclidean norm. A priority queue that minimizes input distance from the selected input $\mathbf{a}_{\text{joystick}}$ to each input in the action space $\mathbf{a}_i \in A$ is used to iterate through the action space until a feasible, collision-free trajectory is found. This results in having the operator input mapped to the feasible motion primitive in the library that is parameterized by the closest discretized action, i.e. $\gamma(\mathbf{a}_i) = \operatorname{argmin} \|\mathbf{a}_{\text{joystick}} - \mathbf{a}_i\|$.

The effect of this pruning algorithm is that the vehicle exhibits natural behavior in the presence of obstacles. If a pillar is in front of the vehicle, then the vehicle chooses a motion primitive some angle away and avoids the obstacle. If a wall is present, then the vehicle will choose linear velocities that gradually decrease until the vehicle is stopped. This behavior is demonstrated in Fig. 4.3.


(a) Vehicle slowing to a stop in the presence of an obstacle.

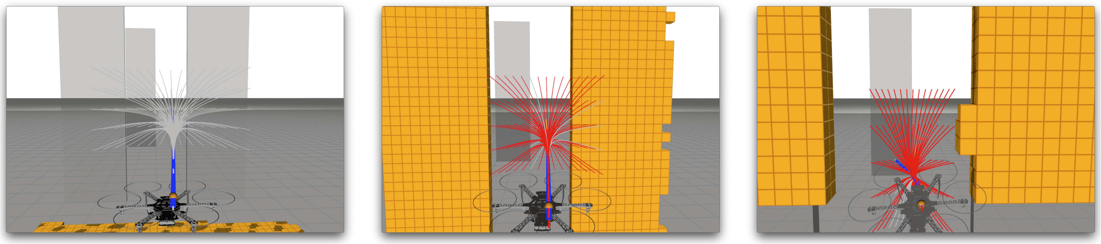

(b) Vehicle slowing down to thread through two pillars.

Figure 4.3: Example demonstration of natural behaviors as a result of the reactive collision avoidance via pruning. The operator selected motion primitive is in yellow. When the selected primitive is unsafe, the closest safe primitive is chosen instead, in blue. Infeasible primitives are shown in red.

4.2.1 Trajectory Pruning for KD-Trees

A spatially consistent local map of the robot surroundings is represented as voxel grids, organized into a KD-Tree. The local map is generated by retaining only the depth sensor measurements obtained at poses that lie in the vicinity of the vehicle's current pose. Given the KD-Tree local map $\mathcal{M}_{\mathcal{L}}$, created in the local frame $\mathcal{L}$, we check for the minimum distance between any point along the trajectory and the surrounding environment. If the minimum distance is above the sum of the vehicle size and an operator specified collision radius, the trajectory is deemed collision-free. Otherwise, the trajectory is eliminated.

Algorithm 2 describes motion primitive library pruning and selection with respect to a KD-Tree local map.

Algorithm 2: Motion Primitive Library Pruning and Primitive Selection with KD-Tree Local Map

```
   Input: Given KD-Tree local map $\mathcal{M}_{\mathcal{L}}$, collision radius $r$, vehicle radius $r_v$
1  Receive input $\mathbf{a}_{\text{joystick}}$
2  Generate the minimum input distance queue according to $d_i = \|\mathbf{a}_{\text{joystick}} - \mathbf{a}_i\|$
3  while $\mathbf{a}_i$ is infeasible do
4      Pop the top action element off of the minimum input distance queue $\mathbf{a}_i$
5      Generate motion primitive $\gamma_{\mathbf{a}_i}$
6      for $\tau = 0 : T$ discretized at some $\triangle t$ do
7          Query $\mathcal{M}_{\mathcal{L}}$ for the closest point $p$ to $\gamma_{\mathbf{a}_i}(\tau)$
8          if $\|p - \gamma_{\mathbf{a}_i}(\tau)\| \geq r + r_v$ then
9              Set $\mathbf{a}_i$ to feasible
10             Set $\gamma = \gamma_{\mathbf{a}_i}$
```

4.2.2 Trajectory Pruning for GMM-based Maps

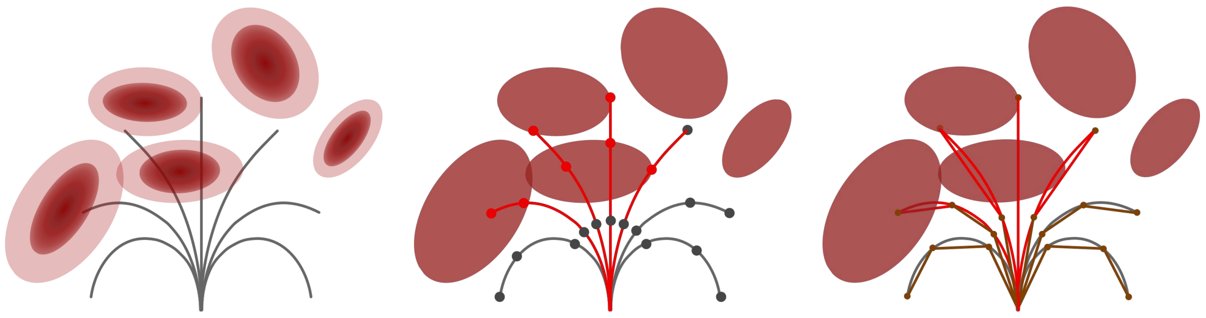


Figure 4.4: A simplified 2D view of the proposed collision avoidance algorithm. A set of motion primitives interacting with a local GMM map (in burgundy) with configuration space inflation (light burgundy) is shown in (a). Each Gaussian component is reduced to its $4\mathbf{\Sigma}$ geometric representation for collision checking, via (b) sampling points along trajectories or (c) creating linear approximations to the trajectory based on curvature and solving for ellipsoid-line intersections. Rejected trajectories are shown in red.

We present two ways of computing collisions given a time-parameterized trajectory and a GMM local map. Each component in the local map can be spatially represented by an ellipsoidal representation given a $\mathbf{\Sigma}$-bound of the distribution. We leverage this geometric property, and assume that each component can be represented as an ellipsoid as given by:

$$f(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu})^\top C(\mathbf{x} - \boldsymbol{\mu}) - 1, \tag{4.1}$$

where $\boldsymbol{\mu}$ is the center of the ellipsoid, $C = \operatorname{diag}(c_1^{-2}, c_2^{-2}, c_3^{-2})$ and c_i are the major axes of the ellipsoid. We represent the configuration space of the vehicle by a sphere with radius r centered at the robot's geometric center. Then, we transform the local map to incorporate the configuration space by inflating the major axes:

$$f(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu})^\top D(\mathbf{x} - \boldsymbol{\mu}) - 1, \tag{4.2}$$

where $D = \operatorname{diag}\left((c_1 + r)^{-2}, (c_2 + r)^{-2}, (c_3 + r)^{-2}\right)$. For sufficiency, we take the ellipsoid defined by the $4\boldsymbol{\Sigma}$ probability bound of each Gaussian component, which provides 99.95% Chi-squared probabilistic coverage of the underlying point density.

Suppose a trajectory is given by $\mathbf{x}(t) = \gamma(t)$, where $\gamma(t)$ is a time-parameterized function with t defined over some interval $t \in [t_0, t_f]$, and $\mathbf{x}(t) = [x(t), y(t), z(t)]^\top$. Then, the ellipsoid-trajectory equation becomes:

$$f(t) = f(\mathbf{x}(t)) = (\mathbf{x}(t) - \boldsymbol{\mu})^\top R^\top D R(\mathbf{x}(t) - \boldsymbol{\mu}) - 1 \tag{4.3}$$

$$f(t) = (\mathbf{x}(t) - \boldsymbol{\mu})^\top A(\mathbf{x}(t) - \boldsymbol{\mu}) - 1, \tag{4.4}$$

where $R \in \mathbb{R}^{3\times 3}$ is the rotation matrix to transform the local trajectory into the frame of the mixture component. An intersection or collision occurs when $f(t) \leq 0$.

For arbitrary trajectories $\mathbf{x}(t)$, no analytic solutions exist to Eq. (4.4) unless $\mathbf{x}(t)$ is affine. In the following subsections, we present two algorithms for collision checking for arbitrarily complex trajectories and provide a brief discussion on computational complexities.

4.2.2.1 Sampling based Collision Checking

Instead of computing an analytic solution to Eq. (4.4), a simple check would be to sample points along each trajectory. For M Gaussian mixture components, N local trajectories, and S samples per trajectory, the computational complexity would be $\mathcal{O}(MNS)$. This approach is delineated in Algorithm 3.

4.2.2.2 Piecewise Affine Trajectory Approximation

If the trajectory is sufficiently smooth, one can generate piecewise affine (PWA) approximations to the trajectory using heuristics. For each trajectory, suppose s segments of affine approximations sufficiently approximate the trajectory. Then, over each segment, the affine approximation $\mathbf{x}_s(t) = \mathbf{a}_s t + \mathbf{b}_s$ with $\mathbf{a}_s, \mathbf{b}_s \in \mathbb{R}^3$ and $t \in [t_{s-1}, t_s]$ can be analytically solved in the frame of each Gaussian component.

The ellipsoid-line equation using Eq. (4.2), in the frame of the gaussian component, can

```
Algorithm 3: Collision Checking with GMM Local Map via Sampling
  Input: Given $M$ Gaussian mixture components, $N$ local trajectories, $S$ samples per
         trajectory
1 Discretize time interval $[t_0,\, t_f]$ into $\mathbf{t} = \{t_i\}$, $i = 1, \dots, S$ s.t. $t_i \in [t_0,\, t_f]$
2 for $n = 1 : N$ trajectories do
3     for $m = 1 : M$ Gaussian components do
4         Obtain the eigenvector matrix $R_m$, centers $\boldsymbol{\mu}_m$
5         compute $A_m = {R_m}^\top D_m R_m$ where
           $D_m = \mathrm{diag}\left((c_{m1}+r)^{-2}, (c_{m2}+r)^{-2}, (c_{m3}+r)^{-2}\right)$
6         for each $t \in \mathbf{t}$ do
7             Query point at time $t$: $\mathbf{x}_s = \mathbf{x}(t)$
8             if $f(\mathbf{x}_s) = (\mathbf{x}_s - \boldsymbol{\mu}_m)^\top A_m (\mathbf{x}_s - \boldsymbol{\mu}_m) \leq 1$ then
9                 Reject trajectory and increment
```

be written as:

$$f(\mathbf{x}_s) = (\mathbf{x}_s)^\top D(\mathbf{x}_s) - 1 \tag{4.5}$$

$$f(t) = (\mathbf{a}_s t + \mathbf{b}_s)^\top D(\mathbf{a}_s t + \mathbf{b}_s) - 1, \tag{4.6}$$

Without loss of generality, a trajectory can always be transformed into the frame of the mixture component such that D is diagonal. Collisions are found via solutions to

$$0 = \left(\frac{a_1^2}{c_1^2} + \frac{a_2^2}{c_2^2} + \frac{a_3^2}{c_3^2}\right) t^2 + 2\left(\frac{a_1 b_1}{c_1^2} + \frac{a_2 b_2}{c_2^2} + \frac{a_3 b_3}{c_3^2}\right) t + \left(\frac{b_1^2}{c_1^2} + \frac{b_2^2}{c_2^2} + \frac{b_3^2}{c_3^2} - 1\right) \tag{4.7}$$

With the assumption that the trajectory does not begin inside a Gaussian component.

For M components, N trajectories, the number of segments is dependent on the curvature of the trajectory. The computation complexity would be $\mathcal{O}(MNS_n)$, where $S_n \leq S$, $n = 1, \dots, N$ such that the worst case complexity collapses to that of the sample based approach (with S samples per trajectory). This approach is delineated in Algorithm 4.

For forward-arc motion primitives, the curvature of the trajectory is correlated with the angular velocity that is used to generate the motion primitive. As such, our heuristic for generating the number of segments is defined as follows:

$$S_n = \lceil 1 + k\,|\omega_n| \rceil \tag{4.8}$$

where $k = 3$ is empirically chosen and $\lceil \cdot \rceil$ is the ceiling operator.

Algorithm 4: Collision Checking with GMM Local Map via PWA Trajectory Approximation

```
Input: Given $M$ Gaussian mixture components, $N$ local trajectories
1 for $n = 1 : N$ trajectories do
2     Heuristically discretize trajectory into $S_n$ segments
3     Compute $S_n$ affine approximations
4     for $m = 1 : M$ Gaussian components do
5         Obtain the eigenvector matrix $R_m$, centers $\boldsymbol{\mu}_m$, and transform $\mathbf{x}$ into the frame of
            the Gaussian mixture
6         for each $s = 1 : S_n$ do
7             Solve Eq. (4.7) and denote solutions as $t^*_{1,2}$
8             if $t^*_{1,2} \in [t_{s-1}, t_s]$ then
9                 Reject trajectory and increment
```

4.3 Experiments and Results

4.3.1 Implementation

4.3.1.1 KD-Tree

We experimentally evaluate our proposed approach on a 3.8 kg hexarotor that fits within a 20 cm $\times$ 60 cm $\times$ 80 cm volume (Fig. 4.5a). The hexarotor has an average flight time of 7 min and a power to weight ratio of 3. Two Intel RealSense D435 depth cameras are used for mapping: one facing forwards and one facing upwards at a 45 degree angle, which aids obstacle avoidance while accelerating forwards. A downward facing Matrix Vision mvBlueFOX-MLC200w is used as the RGB camera input for VINS-Mono and the NVIDIA Tegra TX2 is used for computation. The hexarotor uses a cascaded PD control architecture as in [86] with jerk and snap references used to compute feedforward angular velocities and accelerations.

In order to make KD-Tree local mapping online feasible, we downsample each measurement into an occupancy grid with a fixed voxel size, and only register the voxel centers of the occupancy grid in the map. This results in a reduced fidelity local map, resulting in approximately 10000 points in the local map in total. Details of map construction can be found in [19].

Motion primitives are generated with an angular velocity bound of 2 rad/s. There are 25 discretizations for the v_x action, 11 for ω, and 5 for v_z for a total of 1375 motion primitives generated per trajectory iteration. Trajectories are generated at 25 Hz, and the local map is updated at 30 Hz. We choose the maximum velocity of the motion primitives to be such that the vehicle can always safely stop given a known constant maximum acceleration, known sensor range and known sensor and mapping rates. Since our vehicle has a power to weight ratio of 3, we assume a conservative maximum acceleration at 6 m/s^2, and assume a worst case sensor and mapping rate of 10 Hz. With a sensor range of 10 meters, this allows for a maximum motion primitive velocity of 10.37 m/s.

Table 4.1: Experiment descriptions and parameters. $\mathbf{T}$ is the duration of the motion primitive, $\mathbf{v}_x^{\max}$ is the maximum desired speed, $\omega^{\max}$ is the maximum yaw rate, and r is the collision radius. $\cdot^*$ denotes motion primitive duration increased adaptively as a linear function of the desired velocity change.

Experiment	**Description**	$\mathbf{T}$ (s)	$\mathbf{v}_x^{\max}$ (m/s)	$\omega^{\max}$ (rad/s)	r (m)
Outdoor-A,B	Aggressive teleoperation with collision avoidance outside	2.0	5.0	1.0	0.8
Indoor-A,B	Aggressive teleoperation with collision avoidance in a dimly lit garage	1.5	3.0	1.5	0.4
Outdoor-C	High speed, aggressive teleoperation with collision avoidance outside	1.3*	7.0	2.0	0.5
Outdoor-D	High speed, aggressive teleoperation with collision avoidance outside	1.5*	10.0	2.0	0.2

4.3.1.2 GMM

We fit GMMs over depth sensor scans online using an Intel i7-6700K CPU in real-time. For the KD-Tree representation, instead of using an occupancy grid KD-Tree, we create a KD-Tree directly with the incoming depth scans, as the GMM approach is capable of incorporating the incoming depth scans in its representation. We use 1/4 resolution of the original scan resolution, reducing each incoming data set to 19200 data points. Each KD-Tree and GMM local map is built over the downsampled scan with an average of 14 frames per local map, resulting in a maximum of 249600 points per local map. Note that this map size is much larger than the map generated than the map used on the hardware experiments described in the KD-Tree method. A new KD-Tree is created at each iteration using the raw data using a discretization of 0.1m. The GMM local map generates 60 components per frame, which results in approximately 300 components per local map. Each reduced local map contains approximately 30–40 components. Details of map construction can be found in [20].

Motion primitive libraries contain 155 motion primitives, using 31 linearly spaced angular velocities $\omega \in \{-3, 3\}$rad/s, 5 linearly spaced vertical velocities $v_z \in \{-1, 1\}$m/s, and limit the vehicle to a maximum linear velocity of 2m/s. We assume a configuration radius of 0.5m.

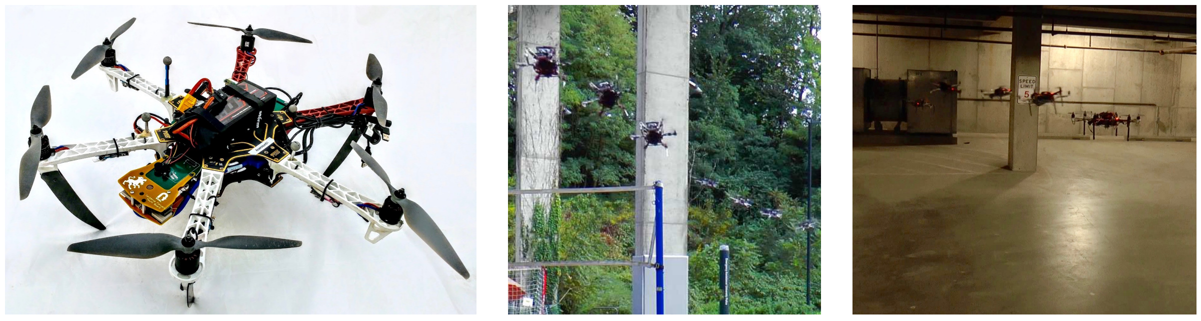

(a) (b) (c)

Figure 4.5: (a) The hexarotor vehicle used for aggressive flights in (b) outdoor environments and (c) a dimly lit garage. The overlays of the vehicle positions over time depict the trajectories the vehicle took to avoid pillars in its way.

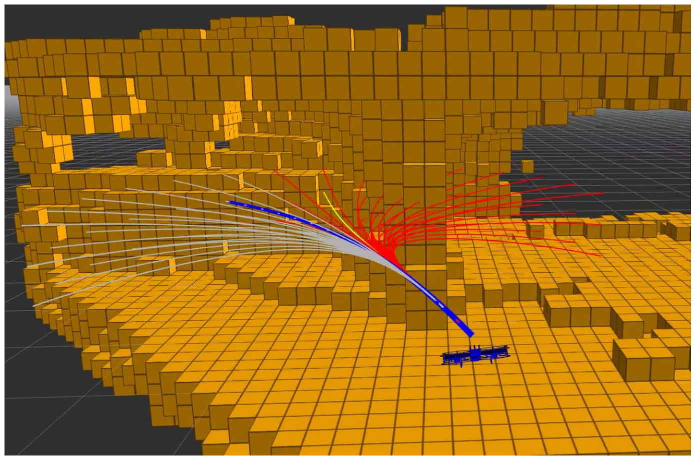


Figure 4.6: A snapshot of the map and motion primitive library during an indoor hardware experiment when the user selected motion primitive (yellow) is not chosen to avoid a collision.

4.3.2 Results

4.3.2.1 KDTree

We conduct seven aggressive flight experiments for KD-Tree based local map, including four outdoor experiments over an area of 40 m × 20 m with obstacles over grass (Fig. 4.5b), and two indoor experiments in a dimly lit garage (Fig. 4.5c). The experiments are described in Table 4.1.

Performance results. Experiments **Outdoor-A,B,C,D**, and **Indoor-A,B** stress test our collision avoidance algorithm at high speeds while maintaining aggressiveness in the commanded trajectories. In both environments, the operator repeatedly tries to fly the vehicle at the maximum speed, as indicated in Table 4.1, into an obstacle. Figure 4.7 shows the speeds and accelerations attained during the six experiments. The vehicle successfully reaches speeds of 10 m/s and accelerations of 8 m/s^2 in the outdoor environment and speeds of 3 m/s and accelerations of 5 m/s^2 in the indoor environment. Figure 4.7 also shows regions where the operator's selected motion primitive would bring the vehicle closer than r meters to an obstacle. In all such cases, the operator's trajectory is pruned and a collision-free trajectory is selected. Figure 4.6 shows an example instance of motion primitive pruning to avoid a collision, along with the local map generated by the vehicle.

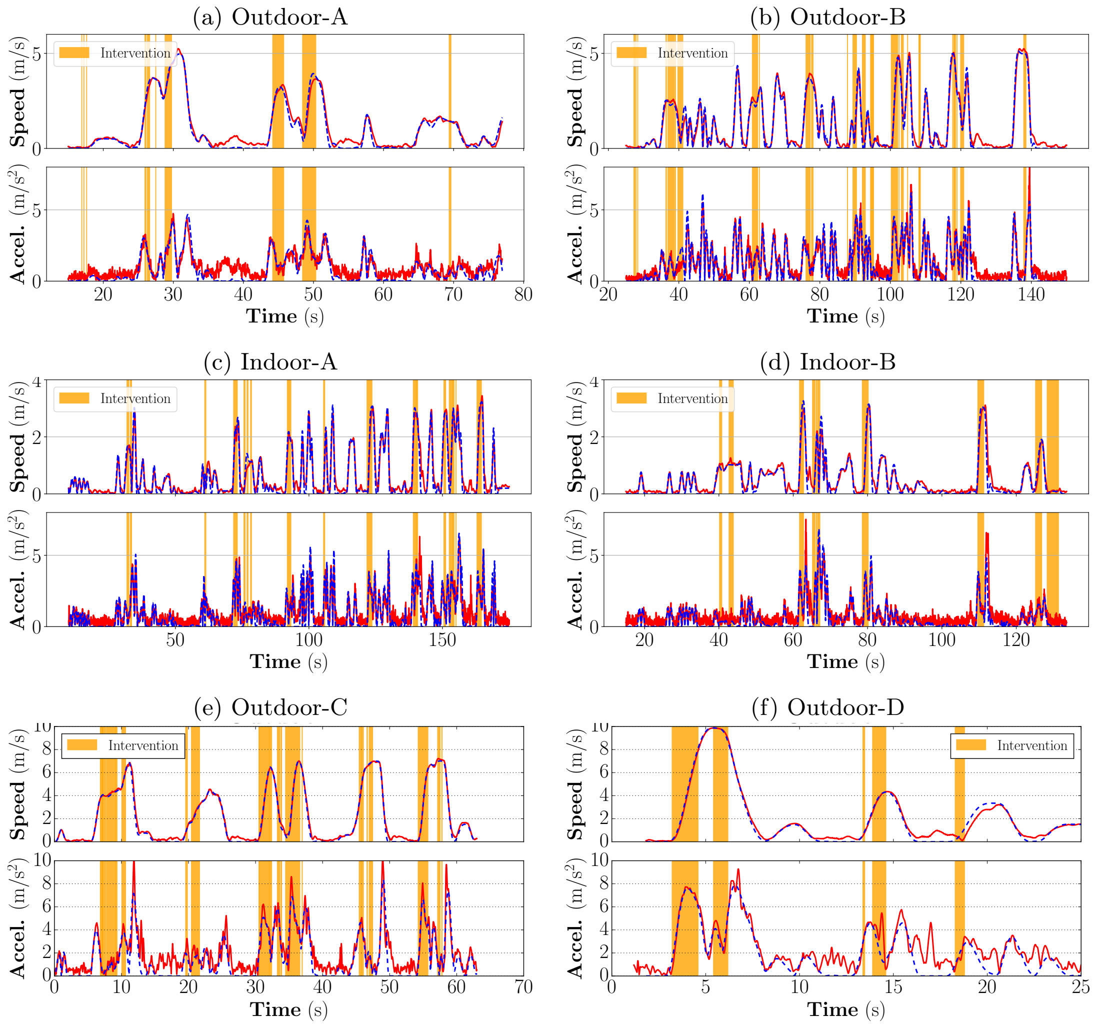


Figure 4.7: Desired (dashed blue) vs. estimated (solid red) speed and acceleration achieved during our six collision avoidance experiments. During intervention, reactive collision avoidance was triggered in order to keep the vehicle safe. The system accurately tracks trajectory references and avoids obstacles while reaching speeds of up to 10 m/s, and accelerations up to 8 m/s^2.

Timing results. Table 4.2 shows that our trajectory generation and pruning time is faster than the user input rate (10 Hz) and that our map generation time is faster than the sensor input rate (30 Hz), enabling real time operation. Furthermore, Table 4.3 indicates that the in-flight computational footprint of the proposed system is less than 75% of the available onboard capacity.

Further, our timing results for trajectory generation and trajectory pruning shows, on average, approximately 15-50ms to generate and prune a motion primitive library of size 1375, and a map size containing approximately 10000 points. This results in approximately a pruning time of $0.01 - 0.036$ms given the KD-Tree map size.

Table 4.2: Execution time (ms) and std. dev. per iteration for safe teleoperation using a local map containing 10000 points.

Experiment	Trajectory Generation	Trajectory Pruning	Local Map Generation
Outdoor-A	0.47 ± 0.072	29.37 ± 24.03	5.72 ± 5.54
Outdoor-B	0.48 ± 0.071	23.65 ± 15.42	5.84 ± 7.95
Indoor-A	0.50 ± 0.125	8.34 ± 5.19	11.67 ± 14.83
Indoor-B	0.51 ± 0.219	15.73 ± 7.23	12.71 ± 16.78

Table 4.3: CPU usage (%, out of a total available 600%) and std. dev. on a 6 core NVIDIA TX2

Experiment	Motion primitive teleoperation	Total including state estimation, control and mapping
Outdoor-A	32.92 ± 17.59	395.72 ± 43.53
Outdoor-B	26.70 ± 19.98	359.89 ± 60.87
Indoor-A	57.38 ± 21.25	450.40 ± 43.14
Indoor-B	58.76 ± 24.09	456.47 ± 36.83

4.3.2.2 GMM

We evaluate the proposed algorithm in simulation in a cluttered environment without dynamic obstacles as shown in Fig. 4.8.

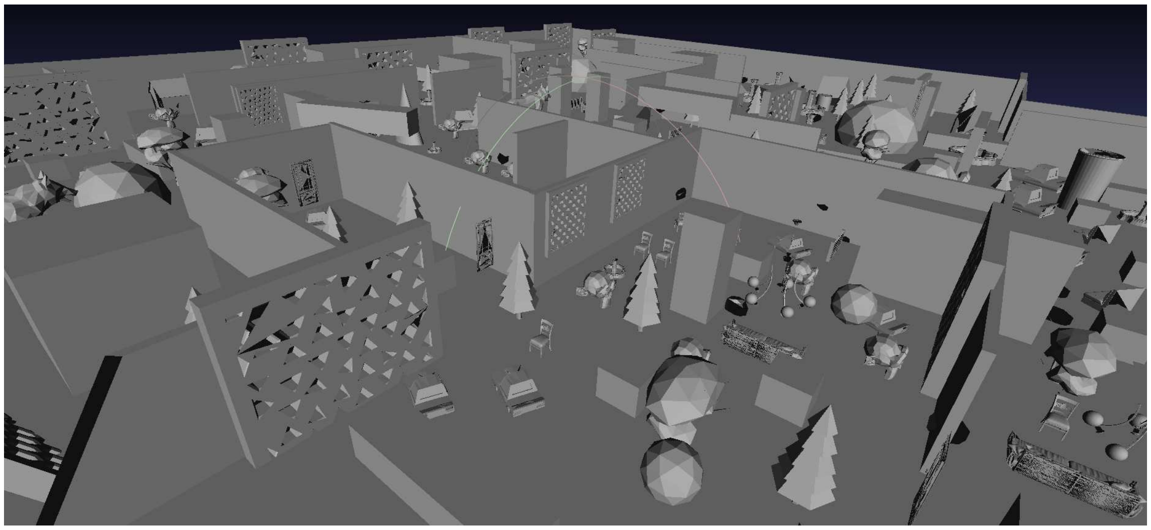


Figure 4.8: The cluttered environment.[1]

Safety results. Safety is evaluated using the minimum distance of the vehicle pose to its surroundings. We use a dense pointcloud representation as a baseline and query the radius of free space around the vehicle at each iteration. In Fig. 4.9, two 6-8 minute example trials are shown. Throughout each trial, the vehicle's configuration space, denoted in blue, is contained within the free space around the vehicle, denoted in grey, indicating that the vehicle is safe at all times.

Timing results. We analyze timing and memory efficiency of GMM local maps as compared to KD-Tree local maps.

We observe that each GMM local map can be generated in approximately 22.89ms on a single CPU (Fig. 4.10, Top). To store the same amount of data, KD-Tree requires approximately 41.92ms. GMM based local map is agnostic to the number of frames in the local map; as new components only need to be appended to the current local map. In contrast, a new KD-Tree needs to be generated with each new sensor measurement and scales poorly over increasing amounts of data.

Collision checking timing analysis averaged over 10000 primitives is shown in Fig. 4.10 (Bottom). For each motion primitive, collision check is performed every 0.1m, resulting on average approximately 20 points per primitive. We observe 0.737ms for KD-Tree based

[1]Available at: https://github.com/vibhavg/simulation_environments

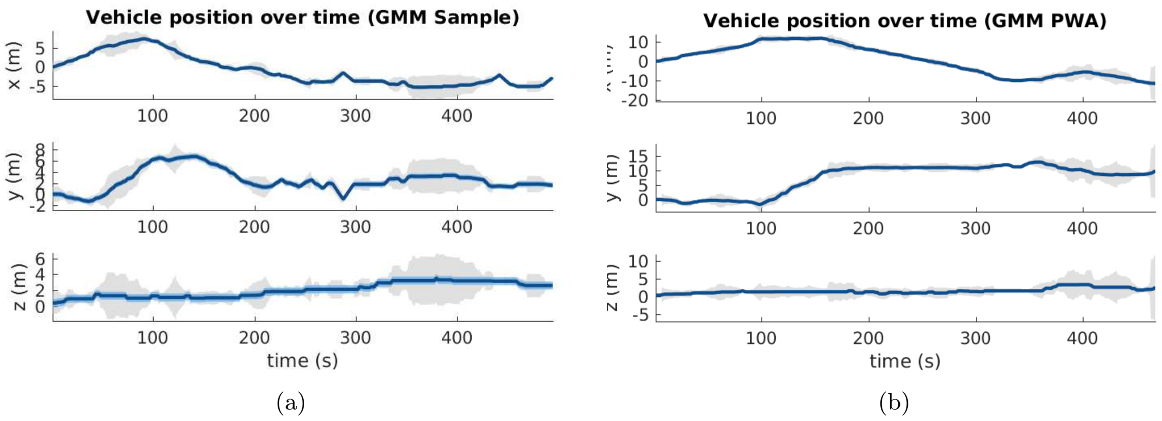


Figure 4.9: A visualization of free space around each vehicle vs. configuration space for example trials with (a) GMM local map with sampling, and (b) GMM local map with PWA approximations based collision avoidance. The blue line denotes the pose of the vehicle and the light blue shading denotes the configuration space of 0.5m.

collision check per motion primitive, 0.204ms for GMM local map with sampling-based collision check per motion primitive, and 0.150ms for GMM local map with PWA motion primitive approximation. Sufficiently representing the local map using a low number of components contributes to significant speed-ups over KD-Tree queries.

Note that here, we are using a KD-Tree generated using the raw data scan, rather than a downsampled occupancy-grid approach. With a KD-Tree map size of 25× the size of the previous KD-Tree map, the query time per primitive significantly increases. This is observed in the per trajectory timing, as compared to the results presented in the previous section.

4.4 Discussion

This chapter extends motion primitive libraries to reactive collision avoidance by pruning the available future motions in anticipation to obstacles in various local map representations. We presented a pruning approach for motion primitive libraries by mapping operator input to the safe set of available actions. The pruning approach is demonstrated in both a voxel-based discretized map representation by way of KD-Trees, and a probabilistic continuous map representation by way of Gaussian Mixture Models, with the latter including two analytic methods for collision checking given arbitrary trajectories, leveraging the geometric properties of Gaussian components.

We demonstrate teleoperation and collision avoidance with KD-Tree on a hexarotor vehicle that allows consistent obstacle avoidance while traveling at speeds up to 10 m/s in an outdoor environment and in a dimly lit garage indoor environment. While this approach yielded significantly low computation times for collision checking, it requires significant map downsampling. We test the GMM based approach in a cluttered artificial environment, and

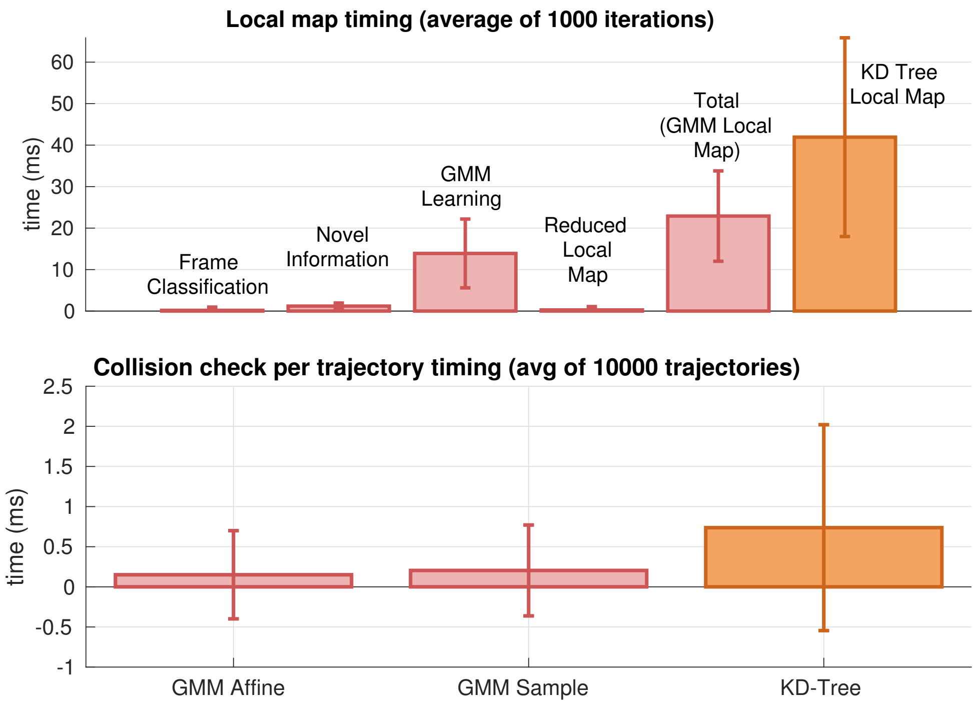


Figure 4.10: Top: Timing analysis for GMM local map generation, averaged over 1000 local maps containing 249600 points. GMM local map takes 22.89ms to learn, whereas KD-Tree local map would take 41.92ms to generate. Bottom: Timing analysis for per motion primitive collision checking with samples and PWA motion primitive approximations, as compared to using KD-Tree representations. GMM methods take 0.25ms for collision checking per motion primitive, whereas KD-Tree takes 0.75ms per motion primitive. Error bars report standard deviation of the mean.

demonstrate that, using the raw depth sensor data for map creation, GMM based mapping reduces the collision checking time as opposed to using the KD-Tree map representation on the raw data. For both approaches, we show successful reactive collision avoidance in teleoperation on computationally constrained platforms on hardware and densely cluttered simulation environments.

Chapter

5

Local Trajectory Generation for HITL Planning

Reactive methods, as discussed in the previous chapter, assists the operator in keeping the vehicle safe, but neglect the operator's intention in doing so. In this chapter, we present a method that generates long horizon, smooth trajectories that follow the operator's intended direction while circumventing obstacles for a seamless teleoperation experience. As trajectories of various lengths can satisfy the same directional objective, we iteratively construct a tree of sequential actions that form multiple trajectories along the intended direction. We show our algorithm on a real-time teleoperation task of a simulated hexarotor vehicle in a dense random forest environment. By doing so, our approach allows operator to achieve the navigation task while requiring less effort than reactive methods.

This chapter first appears in [21].

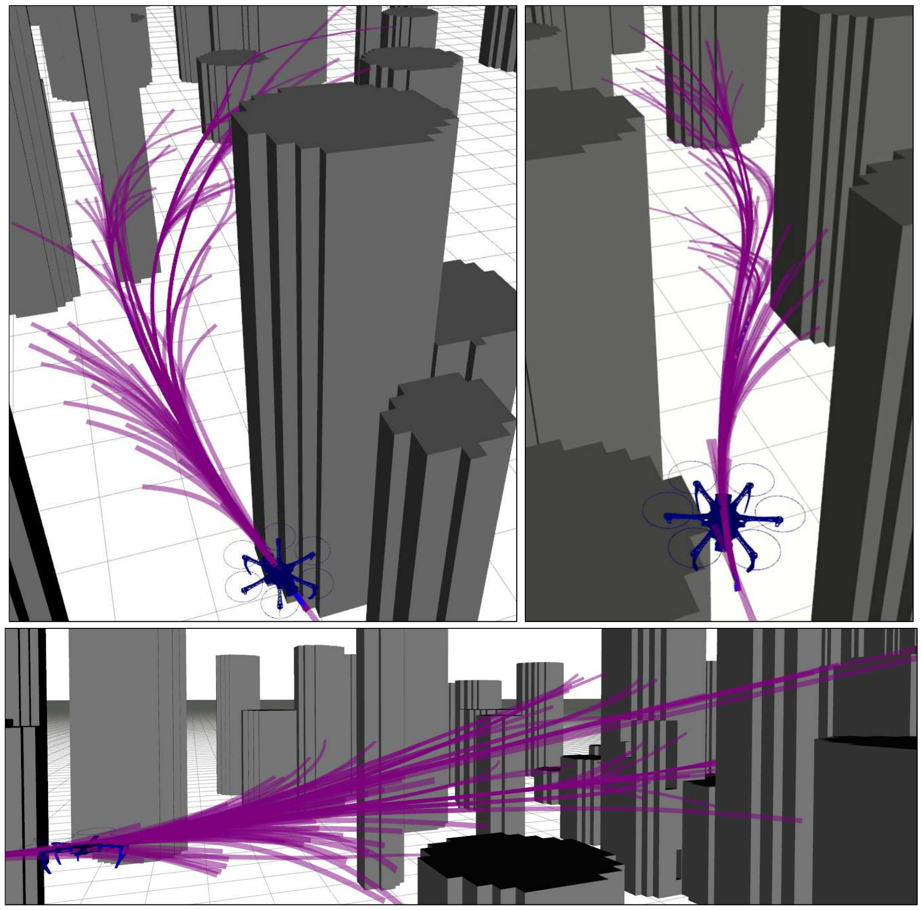


Figure 5.1: A human-in-the-loop controlled hexarotor being teleoperated in a densely cluttered environment, with trajectories that follow the directional intent of the operator while bypassing obstacles in the environment. The proposed method generates long-horizon, smooth trajectories that enables operator navigation of mobile vehicles in unstructured environments with reduced effort, especially in dense environments that require operators to be vigilant to avoid collisions while focusing on the navigation task.

5.1 Introduction

Teleoperation of mobile robots is often used in unstructured environments to achieve a task such as navigation, exploration, or search and rescue. To assist operators, current works mitigate collisions by reactively steering the vehicle away from obstacles using haptic feedback [25] or augmenting control inputs [28]. These approaches do not take into consideration the operator's intentions, and require operators to react fast in environments such as those shown in Fig. 5.1. Instead of pushing the vehicle away from an obstacle that may be along the operator's intended path, we propose generating long horizon smooth trajectories that circumvent obstacles while following the operator's intention. The autonomously generated trajectory reduces the need to engage the operator at a high frequency in dense environments,

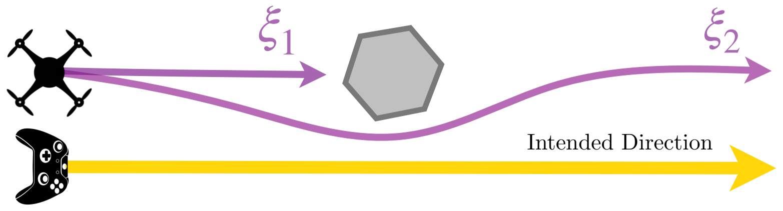


Figure 5.2: This illustration shows two trajectories, ξ_1 and ξ_2, that both characterize a "go straight" motion. In this scenario, ξ_1 can be achieved by taking one action (forward), and ξ_2 requires at least three actions in order to successfully avoid the obstacle (forward, right, left). However, the vehicle will likely end up in a collision state after taking ξ_1 with the obstacle. It is more likely that ξ_2 is the intended trajectory that the operator prefers.

which has been shown to cause fatigue [110].

Generating trajectories that align with the operator's intentions is a difficult task, as a fixed goal may not be known, or the operator has a specific trajectory in mind that they would like to take. In this case, it is prudent to represent operator's intent as a direction instead of an explicit goal in the state space [18]. An example of this is directional intent objective that corresponds to a "go straight" motion, as shown in Fig. 5.2. In this navigation scenario, it is more likely that the operator want to move forward while avoiding the obstacle than to crash into the obstacle, even though the provided input would lead to a crash.

There are a few challenges for this trajectory generation problem with the intent objective as a cost function: 1) The intent objective may be optimized with multiple trajectories with varying lengths; 2) the trajectories needs to remain in a collision-free space while adhering to the intent objective; and 3) the trajectories needs to be generated without specifying a specific state-space goal.

Therefore, we propose a trajectory generation method, Biased Incremental Action Sampling (BIAS), that minimize length-agnostic intent objectives while remaining inside a nonconvex collision-free space. Instead of sampling states in the larger state space, the proposed approach iteratively constructs trajectories by sampling and adaptively increasing the feasible action space as needed, with the cost function acting as a *bias* towards the minimum cost action sequences. The resulting action sequences are translated into sequential motion primitives which form a snap-continuous trajectory.

We showcase this approach in a navigation task of a UAV in a dense random forest as shown in Fig. 5.1. *The contribution of this chapter is a teleoperation framework that, instead of taking reactive measures to assist the operator from collisions, assists the operator by generating predictive long horizon trajectories that align with the operator's directional intent while circumventing obstacles.* The resulting long-horizon trajectories enable navigation tasks to be completed according to operator intent with reduced effort required from the operator. The proposed approach can run online up to 10Hz on a CPU in high-density environments.

5.2 Problem Formulation

We build upon Chapter 3 and introduce trajectories based on sequential motion primitives.

Recall, a motion primitive $\gamma(t)$ is a parameterized function defined over a time interval $t \in [0, T]$ which generates a unique sequence of states given an initial state $\mathbf{x}_0 \in \mathcal{X}$:

$$\gamma_{\mathbf{a},T} : [0, T] \to \mathcal{X} \qquad \mathbf{a} \in \mathcal{A},\ T \in [0, \infty)$$

where $\mathcal{A} = \mathbb{R}^m$ is the action or parameter space. The motion primitive function we will use is first discussed in Chapter 3, Section 3.2.1.3.

The parameter space $\mathcal{A}$ could be constrained due to safety or feasibility. For example, the set of parameters that are dynamically feasible for a mobile robot can be defined as follows:

$$\mathcal{A}_{\text{feas}} = \{\mathbf{a} \in \mathcal{A} \mid \gamma^{(i)}_{\mathbf{a},T}(t) \leq \mathbf{x}^{(i)}_{\text{max}}\} \ \forall\, t \in [0, T] \tag{5.1}$$

where the relevant i-th order derivative (e.g., acceleration) along the trajectory is upper bounded by some known limit of the vehicle $\mathbf{x}^{(i)}_{\text{max}}$. The set of parameters that are safe for a mobile robot can be defined as follows:

$$\mathcal{A}_{\text{safe}} = \{\mathbf{a} \in \mathcal{A} \mid \gamma_{\mathbf{a},T}(t) \in \mathcal{X}_{\text{safe}}\} \ \forall\, t \in [0, T] \tag{5.2}$$

where $\mathcal{X}_{\text{safe}} = \mathcal{X} \setminus \mathcal{O}$ and $\mathcal{O}$ is the set of obstacles. Note that $\mathcal{A}_{\text{feas}}$ and $\mathcal{A}_{\text{safe}}$ may not be convex spaces.

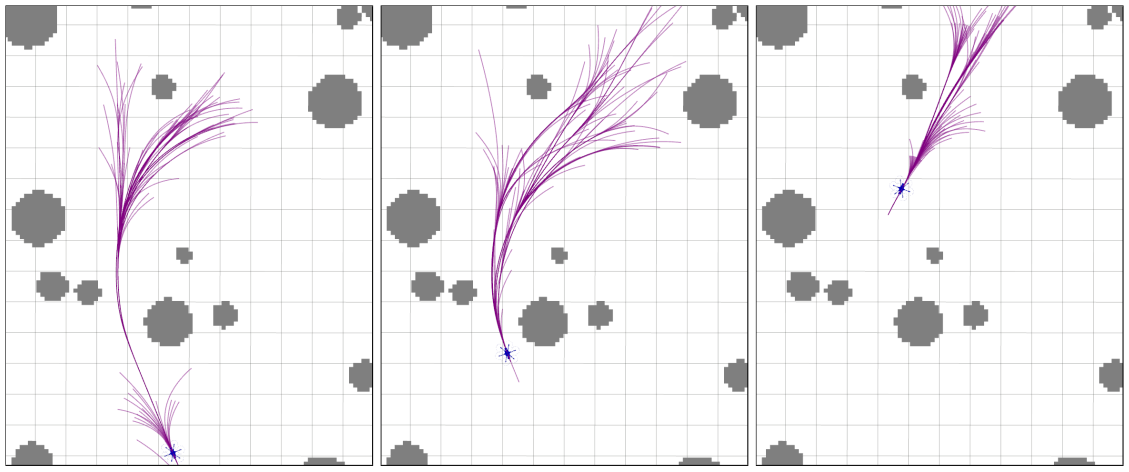


Figure 5.3: Vehicle completing an obstacle avoidance maneuver with trajectories generated using motion primitive trees, shown in the x-y plane. The three sets of trajectories successfully enable the vehicle to complete the maneuver according to the intended direction of motion.

Define a trajectory to be a sequence of N motion primitives:

$$\xi = (\gamma_1, \ldots, \gamma_N) = (\gamma_i)_{i=1}^{N} \tag{5.3}$$

The duration of the trajectory is given by $\mathbf{T} = \sum_{i=1}^{N} T_i$, where T_i is the duration of the i-th primitive. The trajectory function is defined as:

$$\xi(t) = \gamma_i(t - \tau_{i-1}) \qquad t \in [\tau_{i-1}, \tau_i) \tag{5.4}$$

where τ_i is the cumulative duration up to primitive i. The parameters of a trajectory are given by

$$\alpha = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_N] \in \mathbb{R}^{mn}. \tag{5.5}$$

For a trajectory to be feasible and safe, the parameters of each motion primitive must be in the feasible parameter set $\mathbf{a}_i \in \mathcal{A}_{\text{feas}} \cap \mathcal{A}_{\text{safe}},\ i = 1, ..., N$.

A cost function that evaluates a trajectory with respect to the operator's directional intent is given as, $C_{\mathbf{a}^*}(\xi)$, where $\mathbf{a}^*$ is the operator's desired input. The problem statement is then as follows: Given $\mathbf{a}^*$, find a trajectory ξ that minimizes the cost function with respect to the parameters of the trajectory with n-segments in the feasible space:

$$\min_{\alpha \in \mathbf{\Lambda}_{\text{feas}} \cap \mathbf{\Lambda}_{\text{safe}}, n} C_{\mathbf{a}^*}(\xi) \qquad n \in \mathbb{N}^+, \tag{5.6}$$

where $\mathbf{\Lambda} = \prod_{i=1}^{n} \mathcal{A}_i$ represents the action space of a trajectory. The action space scales with respect to the number of segments. The space of all of the number of possible segments in each trajectory sequence, n, forms a discrete set; therefore Eq. (5.6) is a mixed integer program unless n is fixed by choice.

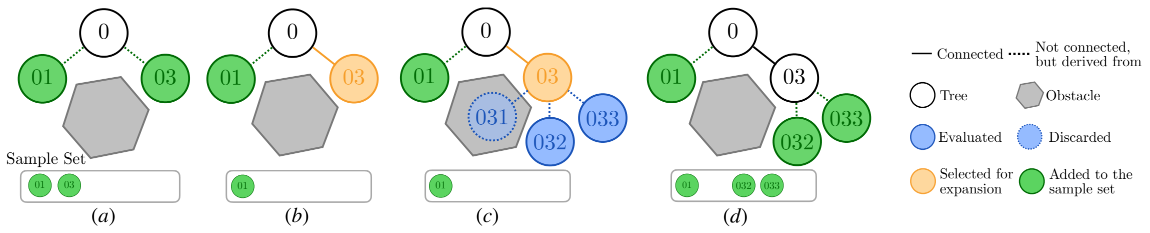


Figure 5.4: A simplified illustration of one iteration of the tree generation process (Algorithm 5). (a) The tree contains the root node (0), and two nodes (01, 03) in the sample set. (b) From the sample set, node 03 is selected and added to the tree. (c) The set of possible successors to 03 (031, 032, 033) are evaluated. (d) From the set of successors, the nodes leading to a collision (031) are discarded, and the others added to the sample set. This process is repeated until a terminating condition is met.

5.3 Approach

5.3.1 Biased Incremental Action Sampling (BIAS)

We iteratively build a tree of sequential actions that minimizes an objective as described in Eq. 5.6. The algorithm is detailed in Algorithm 5, and a simplified illustration of one iteration of the process is shown in Fig. 5.4.

Starting at the root node, the tree is constructed by selecting actions via weighted sampling from the Sample Set $\mathcal{S}$ (the set of possible nodes for expansion). For each node to be expanded, all known feasible actions (successors) are evaluated, and added to the Sample Set. The set of possible actions at each step is known a priori: they are generated by uniformly discretizing[1] the continuous action space $\mathcal{A}$ along each parameter dimension. Let the discretized action set of $\mathcal{A}$ be represented by $\mathcal{B} = \{\mathbf{a}_i\}_{i=1}^{K}$; therefore the maximum branching factor of the tree is K. At each iteration, the feasibility of an action $\mathbf{a}_i$ is evaluated by checking Eq. (5.1) and (5.2). In our implementation, safety of each action is checked with respect to a given map representation, which is discussed in Sect. 5.4.1.

Each node η at depth D represents the sequence of parameters starting from the root node, i.e. $\eta^D = \left(\mathbf{a}^d\right)_{d=0}^{d=D}$. Therefore, the cost function $C\left(\eta^D\right)$ of a node at depth D evaluates the sequence of parameters up to η^D. The cost of each node is used to generate a weighting of the node such that lower cost nodes maintain a higher likelihood to be sampled; e.g., $w(\eta) = \frac{1}{C(\eta)}$.

The Sample Set $\mathcal{S}$ is a set of tuples that contains the node parameters and its associated cost (η, w). At each iteration, J nodes are sampled from $\mathcal{S}$ according to their weights (Line 4). All feasible and safe children of the sampled nodes are then evaluated and added to the Sample Set (Line 9). As elements are added, the Sample Set increases exponentially in size thus causing this set to contain a large number of lower weighted elements. In order to highlight the higher weighted elements, the sampling is limited to an elite set containing the top ρ samples within the Sample Set, and those weights are passed through a softmax function:

$$\sigma(w_i) = \frac{e^{\beta w_i}}{\sum_{j=1}^{N} e^{\beta w_j}} \qquad \beta > 0 \tag{5.7}$$

where β is a tuning parameter. This amplifies the likelihood of the higher weighted elements being sampled. We introduce a cost bound $\bar{C}$ which is updated after every iteration (Line 18). By doing so, this effectively bounds our search to the first local minima encountered.

[1]Instead of sampling from a uniformly discretized action set, one can sample from a non-uniform prior over the continuous action space based on the initial state of the system. Such formulation would allow the samples to be chosen via an informative prior, and possibly reduce the number of nodes to be evaluated. We leave this to future work.

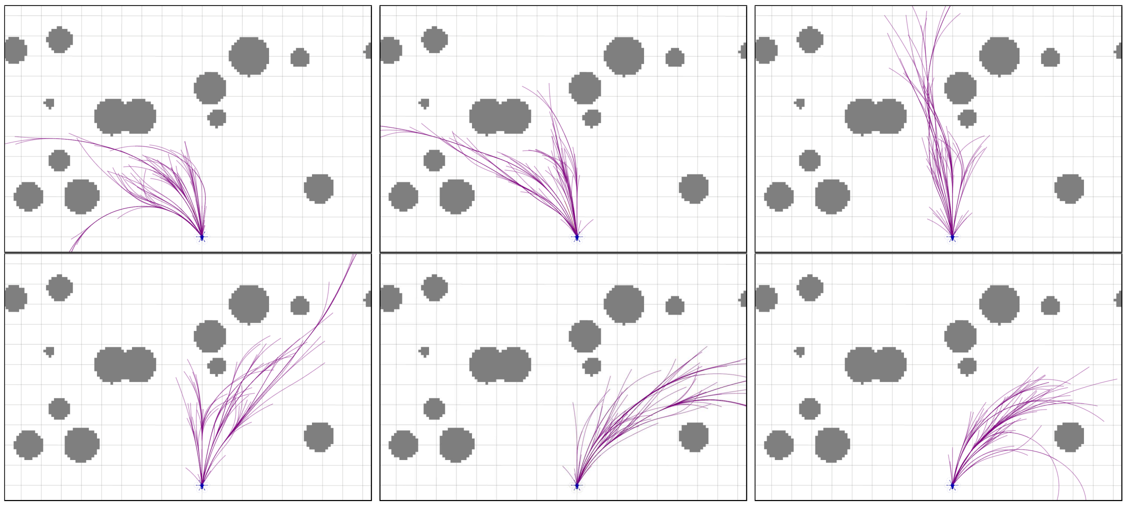


Figure 5.5: Motion primitive trees generated in the x-y plane according to joystick inputs with varying angular velocity in a cluttered environment, showing effective obstacle avoidance while adhering to the directional intent specified by the operator. All trajectories are smooth and continuous up to snap.

5.3.2 Motion Primitive Trees

A motion primitive tree is generated once a new input is received from the operator via the joystick, and the lowest cost trajectory is executed. The resulting tree of actions generated using BIAS is transformed into a motion primitive tree by computing the appropriate

Algorithm 5: Biased Incremental Action Sampling

```
Input: Given a cost function $C(\xi)$, a batch sampling parameter $J \in \mathbb{N}^+$, and an initial
       state $\mathbf{x}_0$
1  Initialize the Sample Set with the root node: $\mathcal{S} = \{(\eta^0, 0)\}$
2  Initialize cost bound $\bar{C} = \infty$
3  while terminating condition not met do
4      Sample $\underline{J} = \min(|\mathcal{S}|, J)$ nodes from $\mathcal{S}$ according to their weights without replacement
5      for each sampled node $\eta_j$ with depth $d$, $j = 1, ..., \underline{J}$ do
6          Add the sampled node $\eta_j$ to the tree
7          Remove the sampled node from the Sample Set: $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(\eta_j, w_j)\}$
8          for all $\mathbf{a} \in \mathcal{B}$ do
9              if $\mathbf{a} \in \mathcal{A}_{feas} \cap \mathcal{A}_{safe}$ then
10                 Generate new node $\eta(\mathbf{a}) = (\eta_j^d, \mathbf{a})$
11                 Evaluate cost of node $C(\eta)$
12                 if $C(\eta) < \bar{C}$ then
13                     Compute weight of node $w(\eta)$
14                     Add node and its weight to the Sample Set: $\mathcal{S} \leftarrow (\eta, w)$
15         Optional: Update cost bound: $\bar{C} = \max_{s \in \mathcal{S}} C(s)$
```

motion primitives parameterized by the actions and an initial state. Each node η in the tree represents a motion primitive that takes the robot from one state to another via a time-parameterized trajectory function γ, generated by our choice of $\mathbf{f}$ with action $\mathbf{a}$. Each successor contains parameters that guarantee a snap-continuous trajectory from the parent node by construction. The root node is seeded with the initial state of the system. Thus, each sequence of nodes η with depth D is an equivalent representation of trajectory ξ with N segments given γ, with $D = N$:

$$\eta^D = \left(\mathbf{a}^d\right)_{d=0}^{d=D} \quad \Longleftrightarrow \quad \xi = (\gamma_i(t))_{i=1}^{i=N} \tag{5.8}$$

The continuous action space $\mathcal{A}$ is the action space of the motion primitive parameters. The successors to each node form a motion primitive library with the initial state as determined by the end state of the current primitive.

The motion primitive tree becomes a set of trajectories given an initial condition. The application of this algorithm to a directional intent function (described in Sect. 5.3.3) enables trajectory generation for a right turning maneuver as shown in Fig. 5.3. Each multi-step primitive trajectory, by construction, naturally becomes an extension of motion primitives based teleoperation discussed in Chapter 3: In the worst case where every action downstream from the one-step action set incurs a higher cost, the algorithm simply returns the one-step motion primitive library, defaulting to the behavior of one-step teleoperation where the operator's selected primitive is carried out until the next iteration.

Trajectory selection policy The existence of many local minimas will necessarily require us to terminate the algorithm once a desirable set of trajectories have been generated. Therefore, the termination criterion can be reaching a max tree size P; or until the Sample Set becomes empty. The algorithm returns a set of trajectories and the vehicle executes the minimum cost trajectory.

5.3.3 Directional Cost Formulation

We present an intent cost function that encodes the desired direction of motion, which are used to generate motion primitive trees as shown in Fig. 5.5. Given a prediction of the most likely input, $\mathbf{a}^*$, the intent cost function can be described as follows. The cost function compares an approximate directional vector between the trajectory $\xi(t)$ and a one-step motion primitive, generated using the most optimal action as given by the intent model. Specifically, the trajectory is evaluated at its end points $\mathbf{x}_0 = \xi(0)$, and $\mathbf{x}_\mathbf{T} = \xi(\mathbf{T})$. A one-step motion primitive γ^* is generated using $\mathbf{a}^*$ with duration $\mathbf{T}$ according to Eq. (3.6), such that $\gamma^*(t) = \gamma_{\mathbf{a}^*,\mathbf{T}}(t)$. The endpoints generated by the desired motion primitive would be given by $\mathbf{x}_0^* = \gamma(0)$, and $\mathbf{x}_\mathbf{T}^* = \gamma(\mathbf{T})$. The cost function is given as:

$$C_{\text{input}_{\mathbf{a}^*}}(\xi) = \|1 - \mathbf{p} \cdot \mathbf{p}^*\| \tag{5.9}$$

$$\mathbf{p} = \frac{\mathbf{x_T} - \mathbf{x}_0}{|\mathbf{x_T} - \mathbf{x}_0|} \qquad \mathbf{p}^* = \frac{\mathbf{x}_\mathbf{T}^* - \mathbf{x}_0^*}{|\mathbf{x}_\mathbf{T}^* - \mathbf{x}_0^*|} \tag{5.10}$$

which shifts the dot product such that $C_{\text{input}} \geq 0$ and remains within $[0, 2]$. The most likely input $\mathbf{a}^*$, uses the most current input of the operator.

User preference and behavior heuristics We augment the intent objective with descriptors of trajectory behavior in order to capture user preference in the qualitative shape of the trajectory. Trajectory behavior describes *how* a particular trajectory is executed. These are adjectives such as *slow* vs. *fast* or *smooth* vs. *aggressive*, which are translated to a value-function that penalizes higher order derivatives such as velocity, acceleration and jerk. For an $(n{+}1)$-segment trajectory comprised of motion primitives each parameterized by actions $\mathbf{a}_0, \mathbf{a}_1, \ldots$, we define and utilize the following behavior functions:

$$C_{\text{Straight}} = \sum_{i=1}^{n} |\omega_i| + \sum_{i=1}^{n} |v_{zi}| \qquad C_{\text{Speed}} = \sum_{i=1}^{n} \frac{1}{\|\mathbf{v}_i\|_2}$$

penalizes nonzero curvature   penalizes trajectory with slow speeds

$$C_{\text{Smooth}} = \sum_{i=1}^{n} |\mathbf{a}_i - \mathbf{a}_{i-1}| \qquad C_{\text{Duration}} = \sum_{i=1}^{n} \frac{1}{T_i}$$

penalizes changes in curvature   penalizes short duration trajectories

The cost functions are linearly combined with a set of user selected weights. A visualization of the effect of these cost functions is shown in Fig. 5.6.

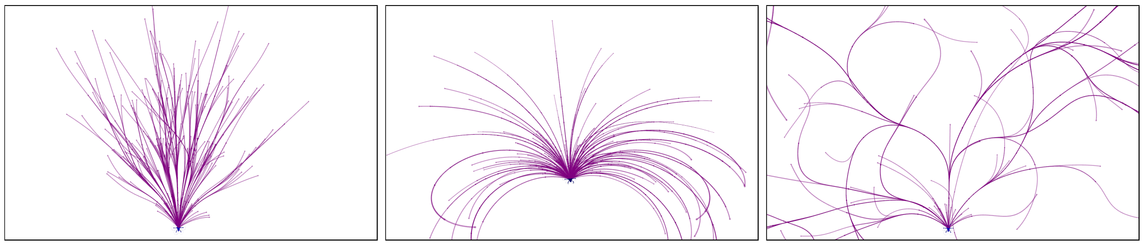


Figure 5.6: Trajectories showing the effects of three behavioral cost functions: C_{straight} (left), C_{smooth} (middle), and C_{duration} (right). These show the effect of each cost function on the shapes of the resulting trajectories. The effect of C_{speed} is omitted as it is not intuitive to visualize in a path.

5.4 Experiments and Results

The proposed method is tested in simulation using a high-fidelity hexarotor model in various density random forest environments using a combination of the intent and behavior

Table 5.1: Trajectory parameters: The 2D action space includes ω and T, and the 3D action space includes ω, T, and v_z.

Parameter	**Min**	**Max**	**Num. discretizations**
Duration T	0.2s	1.5s	5
Angular vel. ω	0.75rad/s	0.75rad/s	15
Z vel. v_z	-0.75m/s	0.75m/s	3
Total size of the motion primitive library			75 (2D), 225 (3D)

Table 5.2: Cost parameters for random forest navigation

Cost function weights		**Sampling Parameters**	
w_{smooth}	0.3	Softmax parameter β	0.5
w_{straight}	0.1	Num. nodes sampled J	2
w_{duration}	0.6	Tree size (Num. nodes P)	100
w_{speed}	0.3	Elite set size ρ	500
w_{intent}	1.8		

cost functions as described in Sect. 5.3.3.

5.4.1 Implementation

The proposed method is implemented in C++ on a CPU (Intel Core 2.20GHz i7-8750H CPU), with 16GB of RAM. 12 threads are allocated in order to support parallel evaluation of the discretized actions (Lines 8 - 17 in Algorithm 1).

We use a global map representation using both a KD-Tree based voxel representation, as well as a signed distance field (SDF) representation. KD-Tree is efficient for a local map representation [19]; However, SDF provide faster queries for global map since it can be processed offline. The method can be readily adapted to local maps instead of global maps. Local maps generated using limited range sensors may require limiting the maximum tree depth, such that the trajectories are within the known map.

Three random forest environments are used in this experiment with varying sparsity, each with size $60\text{m} \times 30\text{m} \times 10\text{m}$. The sparse, medium, and dense random tree forest contains approximately 30, 70, and 120 pillars respectively. The task is as follows: Navigate the simulated hexarotor vehicle from one end of the environment to the other. The operator can choose to take any path they wish to complete the task. The operator is given a third-person follower view of the vehicle which is ensured to be occlusion free such that the operator remains within line-of-sight of the vehicle.

The motion primitive parameters are fixed for all trials (Table 5.1). The linear velocity is directly controlled by the operator using one of the axis of the joystick during operation with the maximum velocity capped at 2m/s. The algorithm-associated parameters are given in Table 5.2. The simulated vehicle has a radius of 0.6m with a collision radius set constant at 0.1m. The discretization size for both KDTree and SDF is fixed at 0.1m.

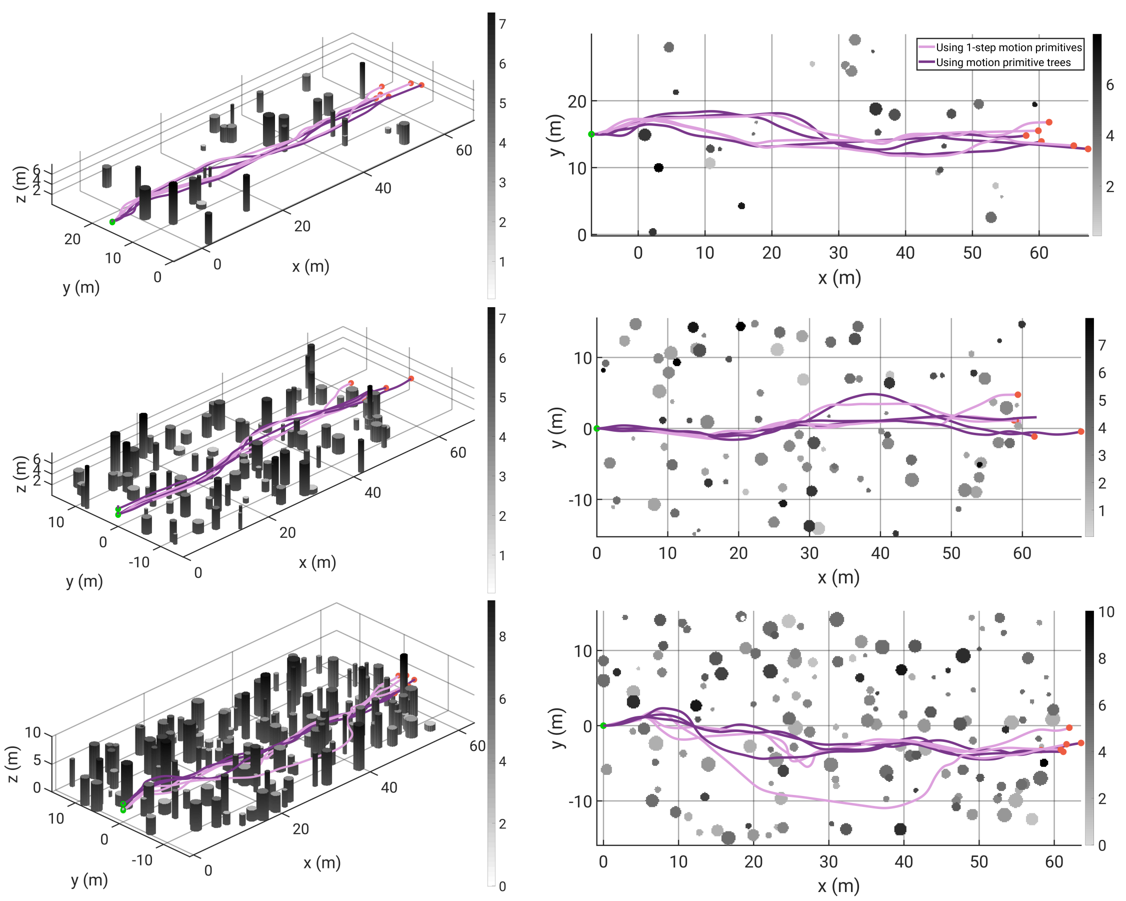


Figure 5.7: Some resulting odometry using reactive motion primitives based teleoperation and teleoperation using the proposed method in sparse (top), medium (middle), and dense (bottom) enviroments for three trials each. The colorbar to the right indicate the pillar heights in meters. Both methods complete sparse environments equally, however the proposed method results in qualitatively smoother trajectories in the dense scenario. Data from these trials are summarized in Table 5.5.

5.4.2 Results

Timing Results. The per-node timing evaluation is provided in Table 5.3. The cost evaluation per node for all cost functions averages 0.0168ms. The collision check time in the densest environment for both SDF and KD-Tree representations are on the order of 10^{-3}∼10^{-2}ms. The collision check time should reduce significantly if using a local map.

The timing results for the trajectory generation process are shown in Table 5.4. Trajectory generation for a 2D tree takes approximately 113.12ms for an average depth of 5, and 300ms for a 3D tree with an average depth of 4. Approximately 9785 and 28906 nodes are evaluated to generate the 2D and 3D tree respectively. To contextualize our result, note that naively generating a fixed size 2D tree with depth 5 and 3D tree with depth 4 would result in $75^5 = 2.373$T and $225^4 = 2.562$B nodes respectively.

Table 5.3: Timing results per node (data from 2.4M node evaluations with the densest random forest environment)

Per node	**Time**
Cost Evaluation	0.0168 ± 0.0664 ms
Collision Check with SDF	0.00269 ± 0.00421 ms
Collision Check with KDTree	0.01351 ± 0.06001 ms
Total with SDF	0.0230 ± 0.0804 ms
Total with KDTree	0.0320 ± 0.0694 ms

Table 5.4: Timing and size data per tree (data from 100 trees generated in the densest random forest environment)

Time	**2D (75 children)**	**3D (225 children)**
Generation	113.12 ± 91.04 ms	299.78 ± 88.46 ms
Selection	1.14 ± 0.95 ms	1.73 ± 7.01 ms
Generation + Selection	116.44 ± 91.40 ms	315.57 ± 61.9 ms
Num. nodes processed	9785 ± 1499	28906 ± 4584
Num. traj per tree	56 ± 42	93 ± 21
Num. iter. per tree	42 ± 25	61 ± 16
Average depth of tree	5 ± 1	4 ± 1

2D trees refer to motion primitive trees generated in the x-y plane.
3D trees refer to those generated in the euclidean space.

Task completion results We compare the proposed method with respect to the previous reactive, one-step teleoperation proposed in [19]. The reactive method parameterizes the operator's control input and generates a single motion primitive. Both methods are evaluated in the sparse, medium and dense random forest environments with five trials each.

The key metric we evaluate is the number of joystick inputs received to complete each task, which represents the *effort* of the operator. This metric is critical to evaluating the operator engagement with the system during teleoperation: If the number of joystick inputs is high, it indicates that the operator has to frequently engage the system to control the vehicle. If the operator does not need to engage as much with the system as much over time, then this implies that the vehicle is following the operator's intended trajectory and therefore the operator does not feel the need to control the vehicle. We observe the task completion time as well as the smoothness of the trajectory via the jerk integral. The results are reported in Table 5.5. A subset of the resulting trajectories are shown in Fig. 5.7.

The number of joystick inputs of each run ranges from ∼44 control inputs for the sparse environment to ∼58 control inputs for the densest environment. Contrast this to the reactive method, which utilized ∼192 control inputs for the sparse case, and up to ∼421 control inputs for the densest environment. In the sparse enviroment, the operator has to engage less due to the lack of obstacles. But in the dense environment, the number of inputs rise significantly for the reactive method due to the frequent encountering of obstacles and difficult structures that causes the operator to be highly engaged. The frequency of engagement is significantly reduced using the proposed multi-step method, indicating a reduction in effort required to achieve the same navigation task. Only 25% control inputs are required for the sparse environment, and the number of inputs only increased by around 15-20 for the densest environment. This represents a significant reduction operator's effort when navigating through difficult scenarios such as narrow gaps shown in Fig. 5.1.

Lastly, we observe that the completion time for each of the tasks are similar. We also observe that the generated trajectories exhibit a lower jerk profile throughout the task, which indicates that the proposed method generates smoother trajectories than the reactive teleoperation method.

Table 5.5: Comparison of the different density enviroments

Approach	**Sparse**	**Medium**	**Dense**
Time to completion (s)			
MP (single-step)	38.84 ± 1.08	37.06 ± 0.50	37.95 ± 1.91
MPT via BIAS (multi-step, proposed)	39.45 ± 1.43	36.40 ± 1.99	37.55 ± 2.34
Jerk Integral (m^2/s^3)			
MP (single-step)	28.41 ± 2.88	26.72 ± 4.30	50.14 ± 7.38
MPT via BIAS (multi-step, proposed)	18.96 ± 5.05	25.32 ± 4.32	33.75 ± 2.77
Number of joystick inputs received to complete the trial			
MP (single-step)	192 ± 27	147 ± 45	421 ± 67
MPT via BIAS (multi-step, proposed)	44 ± 12	38 ± 9	58 ± 10

5.5 Discussion

This chapter began the first introduction of trajectory-based teleoperation with active collision avoidance by generating smooth trajectories that optimize directional intent objectives. Instead of using optimization based trajectory generation methods, we present a novel trajectory generation for teleoperation by iteratively constructing a tree of sequential actions using BIAS. BIAS generates sequential motion primitives following an intent objective, which balances the directional objective of the operator and behavioral descriptors that customize the shape of the trajectories. The method is evaluated on navigation tasks in sparse, medium and dense random forest environments. We show a significant reduction in required operator effort to complete the task using the proposed method as compared to one-step reactive teleoperation.

While the local trajectories removes the need for the operator to specify exact paths around obstacles, it does not consider the operator's higher level intention in terms of longer term paths. In the next chapter, we introduce a framework that allows the explicit incorporation of the operator's intended path.

Chapter

6

Hierarchical HITL Planning

In long duration navigation, the operator's intention is to locally avoid obstacles while planning long-horizon paths in order to complete the navigation task. This chapter extends beyond the work in the last chapter and presents a hierarchical teleoperation framework that captures these characteristics of intention and generates trajectories that are locally safe and follow the operator's global plan. The hierarchical teleoperation framework consists of 1) a global path generator which encapsulates the intended direction of the operator, 2) local trajectory generation that circumvents obstacles near the vehicle's vicinity while following the global path, and 3) a safety monitoring system to avoid possible imminent collisions. By removing the operator from teleoperation using dynamic-level control input and instead having inputs *inform* trajectory generation, we are able to significantly improve the performance of task completion while reducing the operator's engagement around dense obstacles.

The hierarchical teleoperation framework is showcased in navigation tasks in a random forest environment and a high-clutter warehouse characterized by narrow gaps and dense obstacles. With our method, we maintain consistent high speed throughout the task with smooth jerk profiles, decreased time to completion, and significantly reduced operator engagement.

This chapter first appears in [22].

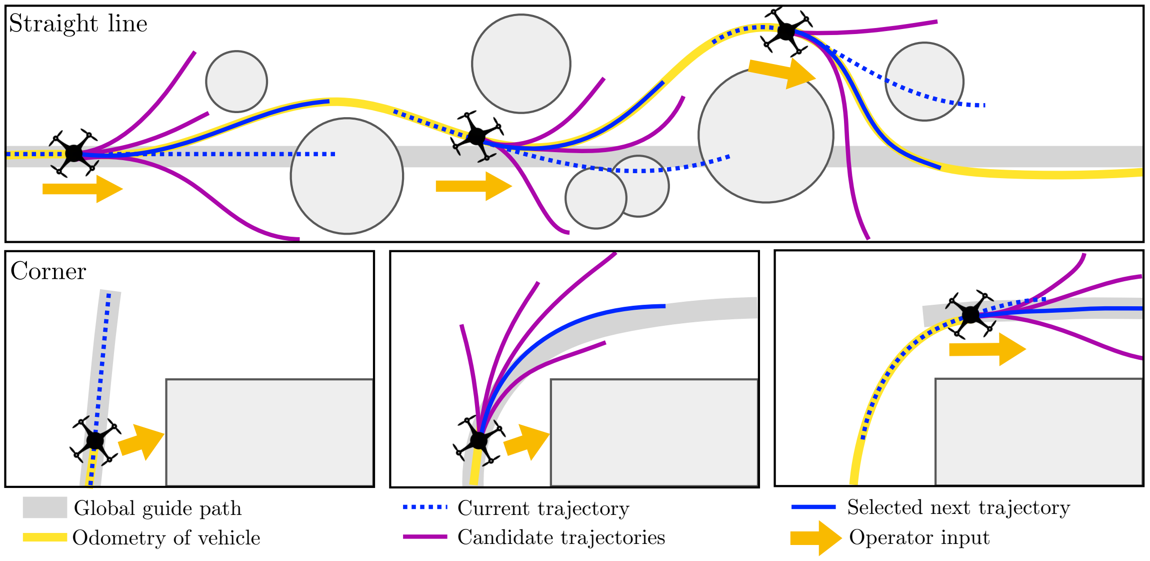


Figure 6.1: Illustrations for the proposed hierarchical teleoperation framework in two example scenarios. Operator's intention, reflected in the global path, guides local trajectory generation. Top: the vehicle avoids obstacles following a linear motion. Bottom: The intention to round a corner is reflected in the global path, which triggers a local trajectory regeneration.

6.1 Introduction

In human-robot collaboration, *intention* can be represented in various ways. Notably they can be represented either as a goal [7–9] in the state space (e.g., a door, an address, a specific object), or they can be represented as a path [10–12] (e.g., a sequence of streets to take to reach an address). However, the operator's intention can also be represented at a lower level, either as a specific motion [13, 14] (e.g., a left turn, a right turn or perform an in-place yaw), or as a system set-point [15, 16] (e.g., drive a vehicle at a fixed velocity). These various levels of intention are dependent on the context and scenario.

Human-in-the-loop control in unstructured environments for tasks such as navigation or exploration requires operators to 1) maneuver the vehicle with safety and dynamic feasibility to avoid collisions and 2) plan global paths that achieve the objective. In critical situations where the vehicle is traveling fast through dense obstacles, fast reactivity and high-frequency engagement is required to mitigate collisions. The operator must balance generating reactive motions that evade obstacles, and long-term path planning in order for task completion. The operator's intention is characterized according to the above requirements: to locally avoid obstacles while generating dynamically safe motions, and planning long-horizon paths for task completion. Typically, a global path refers to a path from the starting location to a final goal location. However, it is not often that a global goal can be clearly defined during such navigation tasks. Therefore, we represent and interpret the operator's intention as the *global path*, and the *local trajectory* can be interpreted as *a set of sequential actions to*

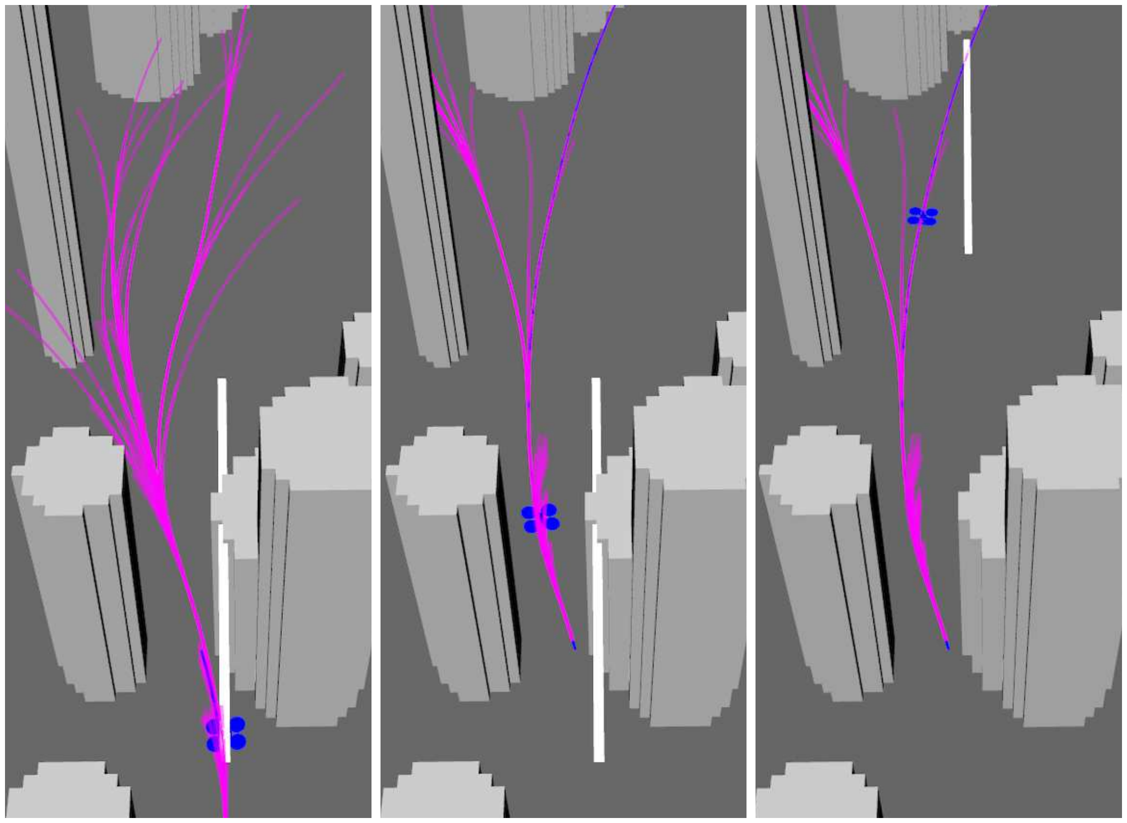


Figure 6.2: A sequence of the hierarchical teleoperation framework in action in the random forest environment. In this snapshot, the global trajectory (in grey) represents a forward linear motion. The local trajectory generator generates candidate trajectories (in magenta). The vehicle (in blue) follows a chosen trajectory and successfully passes through a narrow gap and returns to follow the global path.

satisfy the operator's intention. Consequently, the global path is therefore longer in horizon as compared to the the local trajectory. An illustration of this is shown in Fig. 6.4.

This chapter presents a hierarchical human-in-the-loop control framework that captures the instrinsic nature of human intenion for navigation in unstructured environments, and generates safe trajectories accordingly, removing the operator from the responsibilities of safety, reactivity and dynamic feasibility. The proposed framework generates predictive trajectories that utilize the operator's input to infer a global navigational direction, and local trajectories that are both locally safe and and follow the intended direction, as shown in Fig. 6.1. Our proposed method allows the operator to remain in control of the vehicle while significantly reducing operator engagement and maintaining consistent high speeds, especially during critical navigation scenarios, as shown in Fig. 6.2 and Fig. 6.3.

The hierarchical framework consists of three components: 1) global path generation using a directional intention model informed by the operator's inputs; 2) a local trajectory generator that evaluates the current joystick input and generates snap-continuous trajectories

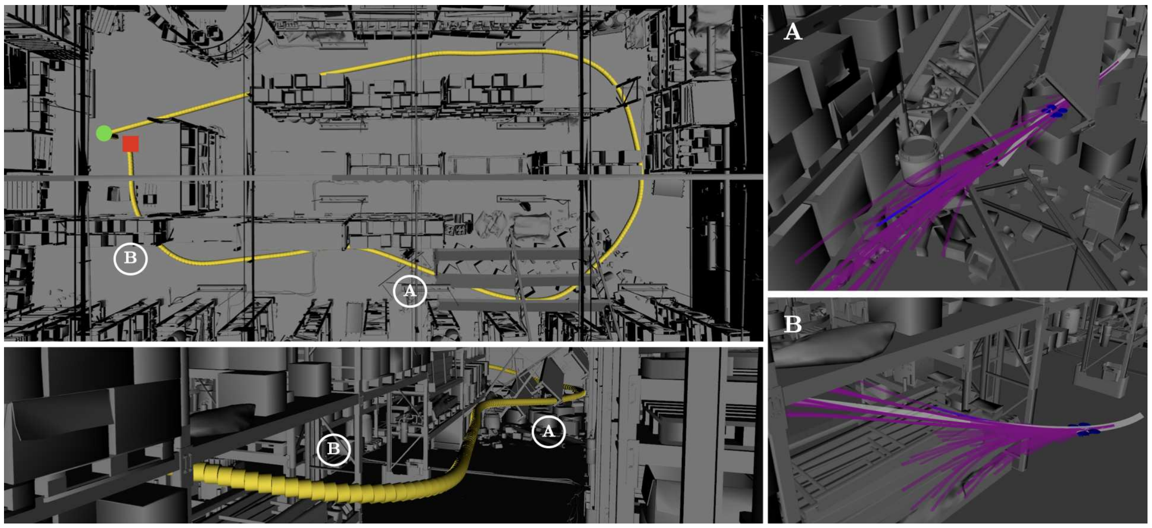
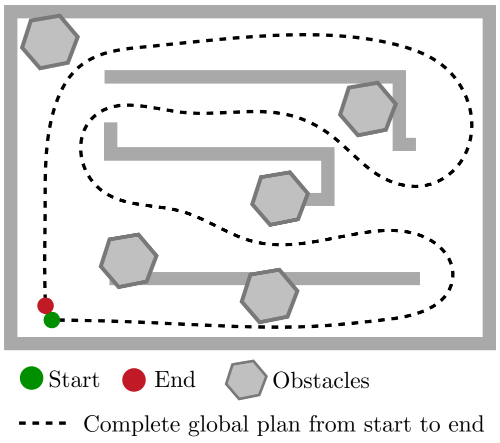


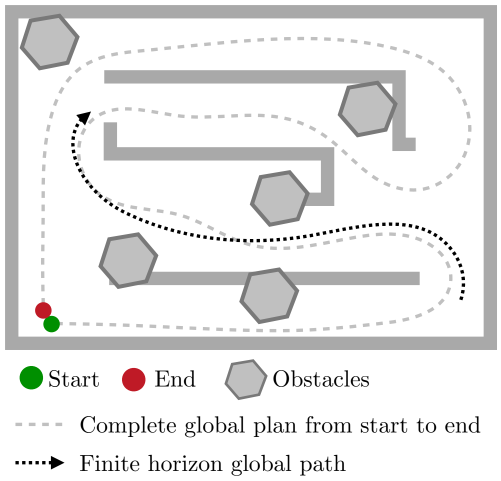
Figure 6.3: Snapshot of the hierarchical teleoperation framework in action in the warehouse environment. Top left: An overhead view of the resulting odometry (in yellow) starting near the entryway of the warehouse and exploring clockwise. Bottom left: Highlighting a section of the trial where the vehicle encounters two difficult scenarios: (A) through a partially collapsed shelf and (B) return to the origin by going through two shelves with low clearance. In those two scenarios, the global guiding path (in grey) allows generation of candidate local trajectories (in magenta). The selected trajectory (in blue) successfully leads the vehicle through narrow gaps in the highlighted scenarios.

that circumvent obstacles while following the global path; and lastly, 3) a safety monitoring system that continuously monitors collision safety of the vehicle. The inputs are processed such that linear motions such as yaw and stop are directly executed while other inputs (representing navigation) *inform* the trajectory generation process. This allows the operator to retain natural control of vehicle with imperceptible vehicle performance while reducing the number of inputs required of the operator in order to achieve task. *The application of the proposed method to human-in-the-loop control allows teleoperation to move from an operator-controlled vehicle to an operator-informed, predictive trajectory generation model, such that the trajectories naturally complete the operator's intended motion.* The hierarchical design of the proposed framework naturally corresponds to the spectrum of intentions that would otherwise require high task-switching costs on the operator, prioritizing between safety and task completion.

We test the proposed teleoperation frameworks in two types of tasks: Navigating a quadrotor in a random forest environment where the operator is asked to follow a straight line, and navigating in a cluttered warehouse characterized by narrow gaps and irregular free space formed by collapsed shelves. We show that with our method, we are able to maintain consistent high speed throughout the task with smoother jerk profiles, with a significant engagement reduction as compared to single-step trajectory-based teleoperation.



(a) Complete global plan from a start configuration to an end configuration


Start
End
Obstacles
Complete global plan from start to end
Finite horizon global path

(b) HITL representation with unknown goal: Finite horizon global path

Figure 6.4: Illustration of a global path for HITL: A global path typically represents a complete plan from a start configuration to an end goal configuration. For HITL with an unknown goal specification, a finite horizon path can be used instead.

6.2 Approach

We now introduce the hierarchical teleoperation architecture, as shown in Fig. 6.5. The proposed teleoperation system is composed of a global planning component and and a local trajectory generator, which is supplemented by a collision safety monitoring system. The operator's inputs are assumed to come from a continuous stream of values via a joystick or gamepad.

The current architecture assumes the existence of a local map representation for the purposes of collision checking. For this paper, we utilize a KD-Tree based local map representation [19]. The existence of various map representations can be readily incorporated into this framework.

6.2.1 Preliminaries

We build on definitions first introduced in Chapter 3 and Chapter 5.

Recall, a trajectory is a time-parameterized function $\xi(t)$, defined over a time interval $t \in [0, T]$ that maps a given time t to a state $\mathbf{x}_t$. Operator's inputs are given in the form of $\mathbf{a}=[v_x, \omega, v_z]^\top$ via a joystick, where $v_x \in \mathcal{V}_x$ is the linear velocity, $\omega \in \Omega$ is the angular velocity, and $v_z \in \mathcal{V}_z$ is the z-velocity. A motion primitive $\gamma(t)$ is a parameterized trajectory function which generates a unique sequence of states given an initial state $\mathbf{x}_0 \in \mathcal{X}$ and an input

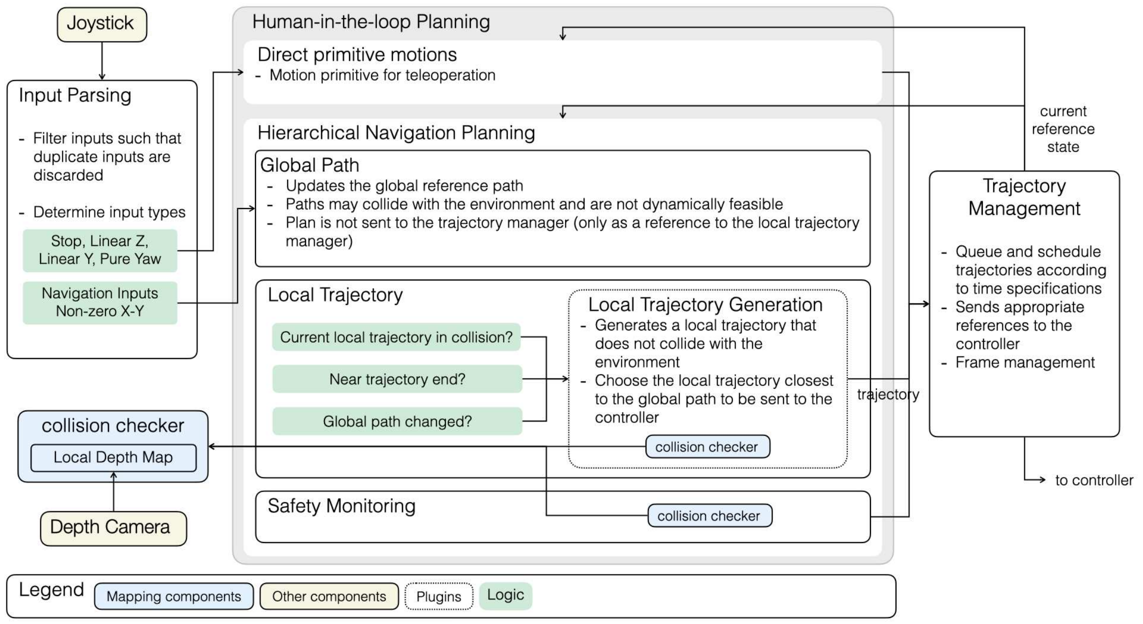

Joystick
Input Parsing
- Filter inputs such that duplicate inputs are discarded
- Determine input types
Stop, Linear Z, Linear Y, Pure Yaw
Navigation Inputs Non-zero X-Y
collision checker
Local Depth Map
Depth Camera
Human-in-the-loop Planning
Direct primitive motions
- Motion primitive for teleoperation
Hierarchical Navigation Planning
Global Path
- Updates the global reference path
- Paths may collide with the environment and are not dynamically feasible
- Plan is not sent to the trajectory manager (only as a reference to the local trajectory manager)
Local Trajectory
Current local trajectory in collision?
Near trajectory end?
Global path changed?
Local Trajectory Generation
- Generates a local trajectory that does not collide with the environment
- Choose the local trajectory closest to the global path to be sent to the controller
collision checker
Safety Monitoring
collision checker
trajectory
current reference state
Trajectory Management
- Queue and schedule trajectories according to time specifications
- Sends appropriate references to the controller
- Frame management
to controller
Legend
Mapping components
Other components
Plugins
Logic

Figure 6.5: System diagram of the proposed framework. The teleoperation framework takes in a continuous stream of inputs from an user-operated joystick, and generates a trajectory to be sent to the controller. Inputs are processed and sent to the planning pipeline. The safety monitoring system intervenes when the vehicle is about to face an imminent crash.

$\mathbf{a} \in \mathcal{A}$ according to specific dynamics. A sequence of N motion primitives is given by:

$$\xi = (\gamma_1, \ldots, \gamma_N) = (\gamma_i)_{i=1}^{N} \tag{6.1}$$

The total duration of the motion primitive sequence is given by $\mathbf{T} = \sum_{i=1}^{N} T_i$, where T_i is the duration of the ith primitive. The trajectory function for a sequence of motion primitives is defined as:

$$\xi(t) = \gamma_i(t - \tau_{i-1}) \qquad t \in [\tau_{i-1}, \tau_i) \tag{6.2}$$

where τ_i is the cumulative duration up to primitive i. Note that, given the formulation of each of the motion primitives, the endpoint continuity is guaranteed up to snap (see Eq. 3.6). Therefore, the trajectory formed by sequential motion primitives $\xi(t)$ will be continuous up to snap.

6.2.2 User Input Processing

Inputs from the operator are received as a continuous stream of joystick values around 200Hz. A *novel input* is defined as an input that leads a zero-order hold over a horizon of at least 100ms. Only novel inputs are retained and the rest discarded; hence the term

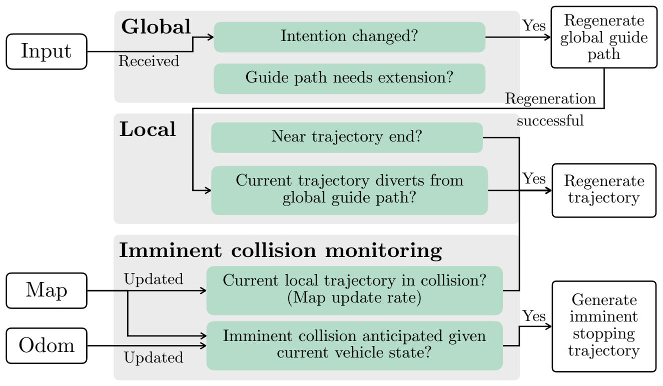

Input
Received
Global
Intention changed?
Guide path needs extension?
Yes
Regenerate global guide path
Regeneration successful
Local
Near trajectory end?
Current trajectory diverts from global guide path?
Yes
Regenerate trajectory
Imminent collision monitoring
Map
Updated
Current local trajectory in collision? (Map update rate)
Odom
Updated
Imminent collision anticipated given current vehicle state?
Yes
Generate imminent stopping trajectory

Figure 6.6: Logic flow diagram for the proposed hierarchical teleoperation framework during navigation-type tasks.

"novel inputs" is used interchangeably with "operator inputs" or "inputs". In order to allow operators to remain in control, inputs are categorized as pure yaw, zero input, or navigation. For pure yaw and zero inputs, a direct linear trajectory is sent to the controller bypassing the rest of the system, as they do not correspond to the task of navigation. The navigation inputs are passed onto the hierarchical planning framework.

6.2.3 Global Planning

The global layer generates a path that reflects a long horizon motion that the operator intends to achieve in the absence of obstacles. We utilize a simplified intention model based on previous navigational inputs as follows: First, the navigational inputs are filtered to produce a likely global input $\mathbf{a}_G$ at time t given current input $\mathbf{a}_t$:

$$\mathbf{a}_G = \lambda \mathbf{a}_G + (1-\lambda)\mathbf{a}_t \qquad 0 < \lambda < 1 \tag{6.3}$$

Then, a global trajectory ξ_G is generated according to Eq. (3.3) with a duration of $\mathbf{T}$. As the global path is only used for guidance, any higher order dynamics of the trajectory can be safely ignored. For our experiments, we choose a horizon of $\mathbf{T}$=10s and λ=0.8.

6.2.4 Local Trajectory Generation

We generate local trajectories that follow the underlying global trajectory while avoiding obstacles. This process uses the operator's input and the global path to generate trajectories that are most closely bound to the operator's intention, while maximizing operator control. The logic flow for this process is summarized in Fig. 6.6.

6.2.4.1 Motion Primitive Parameterization

The operator's input is first parameterized as a single-step snap-continuous motion primitive following Eq. 3.6.

The motion primitive is checked against collisions with respect to the map representation. If the motion primitive is in collision, then we utilize a longer-horizon trajectory generation process as described in the next section.

6.2.4.2 Multi-Step Trajectory Generation

In the case that a single-step motion primitive, which is an exact parameterization of the operator's action in the state space, leads to a collision, it is necessary to generate a trajectory that avoids the obstacle in its immediate environment. To do so, we generate a candidate set of trajectories by constructing a motion primitive tree using Biased Incremental Action Sampling (BIAS), introduced in Chapter 5, in given a local map representation.

Recall, BIAS iteratively builds a tree of sequential motion primitives that minimizes an objective. The objective is a weighted combination of the local intended direction and behavior heuristic cost functions. The local intended direction cost function is as follows: The cost function is given as:

$$C_{\text{input}_{\mathbf{a}}}(\xi) = \|1 - \mathbf{p} \cdot \mathbf{p}^*\| \tag{6.4}$$

$$\mathbf{p} = \frac{\xi(\mathbf{T}) - \xi(0)}{|\xi(\mathbf{T}) - \xi(0)|} \qquad \mathbf{p}^* = \frac{\gamma_{\mathbf{a}}(\mathbf{T}) - \gamma_{\mathbf{a}}(0)}{|\gamma_{\mathbf{a}}(\mathbf{T}) - \gamma_{\mathbf{a}}(0)|} \tag{6.5}$$

where $\xi(\tau)$ is the multi-step motion primitive trajectory evaluated at time τ, and $\gamma_{\mathbf{a}}(\tau)$ is the single-step motion primitive parameterized by the operator's given input $\mathbf{a}$ evaluated at time τ. Therefore, γ is exactly the motion primitive in collision as generated in the previous section, so as to maximize adherance to the operator's intended input.

6.2.4.3 Trajectory Selection

Given a set of candidate trajectories, we select the trajectory to be executed by the vehicle. We aim to first maximize smoothness in transition from the current trajectory, as well as minimize its closeness to the global guiding trajectory. To do so, we introduce a selection cost function that evaluates each trajectory by its closeness to both the current local trajectory and the global path by evaluating the *discrete Fréchet distance* δ_{dF} [111, 112]. We provide a brief definition below.

Consider a discrete sampling of two continous functions f and g that forms two polygonal curves $\mathcal{P} = \{f_1, f_2, ..., f_n\}$ and $\mathcal{Q} = \{g_1, g_2, ..., g_m\}$ which are sequences of n and m discrete points, respectively. An *order-preserving, complete correspondence* between $\mathcal{P}$ and $\mathcal{Q}$ is a

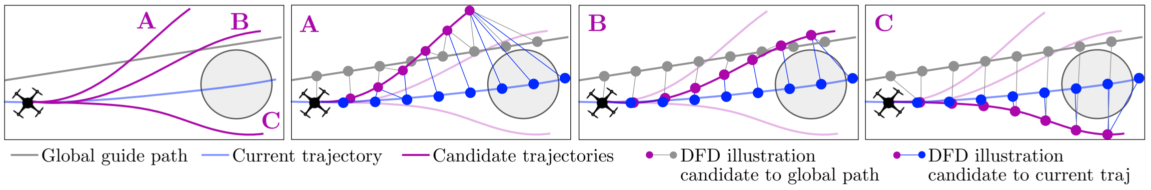

A
B
C
Global guide path
Current trajectory
Candidate trajectories
DFD illustration candidate to global path
DFD illustration candidate to current traj

Figure 6.7: Illustration of new trajectory selection based on weighted discrete Fréchet distance (DFD). Each candidate trajectory is sampled and DFD is computed between the candidate and global path and local trajectory. In this example, candidate B scores the lowest DFD and is sent to the controller.

pair (α, β) of *discrete monotone reparameterizations* [1] of α from $\{1, ..., k\}$ to $\{1, ..., n\}$ and of β from $\{1, ..., k\}$ to $\{1, ..., m\}$. The discrete Fréchet distance of $\mathcal{P}$ and $\mathcal{Q}$ is given by:

$$\delta_{dF}(f, g) := \min_{(\alpha,\beta)} \max_{i\in[1,k]} d(f_{\alpha(i)}, g_{\beta(i)}) \tag{6.6}$$

where (α, β) ranges over all order-preserving complete correspondences between $\mathcal{P}$ and $\mathcal{Q}$. Therefore, the Fréchet distance for a pair of time parameterized trajectories $\xi, \varPhi$ is given by

$$\delta_{dF}(\xi, \varPhi) := \min_{(\alpha,\beta)} \max_{i\in[1,k]} d(\xi_{\alpha(i)}, \varPhi_{\beta(i)}) \tag{6.7}$$

where $\xi_i = \xi(i \cdot \varDelta t)$ for fixed $\varDelta t$ sampling of trajectory ξ.

The trajectory selection is then as follows: Given the current local trajectory ξ_L and the guiding global trajectory ξ_G, and a candidate set of trajectories $\{\xi\}$,

$$\xi^* = \min_{\xi\in\{\xi\}} w_L \delta_{dF}(\xi, \xi_L) + w_G \delta_{dF}(\xi, \xi_G) \tag{6.8}$$

An illustration of the trajectory selection process is shown in Fig. 6.7.

6.2.5 Safety Monitoring

6.2.5.1 Trajectory Safety

The size of the local map is dependent on the accuracy range of the sensors in exploring unknown environments. As such, this range is sometimes fairly limited. Therefore, we treat the unknown space as free space during trajectory generation with the trajectories possibly extending beyond the size of the local map. As the vehicle moves, the local map is being updated on a rolling basis with incoming new sensor scans. Therefore, the current trajectory is continually checked against the updated map with a new trajectory generated in the event of a possible collision.

[1]A *discrete monotone reparameterization* α from $\{1, ..., k\}$ to $\{1, ..., l\}$ is defined as a non-decreasing function $\alpha : \{1, ..., k\} \to \{1, ..., l\}$ for integers $k \geq l \geq 1$, with $\alpha(1) = 1$, $\alpha(k) = l$, and $\alpha(i+1) \leq \alpha(i)+1 \; \forall \; i = 1, ..., k-1$.

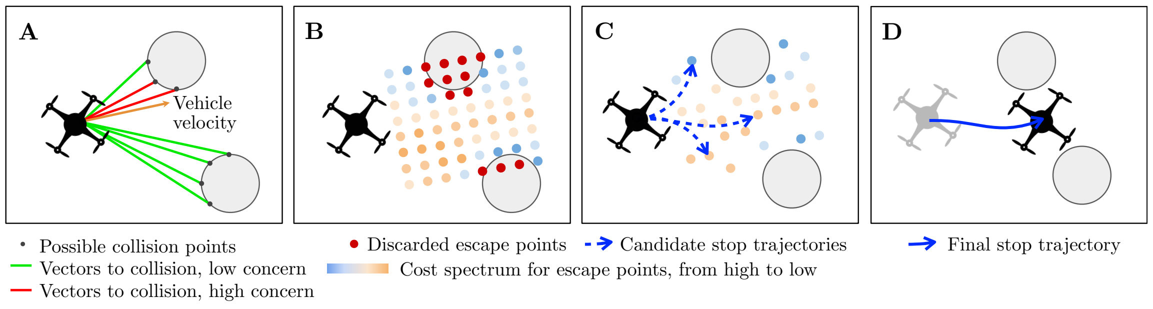

A
Vehicle velocity
B
C
D
• Possible collision points
Vectors to collision, low concern
Vectors to collision, high concern
● Discarded escape points
Cost spectrum for escape points, from high to low
Candidate stop trajectories
Final stop trajectory

Figure 6.8: Illustration of imminent collision monitoring and safe stop. (A) Initial assessment: possible collision points are sampled from nearby obstacles and vectors to collision are assessed. If high concern vectors exist, stopping trajectory generation begins. (B) Initial grid of escape points with costs computed and colliding points discarded (C) A stratified sampling method is used to downsample the possible escape points, with candidate trajectories generated. Dynamically infeasible candidate trajectories are discarded. (D) Lowest cost, dynamically feasible trajectory is selected and sent to the controller.

6.2.5.2 Imminent Collision Monitoring

To mitigate situations where the operator leads the vehicle into state of collision or the vehicle reaches a point where no trajectories are feasible, a collision checking algorithm is implemented alongside the planning pipeline. The imminent collision checking system continuously monitors vehicle safety, and plans a safe, dynamically feasible trajectory that leads the vehicle to a stop within the observed map.

To determine whether an imminent collision will occur given the immediate vehicle surroundings, we first determine the set of possible collision points by evaluating the normalized vector projection between each of the closest obstacle locations and the vehicle velocity:

$$\text{proj}(\mathbf{x},\mathbf{p}) = \left\langle \frac{\dot{\mathbf{x}}}{\|\dot{\mathbf{x}}\|}, \frac{\mathbf{p}-\mathbf{x}}{\|\mathbf{p}-\mathbf{x}\|} \right\rangle = \left\langle \frac{\dot{\mathbf{x}}}{\|\dot{\mathbf{x}}\|}, \frac{\mathbf{r}}{\|\mathbf{r}\|} \right\rangle$$

Where $\mathbf{x}$, $\dot{\mathbf{x}}$ are the vehicle position and velocity respectively, and the vector to an obstacle point $\mathbf{p}$ is $\mathbf{r} = \mathbf{p} - \mathbf{x}$. The points away from the direction of motion $\text{proj}(\mathbf{x},\mathbf{p}) < 0$ can be discarded immediately. Then, we compute a combined stop criterion on the velocity magnitude, distance of the obstacle to the vehicle and the angle offset of the obstacle on a set of closest obstacle points $\{\mathbf{p}\}$:

$$\begin{aligned} C_{\text{stop}}(\mathbf{x},\mathbf{p}) = w_1|\dot{\mathbf{x}}| + w_2|\mathbf{r}| + w_3 \arccos(\text{proj}(\mathbf{x},\mathbf{p})) \\ \text{if } \text{proj}(\mathbf{x},\mathbf{p}) >= 0 \end{aligned}$$

An imminent stop trajectory is issued if for any obstacle point $\mathbf{p}$, $C_{\text{stop}}(\mathbf{x},\mathbf{p}) < \beta$. In our experiments, β, w_1, w_2 and w_3 are experimentally determined to be 0.9, 0.2, 0.8, 1.2

respectively. To generate an imminent stop trajectory that is guaranteed safe, an initial group of escape points, $\{\mathbf{e}\}$, is sampled along the direction of motion using stratified sampling over a uniform grid optimizing for points near the vehicle's original heading via the following cost:

$$C_{\text{escape}}(\mathbf{x}, \mathbf{e}) = w_1 q + w_2 d$$

Where $q = |(\mathbf{x} - \mathbf{e} - ((\mathbf{x} - \mathbf{e}) \cdot \hat{\dot{\mathbf{x}}})\hat{\dot{\mathbf{x}}}|$ is the shortest distance from the escape point $\mathbf{e}$ to the line $l(s) = \hat{\dot{\mathbf{x}}}s + \mathbf{x}$ in the direction of the velocity vector passing through the vehicle position, and d is the distance from the the escape point to its nearest obstacle. The stratified method guarantees a set of points with both high and low costs such that in the event low cost trajectories are not dynamically feasible, we are able to sacrifice cost for dynamic feasibility. A set of stopping trajectories are then generated to the resulting sampled points and checked for safety and dynamic feasibility via an acceleration bound along the trajectory s.t. $\ddot{\mathbf{x}} < \bar{\alpha}$. For our experiments, we select $\bar{\alpha} = 10\text{m/s}^2$. The resulting set of safe, dynamically feasible trajectories with the lowest C_{escape} value is chosen. An illustration of the imminent collision checking system is shown in Fig. 6.8.

6.3 Experiments and Results

We evaluate our method in a simulated random forest environment and in a dense warehouse environment, where the operator is asked to perform a navigation task by following a global path described to the operator. The operator is given a third-person follower view of the vehicle, which may introduce occlusions depending on the complexity of the environment and any overhanging objects. Each task is repeated with five trials.

The simulated vehicle is an exact model of an in-house developed quadrotor. The vehicle weighs 1.14kg, and has an effective diameter of 30.6cm. The simulated experiments are performed with a CPU (Intel Core 2.20GHz i7-8750H CPU), with 16GB of RAM.

The proposed method is compared to direct motion primitive teleoperation (MP) [19], which is a one-step parameterization of the operator's action input, as well as multi-step trajectory generation via motion primitive trees (MPT) [113]. All three methods are equipped with the imminent collision monitoring system in order to avoid possible crashes. An example trial of the collision monitoring and prevention in the warehouse scenario is shown in Fig. 6.9. The stopping trajectory quicky stops vehicle before collision and allows the vehicle to escape.

6.3.1 Metrics

We evaluate our experiments with the following criteria: 1) time to completion, 2) smoothness by evaluation of the jerk integral, and 3) operator engagement by evaluation of the number of novel inputs over the fixed task. Further, for the warehouse scenario, we additionally evaluate 4) the average velocity.

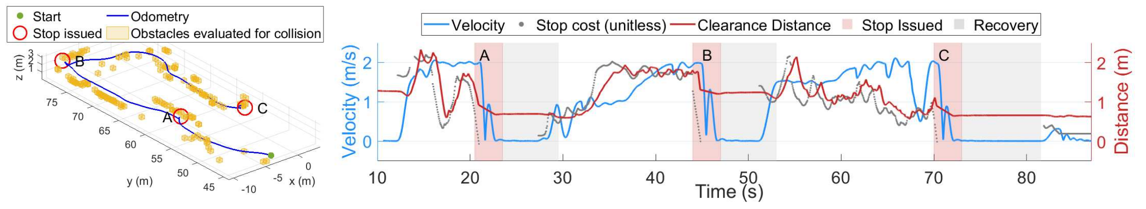


Figure 6.9: Trial run in the warehouse environment with imminent collision checking enabled. Left: Vehicle odometry with queried obstacles. Locations A, B, C indicate where stopping trajectories were issued corresponding with the left plot. Right: Stopping trajectories are issued when stop cost drops below a threshold, ensuring that the vehicle is safe. The operator recovers the vehicle with an in-place yaw before normal flight is resumed.

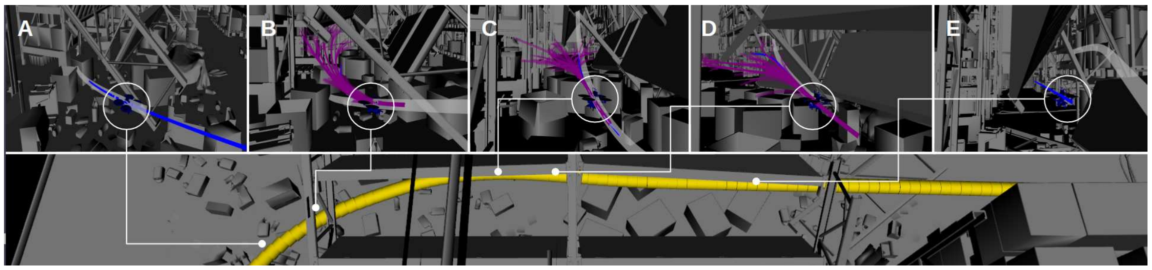


Figure 6.10: Sequence of the passage through the collapsed shelf, from left to right. Trajectory currently following (in blue) leads to a collision, and trajectory generation is triggered resulting in candidate trajectories (in magenta) being generated. Light grey trajectory highlights the global path that guides the trajectory generation process, which adapts over time as the vehicle moves.

We expect the proposed method allows smooth navigation without the need to frequently stop due to possible collisions. Lastly, we focus on the key metric of operator engagement by evaluation of the number of novel inputs to complete a task. If the number of new inputs is high, this indicates that operators feel the need to control the vehicle in order to correct its course. Alternatively, if the number of new inputs is low, it implies that the vehicle is following course on its intended trajectory and does not require correction to its motion.

6.3.2 Random Forest

The random forest environment contains 120 various-height pillars in a 60m×30m×10m volume. The operator is asked to navigate through the random forest environment following a straight line path that passes through many pillars. The vehicle begins on one side of the random forest, and the task is marked complete as soon as the vehicle reaches the other side of the random forest.

The results are tabulated in Table 6.1. Overall, the tasks are complete within reasonable time. We observe that the proposed method maintains a jerk integral of $25\text{m}^2/\text{s}^3$ for all three density environments, whereas previous methods show an increase in jerkiness depending on the density. We also observe a significant decrease in the number of joystick inputs to

Table 6.1: Results for three different density random forest environments

Approach	**Sparse**	**Medium**	**Dense**
	Time to completion (s)		
MP (single-step)	38.84 ± 1.08	37.06 ± 0.50	37.95 ± 1.91
MPT (multi-step)	39.45 ± 1.43	36.40 ± 1.99	37.55 ± 2.34
Hierarchical (proposed)	$\mathbf{38.10 \pm 3.96}$	$\mathbf{33.4 \pm 1.44}$	$\mathbf{34.73 \pm 1.35}$
	Jerk Integral (m^2/s^3)		
MP (single-step)	28.41 ± 2.88	26.72 ± 4.30	50.14 ± 7.38
MPT (multi-step)	18.96 ± 5.05	25.32 ± 4.32	33.75 ± 2.77
Hierarchical (proposed)	$\mathbf{25.10 \pm 18.10}$	$\mathbf{23.12 \pm 10.42}$	$\mathbf{25.94 \pm 7.96}$
	Number of operator inputs to complete the trial		
MP (single-step)	192 ± 27	147 ± 45	421 ± 67
MPT (multi-step)	44 ± 12	38 ± 9	58 ± 10
Hierarchical (proposed)	$\mathbf{21 \pm 12}$	$\mathbf{27 \pm 12}$	$\mathbf{35 \pm 10}$

complete the trial: the proposed method shows a 89%, 82%, 92% reduction from the baseline motion primitive method for the sparse, medium and dense environments respectively. We also observe that the number of inputs required to complete the random forest environment remains constant, regardless of the density of the environment as shown in Fig. 6.11. For the single-step method, the number of inputs increases significantly in the dense environment. However, for trajectory-based methods, this is reduced as the trajectories allows the vehicle to maneuver through high clutter areas without frequent operator engagement.

Table 6.2: Results for the warehouse environment

Approach	**Distance (m)**	**Duration (s)**	**Avg. Vel. (m/s)**
MP (single-step)	82.126 ± 0.92	47.778 ± 0.503	1.772 ± 0.041
MPT (multi-step)	85.07 ± 3.95	59.73 ± 3.23	1.48 ± 0.08
Hierarchical (proposed)	$\mathbf{80.67 \pm 4.63}$	$\mathbf{44.85 \pm 0.93}$	$\mathbf{1.85 \pm 0.04}$

Approach	**Jerk integral (m^2/s^3)**	**Num. Inputs**
MP (single-step)	47.475 ± 21.328	175 ± 30
MPT (multi-step)	78.21 ± 11.72	77 ± 19
Hierarchical (proposed)	$\mathbf{31.19 \pm 6.81}$	$\mathbf{64 \pm 6}$

6.3.3 Warehouse

The warehouse environment is a large room with dimensions of 44.81m×22.17m×11m. The room contains three rows of industrial shelves, with a large shelf partially collapsed over and scattered objects surrounding the collapsed shelf, as highlighted in Fig. 6.3. The operator is asked to navigate in the warehouse following a rectangular path and through the collapsed shelf, which has a maximum clearance of 0.6m, and return to approximately close

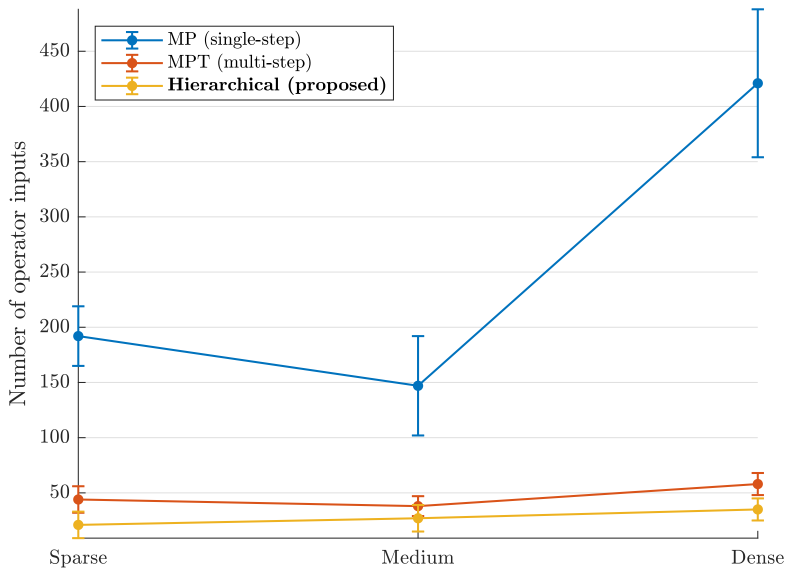


Figure 6.11: Number of operator inputs as a function of the environment density for the random forest scenario, visualized. The number of inputs required to complete the task remains constant for the proposed method regardless of the density of the environment.

to the origin, which would require a narrow pass through between two horizontal shelves. The task is marked complete as soon as the vehicle completes the desired pathway and reaches within 1m of the origin. The difficulty of this experiments is highlighted in the high density clutter near the shelves and changes in direction, as shown in Fig. 6.10.

The results are tabulated in Table 6.2. We highlight that while all three method allows the operator to safely complete the task, the proposed hierarchical teleoperation method is able to do so with 63% and 17% less number of inputs, which is a significant reduction of operator engagement. The single-step and multi-step requires on avg. 175 and 77 inputs respectively, whereas the proposed method requires only 64.

While the multi-step method is comparable, the multi-step method takes 15 seconds longer to complete the task than the proposed method for the approximately same distance travelled with a reduced average velocity of 1.48m/s, compared to the proposed method's 1.85m/s. This is due to the fact that the multi-step optimizes for finite-horizon navigation; a rapid direction change would require the vehicle to slow down significantly to turn the corner, and then speed up.

The single-step method performs comparatively to the proposed method in terms of task completion duration and average velocity. However, we are able to achieve the same level of performance with a 63% reduction in the number of inputs required.

6.4 Discussion

This chapter presents a hierarchical teleoperation framework for mobile robot navigation in unstructured environments, informed by the operator's intention. The proposed method is demonstrated in teleoperation tasks navigating in densely cluttered random forest and warehouse environments characterized by narrow gaps and unstructured free space. The proposed method completed the navigation task with consistently high average speeds while requiring the least operator engagement.

The hierarchical teleoperation framework can be readily extended in a few ways. The hierarchical formulation of the global and local planning allows a natural extension to interface with global and local map representations, such that the global plan incorporates environment features. Second, a more sophisticated operator intention model can be used to inform the trajectory generation process. In the next chapter, we discuss incorporating navigation graphs as global plans and utilizing them to guide local trajectory generation.

Chapter

7

Continuous Dynamic Autonomy via Path Prediction on Semantic Topological Maps

Dynamic autonomy has been introduced to allocate control between human control and robot autonomy. However, dynamic autonomy has primarily been employed in a discrete way, where hand-off between the human and fully autonomous operation is clearly marked by a distinct event. Discrete hand-offs in continuous settings such as driving has shown to be destabilizing to the human-robot system.

We propose a *continuous dynamic autonomy* framework for navigation, such that the transition between human control and robot is imperceptible. Under continuous dynamic autonomy, control is fluidly allocated between the human and robot during operation.

The last chapter utilized a simple projection model akin to Constant Velocity Model (CVM) or Average Velocity Model (AVM) to model the global path. These models has been shown to outperform more complex, state-of-the-art models in single-agent motion prediction [114]. However, to be able to perform complex predictions about motions in constrained environments, these models fail to consider the obstacles and the constraints on traversability they impose. This chapter discusses incorporating the environment context via a semantically topological navigation graph to achieve continuous dynamic autonomy.

7.1 Introduction

Dynamic autonomy has been introduced to allocate control between the human vs. robot autonomy. In continuous settings such as navigation, the most efficient framework of dynamic autonomy should *maximize the strengths of the human and robot wherever possible*, such that the control of the robot fluidly switch between the human and robot autonomy depending on the intention given the environment context. However, dynamic autonomy has primarily been employed in a discrete way, where hand-off between the human and fully autonomous operation is clearly marked by a distinct event. The interaction, from the perspective of the human, is discrete, as marked hand-over from autonomy to human's input; the control reference, from the perspective of the robot, is also discrete, as the control input coming from the human is often discontinuous from the autonomy control input. We refer to this as a *discrete-interaction, discrete-control* scenario. Discrete hand-offs in continuous settings such as driving has shown to be destabilizing to the human-robot system. We propose a continuous, seamless human-in-the-loop control framework with *continuous dynamic autonomy.* Depending on the environment context and the modeling of the operator's intention, the autonomy of the robot is dynamically allocated, ranging from operator's input-driven to fully autonomous trajectory generation. Our method is driven by two key principles: 1) Incorporate the environment information as a contextual prior for the human's decision making during navigation and 2) Enable a continuous and smooth hand-over control between the human and robot.

We return to the two example scenarios first introduced in Chapter 1, Section 1.1, shown in Fig. 7.2. The first task is to perform search and rescue in a wooded area with rough terrain using an all-terrain vehicle (ATV). The second task is a search and inspect operation in a potentially hazardous warehouse using an MAV to identify potentially dangerous objects of interest. In both of these cases, some partial information about the environment is available: Satellite imagery provides an approximate map of the possible paths that the ATV could traverse. However, this information is *imperfect*; it is not an absolute and precise representation of the traversability conditions. Unpredictable obstacles at the ground level such as fallen trees lead to changes to traversability, in addition to smaller arterial pathways that may be unobservable via satellites. Similarly with the warehouse, blueprints provides detailed information of the warehouse in its original state, but do not include changes such as collapsed shelves and additional objects. Strictly using imperfect map information for navigation is infeasible for fully autonomous navigation systems. However, the partial availability of map information provides rich, contextual information about the environment that could assist humans within a human-in-the-loop context. This information is especially informative regarding the operator's intentions, especially in constrained environments where the set of paths are limited.

In constrained environments, the set of paths in the environment that the robot can undertake are enumerable and distinct. These set of paths form a set of homotopic classes that

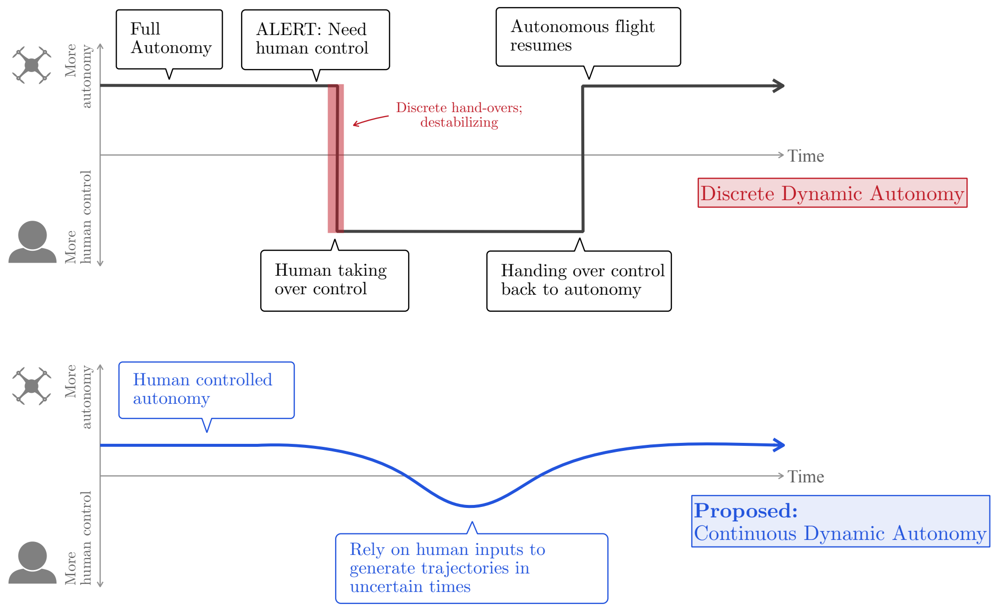

Full Autonomy
ALERT: Need human control
Autonomous flight resumes
Discrete hand-overs; destabilizing
More autonomy
More human control
Time
Discrete Dynamic Autonomy
Human taking over control
Handing over control back to autonomy
Human controlled autonomy
Rely on human inputs to generate trajectories in uncertain times
Proposed: Continuous Dynamic Autonomy

Figure 7.1: Discrete dynamic autonomy is highlighted by distinct hand-offs between human and robot. We propose the idea of continuous dynamic autonomy, where assistance is provided as needed and does not require explicit communication and hand-off between human and robot.

represent certain topologies of the environment, such that the topologies are "semantically meaningful", i.e., they represent traversable navigation routes. Thus, the traversability of the environment can be encoded with a navigation graph that represents the connectivity

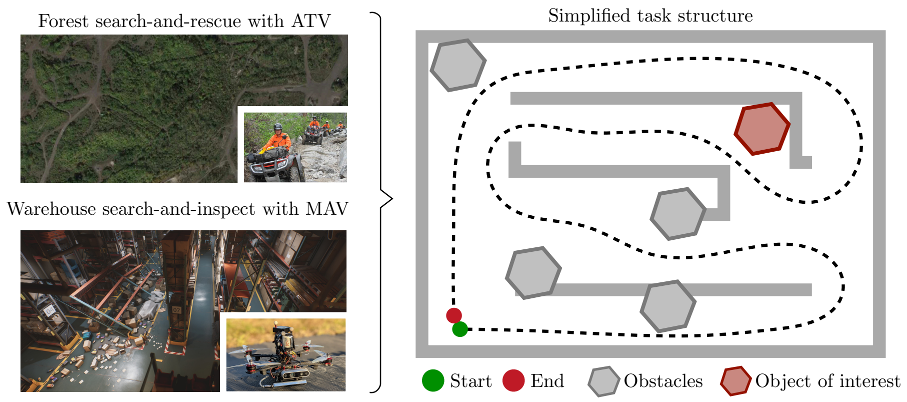

Forest search-and-rescue with ATV
Warehouse search-and-inspect with MAV
Simplified task structure
Start
End
Obstacles
Object of interest

Figure 7.2: Motivating scenarios for human-in-the-loop navigation. The example navigation tasks on the left follow a simplified task structure: navigate and identify/search/inspect along the path, and return to the start. Continuous dynamic autonomy seeks to assist teleoperation in these scenarios without well-defined goals.

of free space. Once the routes are determined as a function of the environment structure, additional rich details about the environment does not necessarily contribute to navigation decisions. This gives rise to a key insight that, *the environment information for navigation purposes can be succinctly condensed into a semantically representative topological navigation map for decision making.*

This insight leads to the following observation: the role of the human in HITL navigation in constrained environments can be reduced to selecting from a set of discrete, traversable paths. By incorporating the environment context as a semantically topological navigation graph, the robot's autonomy can be further increased by predicting the path that the human intends to follow. The intended global path prediction can be formulated by updating beliefs about the set of discrete, distinctive paths from the navigation graph as the robot moves. In certain scenarios where the human provides an indicative input that deviates from the predictions, control of trajectory generation is handed over smoothly to the human operator until autonomous trajectories following an updated intended path can be established. The outcome is that the transition between human control and robot is imperceptible.

This chapter discusses *continuous dynamic autonomy* in a hierarchical HITL framework by generating path predictions on a semantically topological navigation graph and discusses continuous hand-over at the trajectory generation level. As compared to discrete dynamic autonomy, continuous dynamic autonomy proposes a *continuous-interaction, continuous-control* paradigm. That is to say, the interaction from the perspective of the human is continuous, and the control reference provided to the robot is also continuous. We predict the operator's intended path on the navigation graph, and utilize the global path as a guide for local trajectory generation wherever the intention aligns with the operator input [115]. If the prediction becomes inaccurate, the system directly generates trajectories that follow the operator input instead, resulting in smooth vehicle motions without discrete transitions. We evaluate our method with a pilot study in a cluttered warehouse environment in a navigation task, and show that assistance via continuous dynamic autonomy increases human-robot efficiency by shifting control from the human to the robot. Further, we demonstrate promising trends that the operator's engagement is reduced during autonomous navigation. Lastly, we discover some insights in human preference and discuss their implications on future work.

7.2 Relevant Background

Navigation Graphs. Global planning focuses on choosing the optimal route given a fixed goal. Given a known environment, we seek a simplified representation of the environment that captures the free-space connectivity, forming feasible paths that are topologically distinct. Simple grid-based representations represent the environment as a dense voxel map or occupancy grid, which can generate a set of feasible global paths given a global goal by using search-based methods such as A* or Dijkstra's. Random graph representations

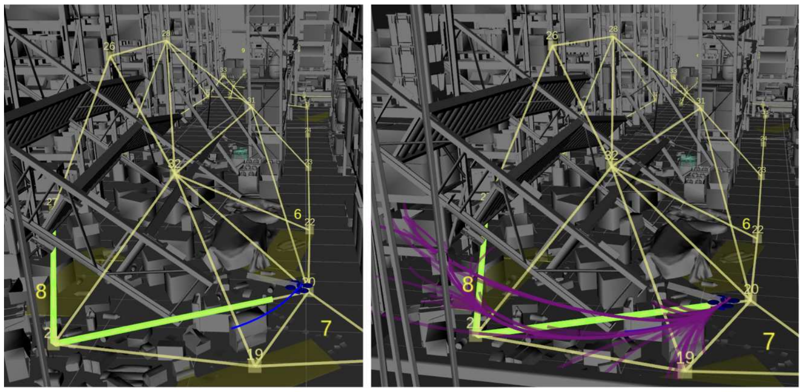


Figure 7.3: Predicted path (green) on a semantically topological graph (yellow) used to generate local trajectory candidates (magenta) during dynamic autonomy.

use sampling-based algorithms to generate feasible paths in the configuration space of the robot; for example, PRM first constructs a graph by sampling points in the configuration space and connecting neighboring feasible nodes [116]. Rapidly-exploring random trees (RRT) [117] and its many variants [118] randomly builds a space-filling tree that represents conectivity. These methods result in graphs with high density vertices and edges, resulting in redundant global paths that may not be topologically distinct. Visibility graphs [119] connect intervisible vertices of object geometries in the environment to create graphs in the traversable workspace. However, these graphs represent edge-to-edge connectivity of the obstacles in the workspace. These connected paths are usually unintuitive for navigational purposes.

Topological graphs are abstract graph-based representations that preserve the topology of the environment and maintain connectivity of the free space. The generation of topological maps broadly divides the map representation into two different categories: 1) Geometric topological maps where the representation derives from the connected geometry of the environment [120–122]; and 2) semantically connected topological maps built online during exploration-based navigation, whose connectivity is minimal but represents traversability of space [123, 124]. Topological graphs have been used in 2D navigation for ground mobile robots, derived from Voronoi partitioning of occupancy maps [120]. For 3D space, extracting the precise topology has proven to be a challenge [125]. Similar Voronoi approaches have been used to various degree of success: Generating 3D generalized voronoi diagram (GVD) has shown to be computationally difficult [126]. Sparse Topological Graph (STG) proposed by Oleynikova et al. [121] has shown success in the 3D domain. However, GVDs and consequently topological graphs derived from raw depth sensor scans will often result in noisy, repetitive edges that are impractical for the purposes of navigation. Alternatively, topological maps built during navigation do not consider the explicit detailed geometry of the depth scans, but rather containing minimal edges that are traversable conditional on the vehicle geometry and dynamics. These maps are often semantically interpretable: each path clearly represents a traversable path. However, precisely generating semantically topological maps remains an open and active area of research.

A graphical comparison of various navigation graphs is shown in Fig. 7.4.

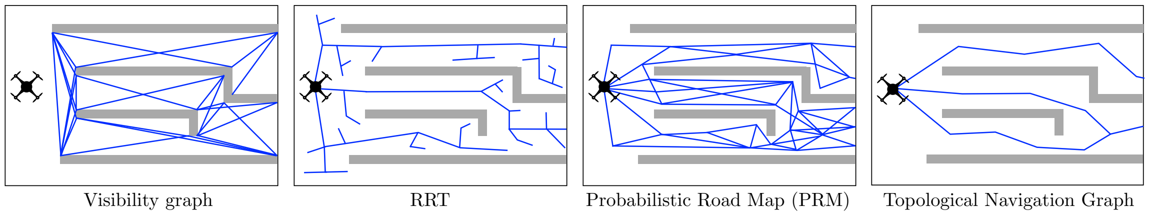


Figure 7.4: Graphical comparison of various navigation map representations.

Path prediction. Given the current state and a set of global distinctive paths accessible from the current state, the operator's intention is modeled as a choice over these paths.

The problem of choosing over these paths is formulated as a discrete selection problem. The *discrete selection* problem updates the belief over a set of hypotheses (or goals) using observations generated from sequential actions. Decision region determination (DRD) adaptively selects and execute a sequence of tests (or actions) and use the corresponding observations to reduce the set of hypotheses [127]. Maximum entropy inverse optimal control (MaxEnt IOC) [12] studies a similar path prediction problem given a navigation graph. In this case, path prediction is modeled as a deterministic Markov Decision Process (MDP) that uses the agent's noisy actions as observations. This method learns the behavior and preference of an agent via inverse reinforcement learning by matching preferences in path features and generates path predictions that optimizes the agent's preference. In contrast to our method, MaxEnt IOC uses actions as observations thus models behavior, and the proposed method uses odometry as observations, thus modeling motion.

In the absence of a discrete goal, trajectory prediction projects a predicted motion for a fixed time horizon into the future, typically in the continuous space. A simple, yet effective model is the constant velocity model (CVM)/ average velocity model (AVM), which outperforms most of the state-of-the-art complex neural network models in static environment navigations [114]. Other methods focus on plan recognition, including modeling inherent goal locations [128], modeling transition probabilities between sub-goals [129], or using reciprocal velocity obstacles [130].

Dynamic Autonomy. Dynamic autonomy has been introduced to allocate control between the human vs. robot autonomy based on environment safety and user preference. Control allocation is typically designated between several levels of discretized levels of autonomy, each with a set of pre-defined behaviors. An example set of levels of autonomy is as follows, ranging from human control to robot autonomy: 1) Teleoperation, where humans inputs are followed exactly, 2) Shared control, where human's input and robot input are combined, 3) Supervisory, where the human provides high-level goals that the robot autonomously, and interjects wherever needed, and 4) full autonomy, where the robot performs completely without a human in the loop [131].

Within a human-in-the-loop navigation setting, dynamic autonomy via discrete hand-offs between pre-defined levels of autonomy has shown to be inadequate. Timing, including reaction time and coordination has shown to be critical during the discrete transition [132] in addition to requiring a need to adapt to the system dynamics [133].

7.3 Preliminaries

We build on definitions first introduced in Chapter 3 and Chapter 5, and follow the hierarchical human-in-the-loop planning first introduced in Chapter 6.

A path ξ is a sequence of edges that connects a set of ordered waypoints. A trajectory is a time-parameterized function $\zeta(t)$, defined over a time interval $t \in [0, T]$ that maps a given time t to a state $\mathbf{x}_t$:

$$\zeta_T : [0, T] \to \mathcal{X} \qquad T \in [0, \infty),\ \zeta(0) = \mathbf{x}(0) \tag{7.1}$$

The operator's inputs are given in the form of $u_\mathrm{h} = [v_x, \omega, v_z]^\top$ via a joystick, where $v_x \in \mathcal{V}_x$ is the linear velocity, $\omega \in \mathit{\Omega}$ is the angular velocity, and $v_z \in \mathcal{V}_z$ is the z-velocity. A motion primitive $\gamma(t)$ is a parameterized trajectory function that generates a unique sequence of states given an initial state $\mathbf{x}_0 \in \mathcal{X}$ and an input $\mathbf{a} \in \mathcal{A}$, $\mathcal{A} = \mathcal{V}_x \times \mathit{\Omega} \times \mathcal{V}_z$ according to specific dynamics. To parameterize the operator input into a motion primitive, we have $\mathbf{a} = u_\mathrm{h}$, following Eq. 3.6; that is,

$$\gamma_{\mathbf{a},T} : [0, T] \to \mathcal{X} \qquad \mathbf{a} \in \mathcal{A},\ T \in [0, \infty),\ \mathbf{x}(0) = \mathbf{x}_0 \tag{7.2}$$

$$\text{s.t. } \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{a}) \tag{7.3}$$

To parameterize the operator input into a motion primitive, we set $\mathbf{a} = u_\mathrm{h}$. Each motion primitive γ and trajectory ξ are generated such that we retain continuity up to snap.

7.3.1 Topological map representation

We seek a minimal map representation that contains topologically distinct paths given constrained environments. This means that paths in the map must be *homotopically distinct*, i.e., every path represents a different homotopy class[1]. Therefore, we assume the existence

[1] Two paths are said to be in the same homotopy class iff one can be smoothly deformed into the other without intersecting obstacles [125].

of a semantically topological navigation graph, defined as follows:

Definition 1 (Semantically topological navigation graph) *A* semantically topological navigation graph *is a minimally representative, undirected navigation graph where each path formed by connected vertices are homotopically distinct, and represent free-space that is traversable.*

Semantically topological navigation graphs are highlighted by two key requirements. First, the paths are homotopically distinct, and second, they are *traversable*. This implies there are no edges that lead to geometrically distinct but untraversable[2] regions.

An illustration of these requirements are shown in Fig. 7.5.

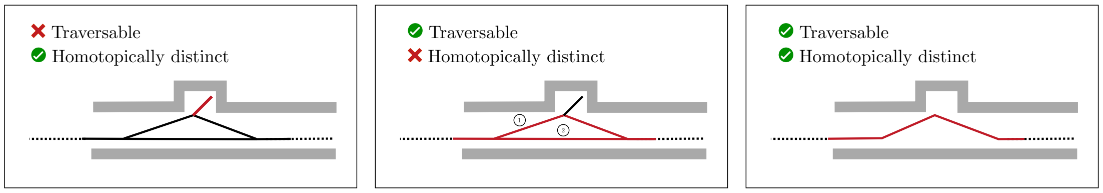


Figure 7.5: The illustrated paths violate the requirements for a semantically topological navigation graph. (Left) The red path, although homotopically distinct from the other paths, leads to an untraversable corner. (Center) Paths 1 and 2 are traversable; however, they are homotopically equal. (Right) Map satisfying both requirements.

Such a map can be obtained a priori, or generated using topological exploration [123]. For this work, we assume that a semantically topological navigation graph is provided.

7.3.2 Independent given irrelevant alternatives.

The above assumptions with respect to homotopic classes of paths give rise to an important characteristic of the paths on the navigation graph. The homotopically distinct paths can be assumed to be *independent given irrelevant alternatives (IIA)*[3]; that is, each choice of path is independent and are non-substitutable given the agent's preference for each choice.

An example is illustrated in Fig. 7.6. In the left scenario, imagine the paths A, B, and C lead to three different aisles, separated by shelves. These three paths cannot be interchanged

[2] *Traversability* in this context is conditional on the robot geometry and dynamics. For example, extreme rugged terrain may not be traversable for a limited power mobile robot. In 3D, an opening much smaller than the robot geometry is non-traversable. This is to say, traversability is not only defined as a function of the environment, it is also dependent on the robot.

[3] IIA implies that adding another option, or changing the characteristics of a third option (C) does not affect the probability between the first two options (A, B) considered. If the third option C is *similar* to the first two options A and B, then the individual preference for A and B will be affected by the new addition of option C.

to replace one another for tasks that specifically require one of these aisles. Therefore, if it is path B that is chosen, the addition of path C does not change that B is preferred over A and C.

In the scenario on the right, however, path A and B exist within the same homotopy class and span the open space between obstacles. The operator may choose path A over B. However, if a new path C is added, the agent may decide that it is more preferable to choose the middle path. Therefore, B becomes more preferred over path A by adding a third choice, C. Such scenarios violate IIA.

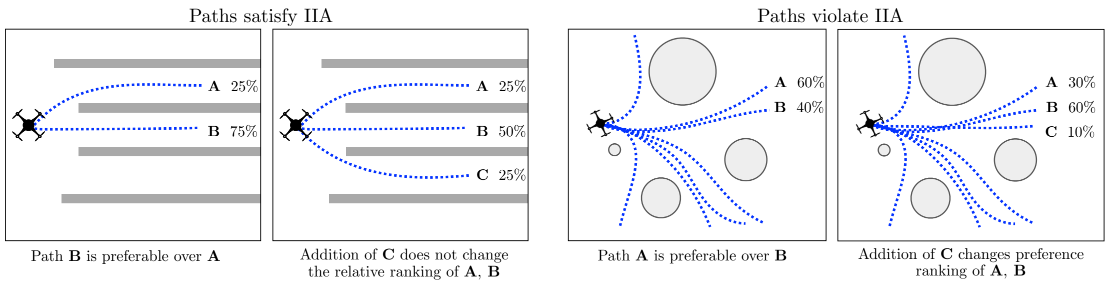

Paths satisfy IIA
A 25%
B 75%
Path **B** is preferable over **A**
A 25%
B 50%
C 25%
Addition of **C** does not change the relative ranking of **A**, **B**
Paths violate IIA
A 60%
B 40%
Path **A** is preferable over **B**
A 30%
B 60%
C 10%
Addition of **C** changes preference ranking of **A**, **B**

Figure 7.6: A hypothetical example of sets of paths that satisfy and violate the IIA assumption. (Left) Addition of path C does not change the relative ranking between B and A. (Right) By adding path C, the choice of paths is influenced and B is now preferred over path A.

The implication of the IIA assumption is that the human's choice model can be modeled using the Boltzmann rationality decision model [134], discussed below.

7.3.3 Boltzmann rationality decision model

Human decision making process is most commonly modeled as a Boltzmann rationality decision model, where the human is assumed to approximately optimize a reward function given a set of choices [52]. This particular model is derived from Luce's choice axiom [46], which models the probability of selecting one option from a set of multiple discrete and distinct options: Given a set of options $\mathcal{O}$, the weight of each option $o \in \mathcal{O}$ is given by $w : \mathcal{O} \rightarrow \mathbb{R}^+$. Therefore, the probability that the human will select any particular option o is given by:

$$P(o) = \frac{w(o)}{\sum_{o' \in \mathcal{O}} w(o')} \tag{7.4}$$

The exponential reward formulation of the Luce choice axiom gives rise to the Boltzmann noisily-rational policy, where the human's intention is modeled as selecting an option in proportion to their exponentiated reward [45, 47]:

$$P(o) = \frac{\exp(R(o))}{\sum_{o' \in \mathcal{O}} \exp(R(o'))} \tag{7.5}$$

7.4 Approach

We propose a continuous dynamic autonomy framework given a semantically topological navigation graph. The framework involves two parts: First, generating a path prediction on the navigation graph, and based on the prediction, generate a dynamically feasible trajectory. The continuous dynamic autonomy framework expands on the hierarchical HITL method, and utilizes a predicted path on the topological navigation graph as the global path.

Given a semantically topological navigation graph and the robot's current state, we extract a set of possible paths from the graph. We infer the likelihood of each path given the vehicle's previous odometry, and validate the prediction using the operator input. The predicted path on the navigation graph represents the operator's long horizon intention, and becomes the global path such that the local trajectory generated follows the global path autonomously. Depending on the certainty of prediction as validated by the operator's input, we utilize a *dynamic autonomy* framework: If the outcome is uncertain, then instead of following the global path, the operator's input is directly used to generate local trajectories, until a prediction becomes confident.

In the following sections, we describe a novel path prediction on a navigation graph, and describe continuous dynamic autonomy given the predicted outcome and the human input.

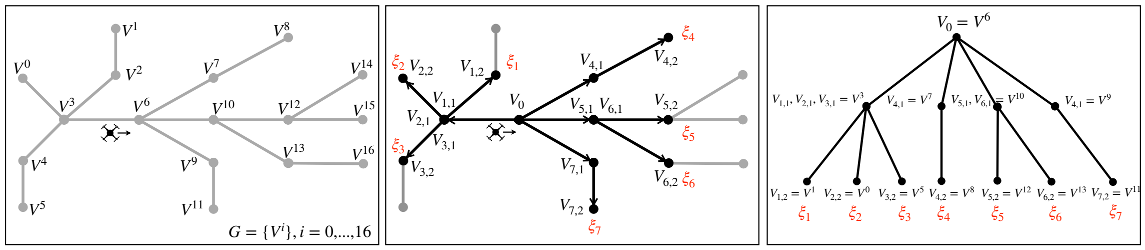
7.4.1 Path prediction on navigation graph

Path extraction from Graph. Given an unstructured environment, we represent the traversable free space by a semantically topological graph $G = \{V^k\}, k \in [1, K]$, an undirected graph represented by a set of K vertices V^k.

As the robot traverses the environment, a set of available paths can be extracted from the graph: As the robot approaches a vertex, a *directed tree* with a finite horizon is formed with the approaching vertex being the root node of the directed tree. The directed tree forms a set of paths, with each path represented by sequential vertices extracted from the graph.

Denote the root of the directed tree at time t as V^0. The set of paths is represented by $\varXi = \{\xi_j\}, j \in [1, J]$. The j-th path ξ_j represented by sequential vertices, s.t., $\xi_j = [V_{j,0}, V_{j,1}, ..., V_{j,N_j}]$. Alternatively, we represent the path ξ_j as a sequence of n_j line segments $\xi_j = [s_{j,1}, ..., s_{j,N_j}]$, anchored by connected vertices $s_{j,i} = \{V_{j,i-1}, V_{j,i}\}$. Note that, each one of the vertices in the path $V_{j,i} \in \xi_j$ will map to a vertex on the graph, $V^k \in G$, however, many vertices on the path may map to the same node on the graph as segments are shared between paths. See Fig. 7.7 for a detailed illustration of the path tree extraction.

Receding horizon observation model. The receding horizon observation model proceeds as follows: At time t, given a window of M state observations $x_{\{t-M\}:t}$, we evaluate


V^0 V^1 V^2 V^3 V^4 V^5 V^6 V^7 V^8 V^9 V^{10} V^{11} V^{12} V^{13} V^{14} V^{15} V^{16}
$G = \{V^i\}, i = 0,...,16$
V_0 $V_{1,1}$ $V_{1,2}$ $V_{2,1}$ $V_{2,2}$ $V_{3,1}$ $V_{3,2}$ $V_{4,1}$ $V_{4,2}$ $V_{5,1}$ $V_{5,2}$ $V_{6,1}$ $V_{6,2}$ $V_{7,1}$ $V_{7,2}$
ξ_1 ξ_2 ξ_3 ξ_4 ξ_5 ξ_6 ξ_7
$V_0 = V^6$
$V_{1,1}, V_{2,1}, V_{3,1} = V^3$
$V_{4,1} = V^7$
$V_{5,1}, V_{6,1} = V^{10}$
$V_{4,1} = V^9$
$V_{1,2} = V^1$
$V_{2,2} = V^0$
$V_{3,2} = V^5$
$V_{4,2} = V^8$
$V_{5,2} = V^{12}$
$V_{6,2} = V^{13}$
$V_{7,2} = V^{11}$

Undirected navigation graph G Directional subgraph given root node V^0 Path tree representing 7 paths

Figure 7.7: Extracting a path tree from a topological graph, given the current vehicle position. From a single node, the undirected graph can be turned into a directed tree.

each path with an evaluation cost function. For simplicity, we will drop the superscript of the past observations $x_{\{t-M\}:t}$ for readability.

Given a cost evaluation function c, we integrate the cost along the path:

$$C(x,\xi) = \int_{\xi} c(x,\xi)w(\xi)d\xi \tag{7.6}$$

Notice the addition of a weighting function $w(\xi)$. Our observation model is a function of the previous state observations x, therefore further down in the future, the previous observations x are less likely to affect choices and costs in the future. The weighting function considers the effects of time on the cost.

This can be normalized to a path integral, and the weighting function can simply be a function of the time parameterization t s.t., $w(t) : [0,1] \to \mathbb{R}^+$:

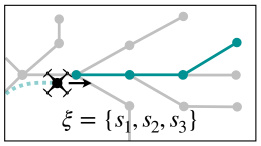
$$C(x,\xi) = \int_{t} c(x,\xi(t))w(t)dt \qquad t \in [0,1] \tag{7.7}$$

For a discretized path $\xi = [s_1, s_2, ...s_N]$ with N segments, this becomes:

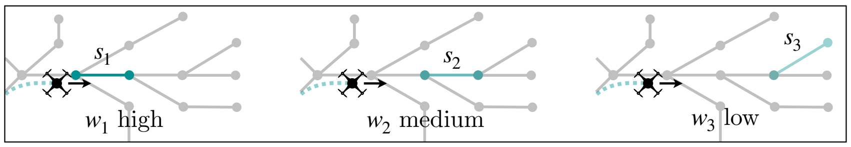
$$C(x,\xi) = \frac{1}{N}\sum_{n=1}^{N} c(x,s_n)w(s_n) \tag{7.8}$$

This effectively says: for each path with N segments, we compute a weighted sum over the segment wise cost $c(x,s_n)$.

Path prediction. Given a set of paths Ξ extracted from the graph G, we now compute the probability of a path $\xi \in \Xi$ given the robot's observations x by using the Boltzmann rationality decision model. Following Bayes rule,



(a) 3-segment path



(b) Decreasing influence (weight) of previous odometry on future segments

Figure 7.8: Illustration of decreasing weight factor for evaluating discrete path segments wrt observations, as the observations are less likely to affect segments in the future.

$$p(\xi|x) = \frac{p(x|\xi)p(\xi)}{p(x)} \qquad \xi \in \Xi \tag{7.9}$$

We follow the principle of max entropy to induce a distribution over the set of paths, which says the probability of a path decreases exponentially with the cost C. The likelihood, $p(x|\xi)$, is given by:

$$p(x|\xi) \propto \exp(-C(x, \xi)) \tag{7.10}$$

Substituting Eq.(7.10) back into Eq. (7.11):

$$p(\xi|x) = \frac{p(x|\xi)p(\xi)}{p(x)} \qquad \xi \in \Xi^V \tag{7.11}$$

$$= \eta \cdot p(x|\xi)p(\xi) \tag{7.12}$$

$$= \eta \cdot \exp\left(-\frac{1}{N}\sum_{n=1}^{N} c(x, s_n)w(s_n)\right) p(\xi) \tag{7.13}$$

Compare this formulation with Eq. 7.5; this is effectively the Boltzmann model with a prior.

Segment wise cost $c(x, s)$. We now discuss a simple evaluation cost function for Eq. 7.8 by vector projection. Given a segment of the path s and a path x, we sample N discretizations along the observation and the path segment, s.t., $\vec{x}_i = x_i - x_{i-1}$, and $\vec{s}_i = s(i) - s(i-1)$ with $s(i)$ indicates the ith discretization of the segment s. A point-wise projection is given by:

$$\text{proj}(x, s) = \frac{1}{D} \sum_{i=1}^{D} \frac{\vec{x}_i}{\|\vec{x}_i\|} \cdot \frac{\vec{s}_i}{\|\vec{s}_i\|} \tag{7.14}$$

$$= \frac{1}{D} \sum_{i=1}^{D} \hat{\vec{x}}_i \cdot \hat{\vec{s}}_i \tag{7.15}$$

Finally, we shift the projection such that the cost is positive:

$$c(x, s) = (1 - \text{proj}(x, s)) \tag{7.16}$$

7.4.2 Continuous Dynamic Autonomy

When humans are interacting with the robot, their actions give great insight with respect to their underlying intention. Assuming the human is observant and not distracted, they could provide no inputs as the robot navigates, which implies that they agree with the motion of the vehicle. However, if they provide an input, this indicates that they disagree with the motion of the vehicle and would like to change it. This information can be used in two different ways: First, how much the operator agrees with the current path of the vehicle, and second, what motion the operator prefers instead.

We leverage this key insight in constructing our dynamic autonomy framework. The continuous dynamic autonomy architecture is shown in Fig. 7.9. The system continuously predicts paths on the topological graph, and generates local trajectories accordingly. The operator inputs are continuously evaluated against the prediction. If the prediction does not match with the operator's intended direction, a trajectory is generated according to the operator's direct input. The vehicle follows the commanded trajectory smoothly and without discrete stops.

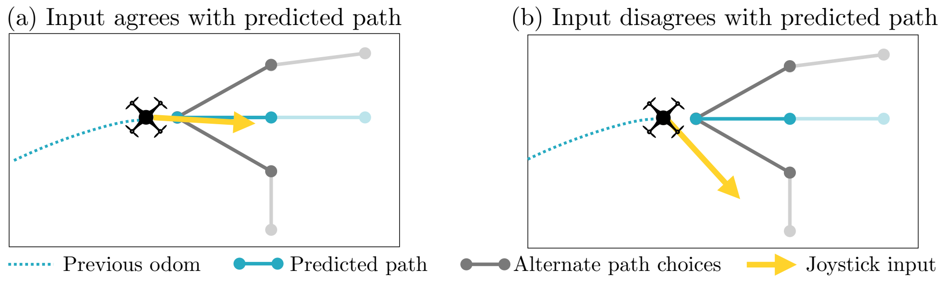

(a) Input agrees with predicted path
(b) Input disagrees with predicted path
Previous odom
Predicted path
Alternate path choices
Joystick input

Figure 7.9: Illustration of prediction diverging with human preference.

Prediction checking. The output prediction is checked against the operator input continuously. For each input u, we evaluate whether the immediate next step agrees with the user input. We treat the first segment of the path as the immediate next step. To evaluate agreement, we compute the dot product of the joystick input u_{h} with the normalized vector representing the first segment of the path:

$$u_{\mathrm{h}} \cdot \overrightarrow{S_j} = u_{\mathrm{h}} \cdot \frac{V_{j,1} - V_{j,0}}{\|V_{j,1} - V_{j,0}\|}$$

However, we must take into account the total number of paths that is available to choose from. If the input most agrees with the predicted path, we can understand it to follow the best possible path. Therefore we evaluate agreement by normalizing over all of the first segment paths stemming from the current vertex:

$$w_{\mathrm{agreement}}(\xi_j, u_{\mathrm{h}}) = \frac{(1 + u_{\mathrm{h}} \cdot \overrightarrow{S_j})/2}{\sum_j (1 + u_{\mathrm{h}} \cdot \overrightarrow{S_j})/2} \quad \xi_j \in \varXi \tag{7.17}$$

for $\varXi$ is the set of paths stemming from the current vertex V_0.

Trajectory generation If the joystick input agrees, then we have fully autonomous operation where the predicted path ξ^* is used as the global path guiding local trajectory generation. Any local trajectory generation method can be used. For this paper, we use Biased Incremental Action Sampling (BIAS) introduced in Chapter 5 and used in the hierarchical setting, as in Chapter 6.

If the operator input disagrees with the first segment of prediction, the prediction is discarded, and the operator input u_{h} is directly used to generate a motion primitive, following [19]. If the primitive is not collision-free, then BIAS is again invoked to generate collision free trajectories.

The algorithm of continuous dynamic autonomy is illustrated in Algorithm 6.

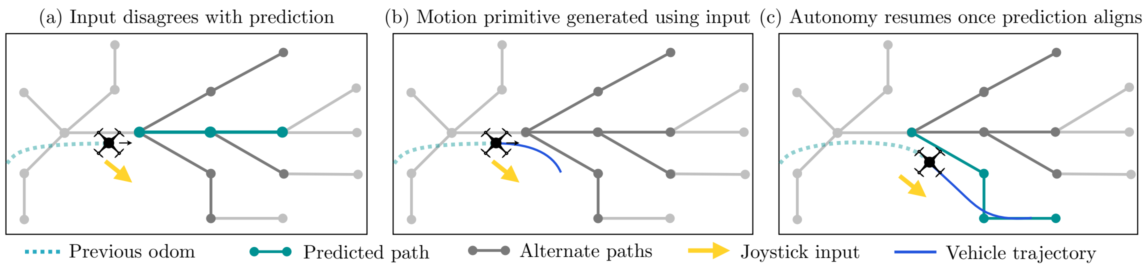

(a) Input disagrees with prediction
(b) Motion primitive generated using input
(c) Autonomy resumes once prediction aligns
Previous odom
Predicted path
Alternate paths
Joystick input
Vehicle trajectory

Figure 7.10: Illustration of the continuous dynamic autonomy transition. As the predicted path is invalid given the operator input, the input is parameterized to generate a dynamically feasible motion primitive. As the prediction becomes confident again, the autonomous hierarchical framework takes over: a local trajectory is generated to follow the predicted global path.

Algorithm 6: Continuous Dynamic Autonomy

```
   Input: Given a topological navigation graph $G$
1  while Robot navigating do
2      Prediction
3      Extract available paths given vehicle state $x_t$: $\Xi = \{\xi\}$
4      Compute path prediction $p(\xi|x)$ via Eq. 7.9
5      Evaluate most likely path $\xi^* = \text{argmax}_\xi\, p(\xi|x)$
6      Agreement
7      if $\xi^*, u_t$ agree then
8          Set global path $\xi_G = \xi^*$
9          Generate local trajectory using BIAS $\zeta$
10     else
11         Set local trajectory to parameterized motion primitive $\zeta = \gamma(u_t, x_t)$
12     return $\zeta$
```

7.5 System Design

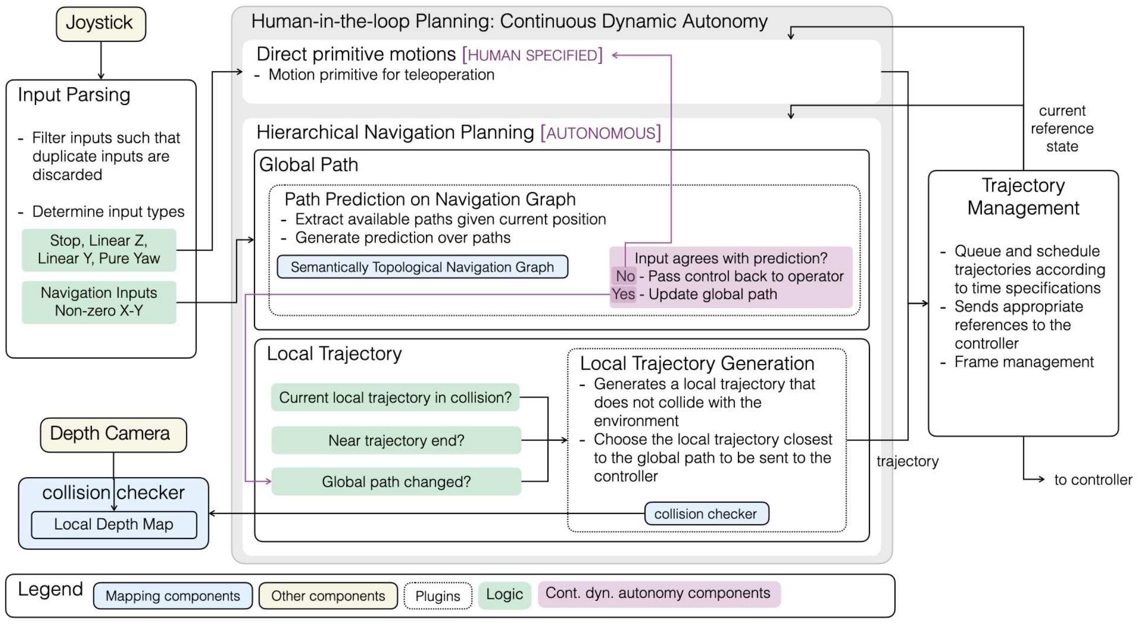

Joystick
Input Parsing
- Filter inputs such that duplicate inputs are discarded
- Determine input types
Stop, Linear Z, Linear Y, Pure Yaw
Navigation Inputs Non-zero X-Y
Depth Camera
collision checker
Local Depth Map
Human-in-the-loop Planning: Continuous Dynamic Autonomy
Direct primitive motions [HUMAN SPECIFIED]
- Motion primitive for teleoperation
Hierarchical Navigation Planning [AUTONOMOUS]
Global Path
Path Prediction on Navigation Graph
- Extract available paths given current position
- Generate prediction over paths
Semantically Topological Navigation Graph
Input agrees with prediction?
No - Pass control back to operator
Yes - Update global path
Local Trajectory
Current local trajectory in collision?
Near trajectory end?
Global path changed?
Local Trajectory Generation
- Generates a local trajectory that does not collide with the environment
- Choose the local trajectory closest to the global path to be sent to the controller
collision checker
current reference state
Trajectory Management
- Queue and schedule trajectories according to time specifications
- Sends appropriate references to the controller
- Frame management
trajectory
to controller
Legend
Mapping components
Other components
Plugins
Logic
Cont. dyn. autonomy components

Figure 7.11: System Diagram for HITL dynamic autonomy with prediction on semantically topological navigation graphs.

The HITL system is illustrated in Fig. 7.11.

The operator inputs are received as a continuous stream of joystick values, indicating linear, angular, and z velocities. The operator's inputs are first categorized into direct primitive motion inputs, and navigation inputs. The direct primitive motion inputs are followed directly; these are: in-place yaw motions, translation along the z-axis, translation along the body y axis, and a zero input to indicate a stop for the vehicle. This enables ease of use for achieving human objectives such as inspection and exploration, as well as to stop the vehicle. The navigational inputs involve motions in the x-y plane, which indicates an intention to navigate. These inputs invoke the hierarchical framework for navigation.

During robot navigation, hierarchical navigation planning is invoked. A global path, ξ_G guides the generation of local trajectories ζ. The local trajectories ζ are dynamically feasible and collision-free, while the global path represents a general intended motion for a longer horizon. The global path is generated via path prediction on a topological navigation graph as described in Section 7.4.1. If the input agrees prediction, global path is updated to be the outcome of the prediction. Otherwise, control is handed back to the operator. Although this process is discrete, the human is always interacting with system via joystick, therefore achieving *continuous-interaction.* Further, the trajectories generated either from a local trajectory or human-input parameterized motion primitive are guaranteed to be continuous up to snap, therefore achieving *continuous-control.*

By design, the continuous dynamic autonomy is only invoked during navigation. If the operator wishes to stop and perform in-place linear motions, the system fully allows the operator to do so without having to switch into discrete modes of operation.

7.6 Experiments

7.6.1 Experiment design

The task is to navigate in a densely cluttered warehouse environment (45m×22m×11m) in simulation following a path described verbally to the operator and indicated by yellow arrows, and return to the red landing pad at the origin as shown in Fig. 7.12. The operators

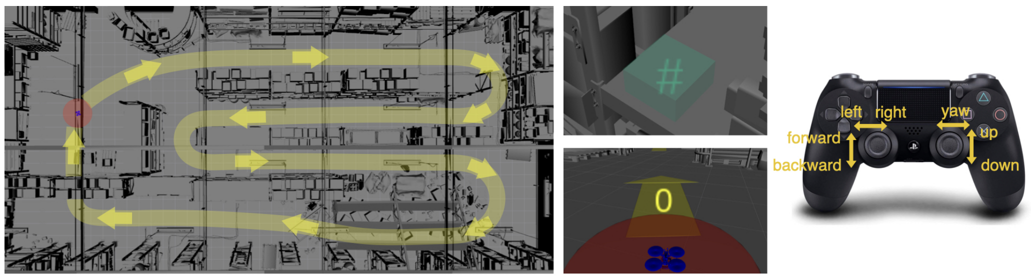


Figure 7.12: Experiment setup: The task is to navigate in a densely cluttered warehouse environment (45m×22m×11m) in simulation following a path described verbally to the operator and indicated by yellow arrows, and return to the red landing pad at the origin. The secondary task is to identify a green box with a symbol. The operator is given a third person omniscient view, and the joystick inputs are given in the body frame of the vehicle.

were also asked to look out for a randomly placed green box along the path as a secondary task, in order to simulate attention division similar to search-and-rescue. The simulated quadrotor is controlled via a joystick, specifying the forward, side, angular, and z velocities scaled according to a max velocity parameter, set to 1.5 m/s. The operator is given a third-person omniscient view with only the guiding arrows visible (Fig. 7.12). The generated trajectory was chosen to be *hidden* so as to simulate control without visual aids. This design choice and its implications will be discussed later in the results section. The semantically topological navigation graph used for the proposed method is shown in Fig. 7.13. Paths extracted from the graph are approximately 10m in length.

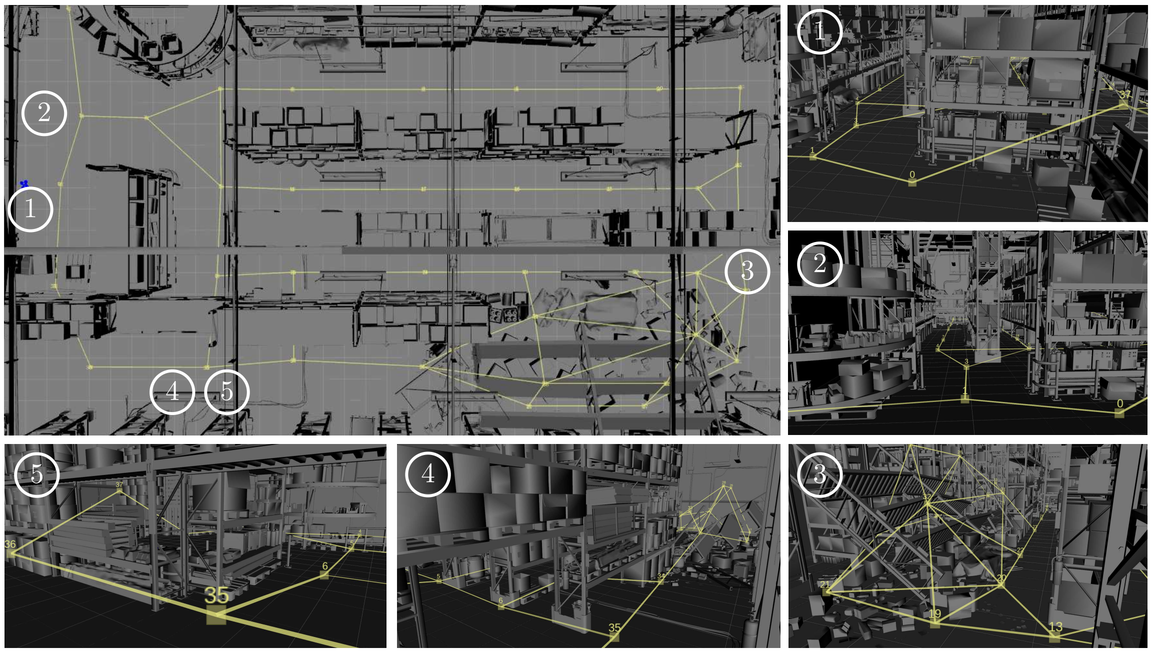


Figure 7.13: Semantically topological navigation graph for the warehouse. Cutaway views of the navigation graph is shown at various viewpoints.

7.6.2 User study design

We conduct a pilot study ($n = 10$) to evaluate the proposed method, continous dynamic autonomy (DA) method against two methods of trajectory-based teleoperation: motion primitive teleoperation (MP) [19], and a velocity based teleoperation method (VEL). All three methods are continuous up to snap. For all three methods, the joystick interface remains the same. However, the underlying dynamic models that generate the trajectories are different: DA and MP both utilize the unicycle model, whereas for the VEL method, the yaw is decoupled from the heading.

The participants have no prior exposure to our system but have varying experience with

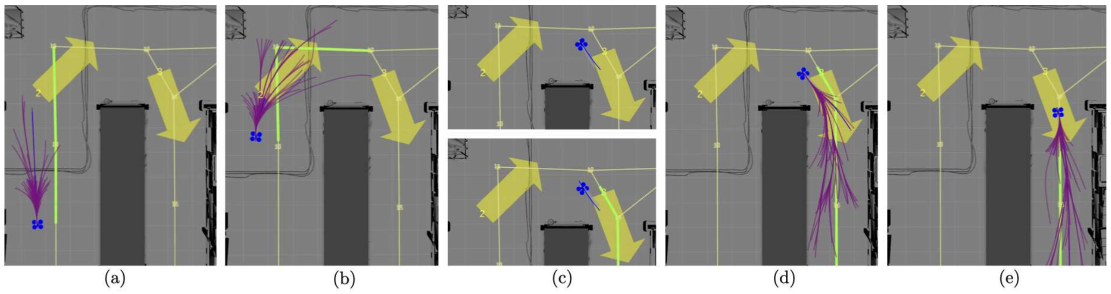


Figure 7.14: A sequence of continuous dynamic autonomy switching for rounding a corner. The operator disagrees with the predicted path (green) from (b) to (c), and thus a human-parameterized trajectory (blue) is generated. As the prediction aligns again, autonomy resumes (d-e).

teleoperating quadrotors. The pilot follows a within-subjects design, where each participant used all three methods, A, B, and C. The ordering of the methods was randomized, such that A, B and C corresponded to one of VEL, MP, and DA. Prior to each trial, the participant was given a tutorial period of 3 minutes to test out the control and dynamics. As the controls required for all three systems are the same, no additional details were provided about the controls. However, note that, since the underlying dynamic model that generates the trajectories for MP and DA are the same, the control during the tutorial period is exactly the same.

7.6.3 Hypotheses

The experiments in this pilot study aim to evaluate human-robot efficiency by way of operator engagement. The operator engagement is evaluated by using number of joystick inputs used per trial. Operator preference is evaluated via a survey post-trial. The hypotheses are:

H1 The system will require *less* direct human control and will navigate *mostly with autonomy* with DA.

H2 Operators will engage with the system *less* when using DA, leading to reduced number of inputs during navigation.

H3 Participants will prefer DA over direct control methods VEL and MP.

7.6.4 Results

7.6.5 Operator Behavior over Time

The operator behavior over time using continuous dynamic autonomy, DA, is shown in Fig. 7.15. These plots show the operator's behavior over time with respect to (1) The number of possible path choices, and (2) the operator's input and how it agreed with the predicted path. As the vehicle approaches a node, the number of possible path choices is computed as the number of possible path stemming from the node given a fixed path length. The number of immediate branching factor denotes the number of edges stemming from the current node. The significance of the path choices indicates complexity of the environment, whereas the immediate branching factor indicates the number of immediate choices that is available to the operator.

As the vehicle approaches a point with many decision points, the operator tends to slow down and stop, observe, and proceed with caution. This behavior is qualitatively observed across multiple participants with varying degrees of cautiousness (high caution is indicated by many stops). However this behavior is not limited to DA; it is also observed across all three methods (Fig. 7.16).

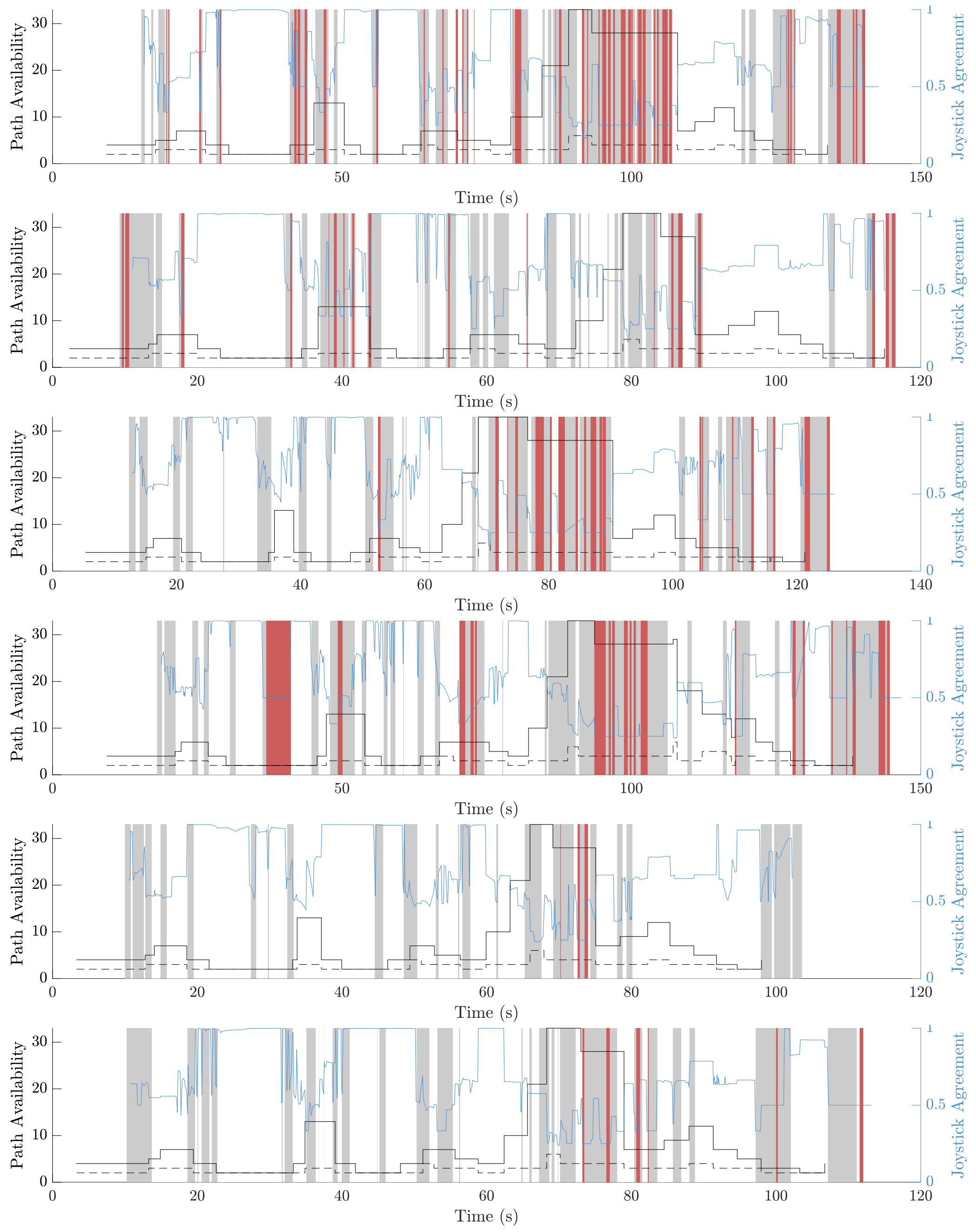


Figure 7.15: Ooperator behavior over time for two select trials, highlighting two observations. 1) The number of path choices/immediate edge choices vs. the type of motion (shaded), Observe that the vehicle is stopped more frequently near areas of increased path choices. 2) Joystick agreement vs. type of motion. As prediction and human's inputs align, vehicle navigates mostly autonomously.

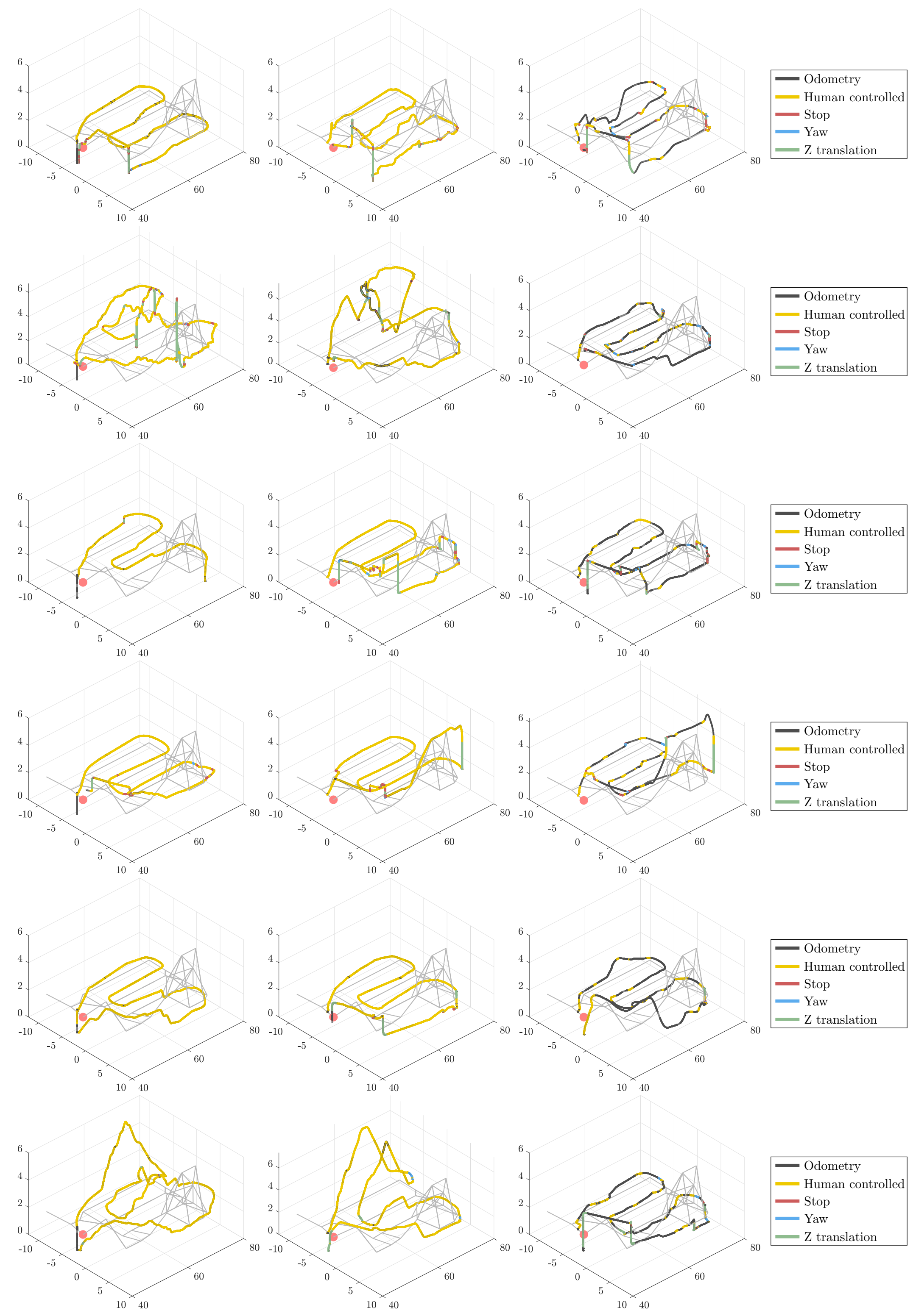


Figure 7.16: Odometry of participant trials for select participants. (left) VEL (center) MP (right) DA

7.6.6 Odometry Evaluation

We categorize motion into 4 modes: (1) **Stop** (2) **In-place yaw**, (3) **Z** and (4) **Nav**, which means that the input provided includes non-zero inputs along the x-y plane. We primarily focus on **Nav**, as DA is invoked only during navigation.

Fig. 7.16 shows some example odometries of the three methods. We observe that the vehicle is mostly autonomous for DA. We compute the amount of navigation done by human control. This result is tabulated in Table 7.1, with an accompanying bar plot in Fig. 7.17. These results were assessed using a one-way repeated measures ANOVA. The results showed that the proposed method was able to reduce the human's role in navigation control from 86% to 24.5% ($F(2, 34) = 88.0$, $p < .001$), supporting **H1**.

Table 7.1: Breakdown of odometry in each mode as a percentage of the total trajectory length. A stacked bargraph is plotted in Fig. 7.17 for visual comparison.

	Stop	**Yaw**	**Z**	**Nav Human**	**Nav Autonomy**
VEL	$5.1\% \pm 3.8\%$	$0.7\% \pm 0.8\%$	$2.2\% \pm 2.7\%$	$92\% \pm 5.5\%$	0.00
MP	$5.6\% \pm 4.7\%$	$1.4\% \pm 0.6\%$	$3.9\% \pm 1.7\%$	$89.1\% \pm 5.8\%$	0.00
DA	$7.5\% \pm 5.4\%$	$2.1\% \pm 1.4\%$	$4.5\% \pm 2.4\%$	$24.5\% \pm 6.4\%$	$62\% \pm 7.5\%$

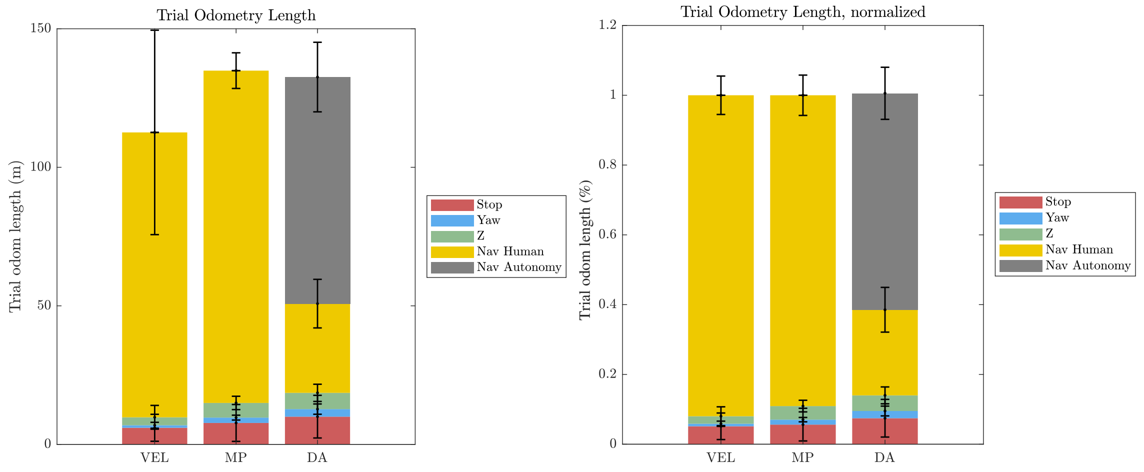


Figure 7.17: Odometry length, broken down by modes. The human controlled navigation (Nav Human) is reduced significantly, with operator directly controlling navigation approx 24.5% of the time. The tabulated result of this graph is shown in Tab. 7.1.

7.6.7 Trends in Operator Engagement

The number of inputs corresponding to each mode is shown in Table 7.2. These results were assessed using a one-way repeated measures ANOVA. While we observe that the number of inputs received during **Nav** is lower than both of the comparison methods, this data is not supported by statistical significance and should be noted as a trend. We believe that this is due to the relatively small sample size and high variance in the number of inputs. Thus, **H2** could not be supported at this time. However, this trend combined with confirmed **H1** indicates that the system's assistance is effective at increasing human-robot efficiency. To validate this trend, we believe the proposed method could benefit from a further study with larger sample sizes to further confirm this hypothesis with statistical significance.

Table 7.2: Number of inputs for the warehouse navigation task

	Stop	**Yaw**	**Z**	**Nav**	**Total**
VEL	25 ± 19	14 ± 11	18 ± 14	219 ± 100	277 ± 113
MP	47 ± 57	50 ± 48	50 ± 46	238 ± 134	384 ± 234
DA	46 ± 36	31 ± 21	43 ± 22	187 ± 48	307 ± 115

7.6.8 Preference and Qualitative Observations

We additionally evaluate operator preference given the three methods. The post-trial survey asked the following questions for each method:

1. I find the controls to be natural/intuitive.
2. I find the controls to be comfortable to use.
3. I was able to stabilize the vehicle with ease.
4. I was able to navigate the vehicle with ease.
5. I was able to avoid obstacles with ease.
6. The vehicle performed the motion that I intended for it to do.

The results are shown in Fig. 7.18, with a strong support against **H3**. The trends suggest that the operators preferred VEL, with the yaw decoupled from the vehicle's heading. The proposed method, DA, was thought to be more difficult to use. This result leads to a *subjective vs. objective gap* in DA's perceived helpfulness. We reason about this gap via the following observations:

Behaviors of naive vs. experienced operators Naive operators tend to act more carefully by provide minor corrections. This is illustrated via "nudging" the system by flicking the joystick. Experienced operators are well versed with motion coupling and dynamics of the quadrotor. As the VEL system is a direct velocity parameterization of their inputs, they are more likely to prefer this method.

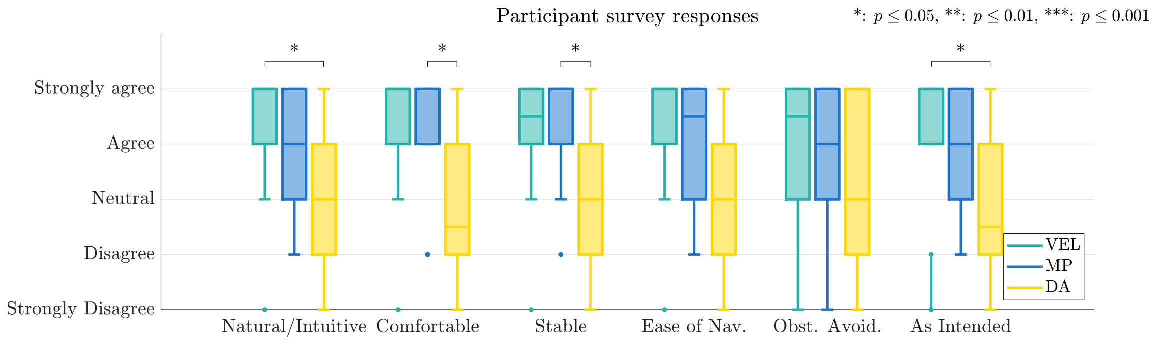


Figure 7.18: Participant survey responses to various questions regarding use, with each individual method evaluated separately. Participants preferred the proposed DA method the least given its inability to respond to individual nudges. F-values: Natural/Intuitive: $F(2,18) = 6.9, p < .01$; Comfortable: $F(2,18) = 6.8, p < .01$ Stable: $F(2,18) = 5.1, p < .05$; Ease of Nav: $F(2,18) = 5.9, p < .05$; As Intended: $F(2,18) = 7.6, p < .01$.

Table 7.3: User survey responses comparing the three methods. Each number indicate number of preferred ranking received for each method by the participants.

	Control				Obstacle Avoidance				Preference		
	Good	So-so	Least		Easy	So-so	Hard		Preferred	So-so	Least
VEL	3	2	0	VEL	3	2	4	VEL	5	3	2
MP	0	2	1	MP	3	4	0	MP	4	6	0
DA	0	3	5	**DA**	1	2	5	**DA**	1	1	8

Subjective perceptions of operators We note many subjective interpretations by the operators for the controls. For example, some participants remarked that "the controls (for DA) feels completely different than the previous (MP)" during the tutorial period. However, the controls and the underlying dynamics are exactly the same. Further, the resulting odometries of VEL and MP were qualitatively observed to be drifty and unstable – however, the operators still preferred them for controllability (Fig. 7.18). We hypothesize that, if we show videos recorded of the vehicle navigation from this study to a separate group of participants and ask which method they believe is more stable, they would perceive that DA is more stable than VEL or MP. We leave these investigations as future work.

Sensitivity of controls All three methods used the same set of parameters. Therefore, we attribute remarks on "sensitivity" to the differing trajectory generation methods: The participants expected the system to respond to minor adjustments in inputs. As the local trajectories generated by DA do not respond to minor adjustments, the participants remarked this to be unresponsive and difficult to control.

Interface design and behavioral changes The generated trajectory was not visualized to the operators. Therefore, many operators provided inputs based on what they perceive the robot will do at the immediate next step, even though the system's current trajectory was safe and ideal. We hypothesize that adding visual feedback of the trajectory would change the operator's interaction with the system and will help to increase "trust" of the robot, although we leave further investigations of these hypotheses to future work.

7.7 Conclusion and Future Work

This chapter presents a continuous dynamic autonomy framework, by generating path predictions on semantic topological maps. The contribution of this chapter is two folds: 1) Introduction of path prediction on navigation graphs by way of a simple receding horizon model; and 2) Continuous dynamic autonomy with control allocated fluidly between the human and the robot. This framework shows that complex environments with dense environment features can be eschewed in favor of simple representations that encode semantic traversability.

The results in this chapter yields some surprising discoveries: While the human-robot performed efficiently with assistance, operators did not like having a system that was non responsive to small control input changes. These observations give great insight to what it may mean to how humans visualize robot motions and how they wish to interact with robots.

Chapter

8

Conclusion

This thesis introduces a new human-in-the-loop control paradigm by way of modeling human intention in navigation as non-goal specifications, and generating trajectories that satisfy the human intention. We first introduced methods that simplify the human-in-the-loop control interface by way of encoding operator inputs into motion primtiives (Chapter 3 and 4). Then, we introduced methods to generate locally collision-free trajectories (Chapter 5) and extended motion planning for HITL to include global paths (Chapter 6) as well. Lastly, we incorporated environment context as a part of human-in-the-loop via a semantic topological navigation map. Inference on such maps is valuable for predicting human intention during navigation, as contextual cue is a common denominator available to both the human and the robot during navigation.

8.1 Summary of Contributions

8.1.1 Motion Primitives based HITL Control

Chapter 3 presents a method for direct HITL control by parameterizing operator inputs to map to a set of motion primitives, generated using a selected dynamics function. Unintuitive system dynamics hinder human operators to be able to control a system with precision; this chapter showed that by remapping inputs to a more intuitive system model via motion primitives, the human-robot system can perform with significantly increased efficiency. This chapter further developed key ideas relating to MAV control and trajectory generation: Other dynamics models (e.g., unicycle model) can be directly translated to multirotor UAVs by leveraging differential flat properties of multirotors. Additionally, this chapter develops a method for generating snap-continuous motion primitives, which allows for computing feedforward references up to angular acceleration, as well as continuous error functions for smoothness of control.

Chapter 4 extends Chapter 3 to reactive collsion avoidance. Construction of motion primitives establishes a direct correspondence between the action space and the state space of the robot. Hence, reactive collision avoidance can be achieved via reactive pruning of motion primitive libraries. Operator input is then mapped to the closest safe primitive, guaranteeing collision-free navigation even though the operator's direct inputs are unsafe. The proposed method led to surprisingly natural behaviors in the presence of obstacles: If the vehicle is headed towards a wall, then the vehicle will choose linear velocities that gradually decrease until the vehicle is stopped, even though the human has not specified such action. If the vehicle is headed to pass through an narrow opening between two pillars, the vehicle will slow down to thread through and speed up once the opening is passed.

8.1.2 Local trajectory generation and Hierarchical Motion Planning for HITL

Instead of taking reactive measures to assist the operator from collisions, Chapter 5 proposes generating predictive long horizon trajectories that align with the operator's directional intent while circumventing obstacles. The key constraints that arise from having a human control the robot, are that the trajectories must be *legible*, and must be able to optimize for non-goal based specifications. To satisfy these requirements, this chapter proposes a computationally efficient way to generate trajectories by concatenating sequential motion primitives using a novel method called Biased Incremental Action Sampling. BIAS allows iterative construction of a motion primitive tree. The resulting tree contains various length trajectories that are smooth, continuous up to snap, and approximately optimize the human's directional objective.

Chapter 6 develops a hierarchical framework that includes global path representation. By including a global path, we remove the human operator from using dynamic-level control input, and instead utilize the inputs to *inform* trajectory generation. The key insight we exercise in this chapter is that, we can represent the operator's directional intention as the global path, and interpret local trajectories as a set of sequential actions that must be taken to satisfy the operator's intention. This framework allows teleoperation to move from an operator-controlled vehicle, to an operator-informed motion generation model.

8.1.3 Incorporating Contextual Information for Continuous Dynamic Autonomy

Lastly, Chapter 7 develops a framework for incorporating contextual information via semantic navigation graphs. The existence of partially available map information provides rich, contextual information about the environment that could assist the human further in the human-in-the-loop context. In constrained environments, the set of paths in the environment that the robot can undertake are enumerable and distinct. Therefore, the environment information can be succinctly condensed into a semantically representative topological navigation map. Hence, the role of the human can be reduced to selecting from a set of discrete, traversable paths. This chapter lays out a method for path prediction on semantically topological navigation graphs, and utilizing it within a framework of continuous dynamic autonomy.

8.2 Discussion

This thesis aim to increase the efficiency of human-robot system for navigation. In order to do so, the robot and human need to operate seamlessly in a continuous fashion. The robot needs to act according to the human's intention during every step of the navigation task. Given the myriad of competing objectives that the human-robot system faces, this thesis proposes shifting as much responsibility from the human to the robot as possible with autonomy. Throughout the chapters, we showed that operator engagement with the system decreased as robot autonomy increased. Increasing robot intelligence by using contextual cues lead to a reduction in the complexity of the human's role. As robot intelligence and autonomy increases, so does the planning horizon. This correlation is a key consideration in designing human-robot systems. A summary of some of the key insights is provided in Table 8.1.

8.3 Future Directions

Much of this thesis focused on assistance by modeling the ideal path that the human envisions. However, the human's intention model is much more complex than just that. Throughout this thesis, we discovered that it is not sufficient to only model the human's intention on a path level. For example, goal modeling for certain applications is necessary

Table 8.1: Key Insights on Human-in-the-loop Navigation

Human control How does the human interface with the robot?	**Robot motion** How should the robot use the human input to generate motion?	**Intention** How should intention be represented and predicted for navigation tasks?
▶ Abstract away difficult dynamics ▶ Shift responsibilities from human to robot as much as possible ▶ Robot and human need to operate seamlessly in a continuous fashion	▶ Motions need to be legible and natural to the operator ▶ Human has a trajectory in mind – Robot needs to translate human input into that trajectory	▶ Increase robot intelligence by inferring and using contextual knowledge ▶ Robot needs to act according to the human intention at every level

for generating globally optimal plans. On the other hand, we have shown that, even though assistance is provided at a path level, humans would like to see their lower level inputs be followed by the robot as well. Human's intention is *multimodal*; regardless of the task, humans tend to have an ideal model of how to go about achieving it. For human-robot system to collaborate, the robot system needs to interpret the human correspondingly. This means that we must model intention along multiple axes. For mobile robots, having an appropriate intention representation at various timescales allows inference and prediction to happen at the corresponding frequencies to aid assistance at the various speeds.

Robots will always co-exist alongside humans. In order for the two agents to work collaboratively, we must understand and build models of intention and behavior of humans, and build implicit/explicit communication models of objectives and intention between human-robot systems. This thesis contributes methods and insights for future advancement in human-robot systems.

Bibliography

[1] Thomas B Sheridan. *Telerobotics, automation, and human supervisory control.* MIT press, 1992. 1.3, 2.3.1.1

[2] C C MacAdam. Understanding and Modeling the Human Driver. *Vehicle System Dynamics*, 40:1-3(January 2014):101–134, 2003. 1.4.1.2, 1.4.1.3

[3] Sanjiban Choudhury. Adaptive motion planning. 2018. 1.4.1.2

[4] Marcel Adam Just, Timothy A Keller, and Jacquelyn Cynkar. A decrease in brain activation associated with driving when listening to someone speak. *Brain research*, 1205:70–80, 2008. 1.4.1.3

[5] Gianluca Borghini, Laura Astolfi, Giovanni Vecchiato, Donatella Mattia, and Fabio Babiloni. Measuring Neurophysiological Signals in Aircraft Pilots and Car Drivers for the Assessment of Mental Workload, Fatigue and Drowsiness. *Neuroscience and Biobehavioral Reviews*, 44:58–75, 2014. 1.4.1.3

[6] Anca D Dragan, Kenton CT Lee, and Siddhartha S Srinivasa. Legibility and predictability of robot motion. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 301–308. IEEE, 2013. 1.4.2.2

[7] Siddarth Jain and Brenna Argall. Probabilistic human intent recognition for shared autonomy in assistive robotics. *ACM Transactions on Human-Robot Interaction (THRI)*, 9(1):1–23, 2019. 1.4.3.1, 6.1

[8] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared Autonomy via Hindsight Optimization. In *Proc. of Robot.: Sci. and Syst.*, Rome, Italy, July 2015. 2.3.1.1, 2.3.1.3

[9] Simon Thompson, Takehiro Horiuchi, and Satoshi Kagami. A probabilistic model of human motion and navigation intent for mobile robot path planning. In *2009 4th International Conference on Autonomous Robots and Agents*, pages 663–668. IEEE, 2009. 1.4.3.1, 6.1

[10] Andreea Bobu, Dexter RR Scobee, Jaime F Fisac, S Shankar Sastry, and Anca D Dragan. Less is more: Rethinking probabilistic models of human behavior. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 429–437, 2020. 1.4.3.1, 6.1

[11] Anirudh Vemula, Katharina Muelling, and Jean Oh. Modeling cooperative navigation

in dense human crowds. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1685–1692. IEEE, 2017.

[12] Brian D Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J Andrew Bagnell, Martial Hebert, Anind K Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3931–3936. IEEE, 2009. 1.4.3.1, 6.1, 7.2

[13] Alan J Hamlet and Carl D Crane. Joint belief and intent prediction for collision avoidance in autonomous vehicles. *arXiv preprint arXiv:1504.00060*, 2015. 1.4.3.1, 6.1

[14] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, 1(1): 1–14, 2014. 1.4.3.1, 6.1

[15] Glenn Wasson, Pradip Sheth, Majd Alwan, Kevin Granata, Alexandre Ledoux, and Cunjun Huang. User intent in a shared control framework for pedestrian mobility aids. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 3, pages 2962–2967. IEEE, 2003. 1.4.3.1, 6.1

[16] Glenn Wasson, Pradip Sheth, Cunjun Huang, Alexandre Ledoux, and Majd Alwan. A physics-based model for predicting user intent in shared-control pedestrian mobility aids. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 2, pages 1914–1919. IEEE, 2004. 1.4.3.1, 6.1

[17] Xuning Yang, Koushil Sreenath, and Nathan Michael. A framework for efficient teleoperation via online adaptation. In *IEEE Int. Conf. on Robot. and Autom. (ICRA)*, 2017. 1.5, 3

[18] Xuning Yang, Ayush Agrawal, Koushil Sreenath, and Nathan Michael. Online adaptive teleoperation via motion primitives for mobile robots. *Special Issue on Learning for Human-Robot Collaboration, Autonomous Robots*, 2018. 5.1

[19] Alex Spitzer, Xuning Yang, John Yao, Aditya Dhawale, Kshitij Goel, Mosam Dabhi, Matt Collins, Curt Boirum, and Nathan Michael. Fast and agile vision-based flight with teleoperation and collision avoidance on a multirotor. In *Int. Sym. on Exp. Robot. (ISER)*. Springer, 2018. 1.5, 3, 4, 4.3.1.1, 5.4.1, 5.4.2, 6.2, 6.3, 7.4.2, 7.6.2

[20] Aditya Dhawale, Xuning Yang, and Nathan Michael. Reactive collision avoidance using real-time local gaussian mixture model maps. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3545–3550, 2018. 1.5, 4, 4.3.1.2

[21] Xuning Yang and Nathan Michael. Assisted mobile robot teleoperation with intent-aligned trajectories via biased incremental action sampling. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. 1.5, 5

[22] Xuning Yang and Nathan Michael. An intention guided hierarchical trajectory generation framework for trajectory-based teleoperation of mobile robots. *2021 IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2021. 1.5,

6

[23] Thomas B Sheridan and Raja Parasuraman. Human-Automation Interaction. *Reviews of human factors and ergonomics*, 1(1):89–129, 2005. 2.3.1.1

[24] Matthias Nieuwenhuisen, David Droeschel, Johannes Schneider, Dirk Holz, Thomas Läbe, and Sven Behnke. Multimodal obstacle detection and collision avoidance for micro aerial vehicles. In *2013 European Conference on Mobile Robots*. IEEE, 2013. 2.3.1.1

[25] Stefano Stramigioli, Robert Mahony, and Peter Corke. A novel approach to haptic tele-operation of aerial robot vehicles. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.* IEEE, 2010. 2.3.1.1, 5.1

[26] Xiaolei Hou and Robert Mahony. Dynamic kinesthetic boundary for haptic teleoperation of aerial robotic vehicles. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.* IEEE, 2013.

[27] Thanh Mung Lam, Harmen Wigert Boschloo, Max Mulder, and Marinus M Van Paassen. Artificial force field for haptic feedback in uav teleoperation. *IEEE Trans. on Systems, Man, and Cybernetics*, 2009. 2.3.1.1

[28] Marcin Odelga, Paolo Stegagno, and Heinrich H Bülthoff. Obstacle detection, tracking and avoidance for a teleoperated uav. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.* IEEE, 2016. 2.3.1.1, 5.1

[29] S. Gnatzig, F. Schuller, and M. Lienkamp. Human-machine interaction as key technology for driverless driving - A trajectory-based shared autonomy control approach. In *Proceedings - IEEE International Symposium on Robot and Human Interactive Communication*, pages 913–918, 2012. 2.3.1.1

[30] Anca D. Dragan and S. S. Srinivasa. A policy-blending formalism for shared control. *Int. J. Robot. Res.*, 32(7):790–805, 2013. 2.3.1.1, 2.3.1.3

[31] Henny Admoni and Siddhartha Srinivasa. Predicting User Intent Through Eye Gaze for Shared Autonomy. In *The 2016 AIII Fall Symposium Series: Shared Autonomy in Research and Practice. Technical Report FS-16-05*, 2016. 2.3.1.1

[32] Sterling J Anderson, James M Walker, and Karl Iagnemma. Experimental Performance Analysis of a Homotopy-Based Shared Autonomy Framework. In *IEEE Trans. Systems, Man and Cybernetics*, volume 44, pages 190–199, 2014.

[33] Ming Gao, Jan Oberl, Thomas Schamm, and J Z Marius. Contextual Task-Aware Shared Autonomy for Assistive Mobile Robot Teleoperation. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pages 3311–3318, Chicago, IL, USA, September 2014. 2.3.1.1, 2.3.1.3

[34] Siddarth Jain and Brenna Argall. An Approach for Online User Customization of Shared Autonomy for Intelligent Assistive Devices. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, Stockholm, Sweden, May 2016. 2.3.1.1

[35] Aditya Goil, Matthew Derry, and Brenna D Argall. Using Machine Learning to Blend Human and Robot Controls for Assisted Wheelchair Navigation. In *IEEE Int. Conf.*

Rehabil. Robot, Seattle, WA, USA, June 2013.

[36] Katharina Muelling, Arun Venkatraman, Jean-Sebastien Valois, John E Downey, Jeffrey Weiss, Shervin Javdani, Martial Hebert, Andrew B Schwartz, Jennifer L Collinger, and J Andrew Bagnell. Autonomy Infused Teleoperation with Application to BCI Manipulation. In *Proc. of Robot.: Sci. and Syst.*, Ann Arbor, MI, USA, January 2015. 2.3.1.1, 2.3.1.3

[37] Jose Ramon Medina, Tamara Lorenz, and Sandra Hirche. Synthesizing Anticipatory Haptic Assistance Considering Human Behavior Uncertainty. *IEEE Transactions on Robotics*, 31(1):180–190, 2015. 2.3.1.1

[38] Eric Demeester, Alexander Hüntemann, Dirk Vanhooydonck, Gerolf Vanacker, Hendrik Van Brussel, and Marnix Nuttin. User-Adapted Plan Recognition and User-Adapted Shared Control: A Bayesian Approach to Semi-Autonomous Wheelchair Driving. In *Autonomous Robots*, pages 193–211, 2008. 2.3.1.1, 2.3.1.3

[39] 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. 2.3.1.1

[40] Lauren Milliken and Geoffrey A Hollinger. Modeling User Expertise for Choosing Levels of Shared Autonomy. In *Proc. of Robot.: Sci. and Syst. Workshop on Planning for Human-Robot Interaction*, Ann Arbor, MI, USA, June 2016. 2.3.1.1

[41] Tom Carlson and Yiannis Demiris. Human-Wheelchair Collaboration Through Prediction of Intention and Adaptive Assistance. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, pages 3926–3931, Pasadena, CA, 2008. 2.3.1.1, 2.3.1.3

[42] Kris Hauser. Recognition, prediction, and planning for assisted teleoperation of freeform tasks. *Autonomous Robots*, 35(4):241–254, 2013. 2.3.1.1, 2.3.1.3

[43] Eric Demeester, Alexander Hüntemann, Emmanuel Vander Poorten, and Joris De Schutter. ML, MAP and Greedy POMDP Shared Control: Comparison of Wheelchair Navigation Assistance for Switch Interfaces. In *Proc. of the Int. Sym. on Robotics*, pages 1106–1111, Taipei, Taiwan, August 2012. 2.3.1.1

[44] Inkyu Sa and Peter Corke. Vertical infrastructure inspection using a quadcopter and shared autonomy control. 92:219–232, 2014. 2.3.1.1

[45] Oskar Morgenstern and John Von Neumann. *Theory of games and economic behavior*. Princeton university press, 1953. 2.3.1.2, 7.3.3

[46] R Duncan Luce. The choice axiom after twenty years. *Journal of mathematical psychology*, 15(3):215–233, 1977. 2.3.1.2, 7.3.3

[47] Chris L Baker, Joshua B Tenenbaum, and Rebecca R Saxe. Goal inference as inverse planning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 29, 2007. 2.3.1.2, 7.3.3

[48] Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. Learning to navigate through crowded environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 981–986. IEEE, 2010. 2.3.1.2

[49] Mark Pfeiffer, Ulrich Schwesinger, Hannes Sommer, Enric Galceran, and Roland Siegwart. Predicting actions to act predictably: Cooperative partial motion planning with maximum entropy models. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2096–2101. IEEE, 2016.

[50] Dizan Vasquez, Billy Okal, and Kai O Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1341–1346. IEEE, 2014.

[51] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 885–892. IEEE, 2015. 2.3.1.2

[52] Hong Jun Jeon, Smitha Milli, and Anca D. Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning, 2020. 2.3.1.2, 7.3.3

[53] Glenn Wasson, Pradip Sheth, Cunjun Huang, and Alexandre Ledoux. A Physics-Based Model for Predicting User Intent in Shared-Control Pedestrian Mobility Aids. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pages 1914–1919, Sendai, Japan, September 2004. 2.3.1.3

[54] Zhikun Wang, Katharina Mülling, Marc Peter Deisenroth, Heni Ben Amor, David Vogt, Bernhard Schölkopf, and Jan Peters. Probabilistic Movement Modeling for Intention Inference in Human-Robot Interaction. *Int. J. Robot. Res.*, 32(7):841–858, 2013. 2.3.1.3

[55] Dirk Vanhooydonck, Eric Demeester, Alexander Hüntemann, Johan Philips, Gerolf Vanacker, Hendrik Van Brussel, and Marnix Nuttin. Adaptable navigational assistance for intelligent wheelchairs by means of an implicit personalized user model. *Robotics and Autonomous Systems*, 58(8):963–977, 2010. 2.3.1.3

[56] Henrik Kretzschmar, Markus Kuderer, and Wolfram Burgard. Learning to Predict Trajectories of Cooperatively Navigating Agents. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, 2014. 2.3.1.3, 3.2.2.2

[57] Eric Demeester, Emmanuel Vander Poorten, H Alexander, Boris Lau, Markus Kuderer, Andrea Fossati, Gemma Roig, Xavier Boix, Marko Ristin, and Michael Hofmann. Robotic ADaptation to Humans Adapting to Robots: Overview of the FP7 project RADHAR. 2012. 2.3.1.3

[58] Alexander Huntemann, Eric Demeester, Emmanuel Vander Poorten, and Hendrik Van Brussel. Probabilistic approach to recognize local navigation plans by fusing past driving information with a personalized user model. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, pages 4376–4383, Karlsruhe, Germany, 2013. 2.3.1.3

[59] Shervin Javdani, J Andrew Bagnell, and Siddhartha S Srinivasa. Minimizing User Cost for Shared Autonomy. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 621–622, 2016. 2.3.1.3

[60] Matthew Derry and Brenna Argall. A Probabilistic Representation of User Intent for Assistive Robots. In *Int. Conf. on Intell. Robots and Syst. Workshop on Rehabilitation and Assistive Robotics*, Chicago, IL, USA, 2014. 2.3.1.3

[61] Robin Glinton, Sean Owens, Joseph Giampapa, Katia Sycara, Michael Lewis, and Chuck Grindle. Intent Inference Using a Potential Field Model of Environmental Influences Intent Inference Data Flow. In *2005 7th International Conference on Information Fusion*. IEEE, 2005. 2.3.1.3

[62] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807. IEEE, 1993. 2.3.2.1

[63] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research (IJRR)*, 2013. 2.3.2.1

[64] Fei Gao, Yi Lin, and Shaojie Shen. Gradient-based online safe trajectory generation for quadrotor flight in complex environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3681–3688. IEEE, 2017. 2.3.2.1

[65] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5332–5339. IEEE, 2016. 2.3.2.1

[66] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Robot. Res.*, 2010. 2.3.2.1

[67] Jesus Tordesillas, Brett T Lopez, and Jonathan P How. Fastrap: Fast and safe trajectory planner for flights in unknown environments. *arXiv preprint arXiv:1903.03558*, 2019.

[68] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.* IEEE, 2017. 2.3.2.1

[69] Wenchao Ding, Wenliang Gao, Kaixuan Wang, and Shaojie Shen. An efficient b-spline-based kinodynamic replanning framework for quadrotors. *IEEE Transactions on Robotics*, 35(6):1287–1306, 2019. 2.3.2.2

[70] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 2019. 2.3.2.1, 2.3.2.2

[71] Zhijie Zhu, Edward Schmerling, and Marco Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 835–842. IEEE, 2015. 2.3.2.1

[72] Boyu Zhou, Fei Gao, Jie Pan, and Shaojie Shen. Robust real-time uav replanning

using guided gradient-based optimization and topological paths. *arXiv preprint arXiv:1912.12644*, 2019. 2.3.2.1

[73] Marin Kobilarov. Cross-entropy randomized motion planning. *Robotics: Science and Systems VII*, 2012. 2.3.2.1

[74] Mikko Lauri and Risto Ritala. Planning for robotic exploration based on forward simulation. *Robotics and Autonomous Systems*, 2016. 2.3.2.1

[75] Micah Corah and Nathan Michael. Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice. *Autonomous Robots*, 2019. 2.3.2.1

[76] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, 2002. 2.3.2.1

[77] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *J. Field Robot.*, 2009. 2.3.2.1

[78] Mihail Pivtoraiko, Daniel Mellinger, and Vijay Kumar. Incremental micro-uav motion replanning for exploring unknown environments. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.* IEEE, 2013. 2.3.2.1

[79] Wennie Tabib, Micah Corah, Nathan Michael, and Red Whittaker. Computationally efficient information-theoretic exploration of pits and caves. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.* IEEE, 2016. 2.3.2.1

[80] Myung Hwangbo, James Kuffner, and Takeo Kanade. Efficient two-phase 3d motion planning for small fixed-wing uavs. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1035–1041. IEEE, 2007. 2.3.2.1

[81] Matthias Nieuwenhuisen and Sven Behnke. Hierarchical planning with 3d local multiresolution obstacle avoidance for micro aerial vehicles. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–7. VDE, 2014. 2.3.2.1

[82] Amador R Diéguez, Rafael Sanz, and J Lopez. Deliberative on-line local path planning for autonomous mobile robots. *Journal of Intelligent and Robotic Systems*, 37(1):1–19, 2003. 2.3.2.1

[83] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Miller, and Yann LeCun. A multirange architecture for collision-free off-road robot navigation. *Journal of Field Robotics*, 26(1):52–87, 2009.

[84] Dave Ferguson, Thomas M Howard, and Maxim Likhachev. Motion planning in urban environments. In *The DARPA Urban Challenge*, pages 61–89. Springer, 2009.

[85] Anthony Stentz and Martial Hebert. A navigation system for goal acquisition in unknown environments. In *Intelligent Unmanned Ground Vehicles*, pages 277–306. Springer, 1997. 2.3.2.1

[86] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011. 2.3.2.2, 3.2.1.3, 4.3.1.1

[87] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016. 2.3.2.2

[88] Vladyslav Usenko, Lukas von Stumberg, Andrej Pangercic, and Daniel Cremers. Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 215–222. IEEE, 2017. 2.3.2.2

[89] Benjamin J Cohen, Gokul Subramania, Sachin Chitta, and Maxim Likhachev. Planning for manipulation with adaptive motion primitives. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, pages 5478–5485. IEEE, 2011. 3.1

[90] Kris Hauser, Timothy Bretl, Kensuke Harada, and Jean-Claude Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. *Algorithmic foundation of robotics VII*, pages 507–522, 2008.

[91] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pages 2172–2179. IEEE, 2011. 3.1

[92] Jacob W Crandall and Michael A Goodrich. Characterizing Efficiency of Human Robot Interaction: A Case Study of Shared-Control Teleoperation. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pages 1–6, Lausanne, Switzerland, October 2002. 3.1

[93] Erik Arthur Nelson and Nathan Michael. Environment Model Adaptation for Autonomous Exploration. Master's thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2015. 3.2.1.2

[94] Nathan Delson and Harry West. Robot Programming by Human Demonstration: The Use of Human Inconsistency in Improving 3D Robot Trajectories. In *Proc. of the IEEE/RSJ Int. Conf. on Intell. Robots and Syst.*, pages 1248–1255, September 1994. 3.2.1.3

[95] Gerald E. Loeb. Optimal isn't good enough. *Biological Cybernetics*, 106(11):757–765, 2012. 3.2.2.1

[96] Katja Mombaur, Jean-paul Laumond, and Eiichi Yoshida. An Optimal Control-Based Formulation to Determine Natural Locomotor Paths for Humanoid Robots. *Advanced Robotics*, 24:515–535, 2010. 3.2.2.2

[97] Abraham Galton Bachrach. *Trajectory Bundle Estimation For Perception-Driven Planning*. PhD thesis, Dept. Elect. Eng. and Comp. Sci., Massachusetts Institute of Technology, Cambridge, MA, USA, 2013. 3.2.2.2

[98] Sethu Vijayakumar, Aaron D'Souza, and Stefan Schaal. Incremental Online Learning in High Dimensions. *Neural Computation*, 17(12):2602–2634, 2005. 3.2.2.3, 3.3.1

[99] Michael A Goodrich, Erwin R Boer, Jacob W Crandall, Robert W Ricks, and Morgan L Quigley. Behavioral entropy in human-robot interaction. Technical report, Brigham Young University, 2004. 3.3.2

[100] Okihiko Nakayama, Tohru Futami, and Tomokazu Nakamura. SAE Technical Development of a Steering Entropy Method for Evaluating Driver Workload. Technical Report 724, 1999. 3.3.2

[101] Daniel Maier, Armin Hornung, and Maren Bennewitz. Real-time navigation in 3d environments based on depth camera data. In *IEEE-RAS Intl. Conf. on Humanoid Robots*, pages 692–697, Osaka,Japan, November 2012. 4.1

[102] Larry Matthies, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, pages 3242–3249, Hong Kong, China, May 2014. 4.1

[103] Debadeepta Dey, Kumar Shaurya Shankar, Sam Zeng, Rupesh Mehta, M Talha Agcayazi, Christopher Eriksen, Shreyansh Daftry, Martial Hebert, and J Andrew Bagnell. Vision and learning for deliberative monocular cluttered flight. In *Field and Service Robotics*, pages 391–409. Springer, 2016. 4.1

[104] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, pages 1484–1491, Stockholm, Sweden, May 2016.

[105] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, Stockholm, Sweden, May 2016. 4.1

[106] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Proc. of the Int. Workshop on the Algorithmic Foundations of Robot.*, San Francisco, USA, December 2016. 4.1

[107] Peter R Florence, John Carter, Jake Ware, and Russ Tedrake. Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, Brisbane, Australia, May 2018. 4.1

[108] Brett T Lopez and Jonathan P How. Aggressive 3-d collision avoidance for high-speed navigation. In *Proc. of the IEEE Int. Conf. on Robot. and Autom.*, pages 5759–5765, Singapore, May 2017. 4.1

[109] Shobhit Srivastava and Nathan Michael. Approximate Continuous Belief Distributions for Precise Auton. Inspection. In *Proc. of the IEEE Intl. Sym. on Safety, Security and Rescue Robotics*, 2016. 4.1

[110] Maarten AS Boksem, Theo F Meijman, and Monicque M Lorist. Effects of mental fatigue on attention: an erp study. *Cognitive brain research*, 2005. 5.1

[111] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994. 6.2.4.3

[112] Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *European Symposium on Algorithms*, pages 52–63. Springer, 2006. 6.2.4.3

[113] Xuning Yang and Nathan Michael. Assisted mobile robot teleoperation with intent-aligned trajectories via biased incremental action sampling. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. 6.3

[114] Christoph Schöller, Vincent Aravantinos, Florian Lay, and Alois Knoll. What the constant velocity model can teach us about pedestrian motion prediction. *IEEE Robotics and Automation Letters*, 5(2):1696–1703, 2020. 7, 7.2

[115] Paul Robinette, Ayanna M. Howard, and Alan R. Wagner. Effect of robot performance on human–robot trust in time-critical situations. *IEEE Transactions on Human-Machine Systems*, 47(4):425–436, 2017. 7.1

[116] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 7.2

[117] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998. 7.2

[118] Rrt variants and improvements for motion planning. `https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#Variants_and_improvements_for_motion_planning`. Accessed: 2022-01-25. 7.2

[119] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, oct 1979. 7.2

[120] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the national conference on artificial intelligence*, pages 944–951, 1996. 7.2

[121] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Sparse 3d topological graphs for micro-aerial vehicle planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018. 7.2

[122] Fabian Blochliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Rol Siegwart. Topomap: Topological mapping and navigation based on visual slam maps. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3818–3825. IEEE, 2018. 7.2

[123] Fan Yang, Dung-Han Lee, John Keller, and Sebastian A. Scherer. Graph-based topological exploration planning in large-scale 3d environments. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12730–12736, 2021. 7.2, 7.3.1

[124] Shuzhi Sam Ge, Qun Zhang, Aswin Thomas Abraham, and Brice Rebsamen. Simultaneous path planning and topological mapping (sp2atm) for environment exploration and goal oriented navigation. *Robotics and Autonomous Systems*, 2011. 7.2

[125] Subhrajit Bhattacharya. Search-based path planning with homotopy class constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010. 7.2, 1

[126] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Efficient grid-based spatial representations for robot navigation in dynamic environments. *Robotics and Autonomous*

Systems, 61(10):1116–1130, 2013. doi: https://doi.org/10.1016/j.robot.2012.08.010. Selected Papers from the 5th European Conference on Mobile Robots (ECMR 2011). 7.2

[127] Shervin Javdani, Yuxin Chen, Amin Karbasi, Andreas Krause, Drew Bagnell, and Siddhartha Srinivasa. Near optimal bayesian active learning for decision making. In *Artificial Intelligence and Statistics*, pages 430–438. PMLR, 2014. 7.2

[128] Graeme Best and Robert Fitch. Bayesian intention inference for trajectory prediction with an unknown goal destination. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5817–5823. IEEE, 2015. 7.2

[129] Tetsushi Ikeda, Yoshihiro Chigodo, Daniel Rea, Francesco Zanlungo, Masahiro Shiomi, and Takayuki Kanda. Modeling and prediction of pedestrian behavior based on the sub-goal concept. *Robotics: Science and Systems*, 2013. 7.2

[130] Sujeong Kim, Stephen J Guy, Wenxi Liu, David Wilkie, Rynson WH Lau, Ming C Lin, and Dinesh Manocha. Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *The International Journal of Robotics Research*, 34(2):201–217, 2015. 7.2

[131] David J Bruemmer, Donald D Dudenhoeffer, and Julie L Marble. Dynamic-autonomy for urban search and rescue. In *AAAI Technical Report WS-02-18*, 2002. 7.2

[132] Helen S Loeb, Elizabeth Vo-Phamhi, Thomas Seacrist, Jalaj Maheshwari, and Christopher Yang. Vehicle automation emergency scenario: Using a driving simulator to assess the impact of hand and foot placement on reaction time. *SAE Technical Paper*, pages 01–0861, 2021. 7.2

[133] Holly EB Russell, Lene K Harbott, Ilana Nisky, Selina Pan, Allison M Okamura, and J Christian Gerdes. Motor learning affects car-to-driver handover in automated vehicles. *Science Robotics*, 1(1):eaah5682, 2016. 7.2

[134] Austin R Benson, Ravi Kumar, and Andrew Tomkins. On the relevance of irrelevant alternatives. In *Proceedings of the 25th International Conference on World Wide Web*, pages 963–973, 2016. 7.3.2