2023

Motion Planning in Artificial and Natural Vector Fields

Bernardo Martinez Rocamora Junior
West Virginia University

MOTION PLANNING IN ARTIFICIAL AND NATURAL VECTOR FIELD

Bernardo Martinez Rocamora Jr.

Dissertation submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Aerospace Engineering

Guilherme Pereira, Ph.D., Committee Chairperson
Adam Halasz, Ph.D.
Jason Gross, Ph.D.
Xi Yu, Ph.D.
Yu Gu, Ph.D.

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2023

Keywords: Robotics, Motion Planning, Obstacle Avoidance, Vector Fields, Strong Currents, Strong Winds, Venus Aerobots

ABSTRACT

Motion Planning in Artificial and Natural Vector Fields

by Bernardo Martinez Rocamora Jr.

This dissertation advances the field of autonomous vehicle motion planning in various challenging environments, ranging from flows and planetary atmospheres to cluttered real-world scenarios. By addressing the challenge of navigating environmental flows, this work introduces the Flow-Aware Fast Marching Tree algorithm (FlowFMT*). This algorithm optimizes motion planning for unmanned vehicles, such as UAVs and AUVs, navigating in tridimensional static flows. By considering reachability constraints caused by vehicle and flow dynamics, flow-aware neighborhood sets are found and used to reduce the number of calls to the cost function. The method computes feasible and optimal trajectories from start to goal in challenging environments that may contain obstacles or prohibited regions (e.g., no-fly zones). The method is extended to generate a vector field-based policy that optimally guides the vehicle to a given goal. Numerical comparisons with state-of-the-art control solvers demonstrate the method's simplicity and accuracy. In this dissertation, the proposed sampling-based approach is used to compute trajectories for an autonomous semi-buoyant solar-powered airship in the challenging Venusian atmosphere, which is characterized by super-rotation winds. A cost function that incorporates the energetic balance of the airship is proposed to find energy-efficient trajectories. This cost function combines the main forces acting on the vehicle: weight, buoyancy, aerodynamic lift and drag, and thrust. The FlowFMT* method is also extended to consider the possibility of battery depletion due to thrust or battery charging due to solar energy and tested in this Venus atmosphere scenario. Simulations showcase how the airship selects high-altitude paths to minimize energy consumption and maximize battery recharge. They also show the airship sinking down and drifting with the wind at the altitudes where it is fully buoyant. For terrestrial applications, this dissertation finally introduces the Sensor-Space Lattice (SSLAT) motion planner, a real-time obstacle avoidance algorithm for autonomous vehicles and mobile robots equipped with planar range finders. This planner uses a lattice to tessellate the area covered by the sensor and to rapidly compute collision-free paths in the robot surroundings by optimizing a cost function. The cost function guides the vehicle to follow an artificial vector field that encodes the desired vehicle path. This planner is evaluated in challenging, cluttered static environments, such as warehouses and forests, and in the presence of moving obstacles, both in simulations and real experiments. Our results show that our algorithm performs collision checking and path planning faster than baseline methods. Since the method can have sequential or parallel implementations, we also compare the two versions of SSLAT and show that the run-time for its parallel implementation, which is independent of the number and shape of the obstacles found in the environment, provides a significant speedup due to the independent collision checks.

This work is dedicated to my Grandma Eloina, who passed during the preparation of this document. My Grandma was a wise woman; she knew the transformative value of education.

Este trabalho é dedicado à minha Avó Eloina, que faleceu durante a preparação desse documento. Minha avó era uma mulher sábia e que conhecia o valor transformativo da educação.

Acknowledgments

I would like to express my gratitude to the individuals and organizations participating in my academic journey and personal life. Without their support, guidance, and encouragement, I would not have been able to achieve my goals and aspirations.

First and foremost, I would like to express my sincere appreciation to Dr. Guilherme Pereira, my advisor, for granting me the privilege to embark on a doctoral journey under his mentorship. I am deeply thankful for your invaluable guidance and the time you've dedicated to supporting me. I would also like to thank the members of my committee, Dr. Jason Gross, Dr. Yu Gu, Dr. Xi Yu, and Dr. Adam Halasz. Your feedback, expertise, and constructive criticism have enriched my research and contributed to its quality.

I am immensely grateful to the Office of the Provost at WVU, in the person of the Vice Provost Dr. Paul Kreider, who funded the first year of my doctoral studies through an initiative to recruit students from Latin America. I am also immensely grateful to Ben and Jo Statler and the Statler College of Engineering and Mineral Resources for the Statler Fellowship, which financially supported me throughout my journey at WVU. I also thank the Alpha Foundation for the Improvement of Mine Safety and Health, Amazon Research Awards, and the National Aeronautics and Space Administration (NASA) for partially funding and providing motivation for my work.

I would like to express my love for my parents, Bernardo and Dora, and my brother, Henrique, who have been pillars of strength throughout my academic journey despite our

significant physical distance. Your unwavering love, encouragement, and belief in me have been my driving force. I am forever grateful for your support.

I would also like to express my love and appreciation to my partner, Rachel. I am immensely proud of her for her hard work and undefeatable spirit. Her determination serves as a reminder that with perseverance, anything is achievable. Thank you for being both my partner and my source of motivation. A special mention to our gato, Max, who has brought joy, dead mice, and companionship into our lives.

To all my Morgantown friends and colleagues from work, Rogerio Lima, Christopher Tatsch, Çagri Kilic, Iuri Lira, Felipe Sozinho, André Fernandes, Anthony Rames, André Viana, Patrícia Viana, Vinicius Ferreira, Dimas Dutra, Jonas Bredu, Shounak Das, Uthman Olawoye, Matteo Petrillo, Derek Ross, Kieren Samarakoon, Paulo Simplicio, Anna Puigvert, Maria Gonzalez, Jared Beard, Jared Strader, Nick Ohi, Jeremy Rathjen, Lucas Hehl, Luis Escobar, who have supported me along the way, thank you.

Agradecimentos

Gostaria de expressar a minha gratidão às pessoas e organizações que participaram na minha jornada acadêmica e vida pessoal. Sem o seu apoio, orientação e encorajamento, não teria sido capaz de alcançar os meus objetivos e aspirações.

Primeiramente, gostaria de expressar minha sincera gratidão ao Dr. Guilherme Pereira, meu orientador, por me conceder o privilégio de embarcar em uma jornada de doutoramento sob sua mentoria. Sou profundamente grato pela sua inestimável orientação e por todo o tempo que você dedicou em me apoiar. Gostaria também de agradecer aos membros do meu comitê, Dr. Jason Gross, Dr. Yu Gu, Dr. Xi Yu e Dr. Adam Halasz. Seus comentários, expertise e críticas construtivas enriqueceram a minha pesquisa e contribuíram para a sua qualidade.

Sou imensamente grato ao Gabinete do Reitor da WVU, na pessoa do Vice-Reitor Dr. Paul Kreider, que, através de uma iniciativa para recrutar estudantes da América Latina, financiou o primeiro ano de meus estudos de doutorado. Também estou imensamente grato a Ben e Jo Statler e à Statler College of Engineering and Mineral Resources pela Statler Fellowship, que me apoiou financeiramente durante a maior parte da minha jornada na WVU. Agradeço também à Alpha Foundation for the Improvement of Mine Safety and Health, à Amazon Research Awards e à National Aeronautics and Space Administration (NASA) por parcialmente financiarem e por fornecerem motivação para meu trabalho.

Gostaria de expressar o meu amor pelos meus pais, Bernardo e Dora, e pelo meu irmão, Henrique, que foram os pilares que me surportaram ao longo da minha jornada acadêmica, apesar da grande distância física entre nós. O seu amor inabalável, encorajamento e fé em mim têm sido a minha força motriz. Estou eternamente grato pelo seu apoio.

Também gostaria de expressar o meu amor e apreço à minha companheira, Rachel. Sou imensamente orgulhoso do seu trabalho árduo e espírito inabalável. A sua determinação serve como um lembrete de que com perseverança, tudo é possível. Obrigado por ser minha companheira e também por ser uma fonte de motivação. Agradeço também ao nosso gato, Max, que trouxe alegria e companheirismo para as nossas vidas.

A todos os meus amigos de Morgantown e colegas de trabalho, Rogerio Lima, Christopher Tatsch, Çagri Kilic, Iuri Lira, Felipe Sozinho, André Fernandes, Anthony Rames, André Viana, Patrícia Viana, Vinicius Ferreira, Dimas Dutra, Jonas Bredu, Shounak Das, Uthman Olawoye, Matteo Petrillo, Derek Ross, Kieren Samarakoon, Paulo Simplicio, Anna Puigvert, Maria Gonzalez, Jared Beard, Jared Strader, Nick Ohi, Jeremy Rathjen, Lucas Hehl, Luis Escobar, que me apoiaram ao longo do caminho, obrigado.

"Just do it. Take all the courses in your curriculum. Do the research. Ask questions. Find someone doing what you are interested in! Be curious!"

— Katherine Johnson

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

WVU	West Virginia University
FARO	Field and Aerial Robotics Laboratory
UUV/AUV	Unmanned/Autonomous Underwater Vehicle
UAV	Unmanned Aerial Vehicle
IEEE	Institute of Electrical and Electronics Engineers
ICUAS	International Conference on Unmanned Aircraft Systems
ICRA	International Conference on Robotics and Automation
RAL	Robotics and Automation Letters
TAES	Transactions on Aerospace and Electronic Systems
NASA	National Administration Space Agency
2D/3D	Two/Three Dimensional
RRT	Rapidly-Exploring Random Tree
RRT*	Rapidly-Exploring Random Tree Star
EWRRT	Energy- and Wind-Aware Rapidly-Exploring Random Tree
FMT*	Fast Marching Tree Star
FlowFMT*	Flow-Aware Fast Marching Tree Star
OCS	Optimal Control Solver
ICLOCS	Imperial College London Optimal Control Software

VCD	Venus Climate Database
GCM	General Circulation Model
LMD	Laboratoire de Meteorologie Dynamique
LATMOS	Laboratoire Atmospheres, Observations Spatiales
IPSL	Institute Pierre Simon Laplace
S-N	North-South direction
W-E	West-East direction
SW	Short Wave
SSLAT	Sensor-Space Lattice Motion Planner
PRM	Probabilistic Roadmap
LIDAR	Light Detection and Ranging
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
APF	Artificial Potential Function
VFF	Virtual Force Field
VFH	Vector Field Histogram
DWA	Dynamic Window Approach
EBand	Elastic Bands
CDC	Collision Detection Circuit
A*	A Star
GMT	Group Marching Tree
CTG	Cost-to-go
CPU	Central Processing Unit
SLAM	Simultaneous Localization and Mapping
BARN	Benchmark for Autonomous Robot Navigation
ROS	Robot Operating System
GNSS	Global Navigation Satellite System

GPS	Global Positioning System
IMU	Inertial Measurement Unit

List of Symbols

x, y, z	m	Cartesian Coordinates
x_g, y_g, z_g	m	Cartesian Coordinates in Global Frame
x_b, y_b, z_b	m	Cartesian Coordinates in Body Frame
c_x, c_y, c_z	$\mathrm{m\,s^{-1}}$	Flow Components in Global Frame
w_x, w_y, w_z	$\mathrm{m\,s^{-1}}$	Wind Components in Global Frame
$\vec{p}$	m	Vehicle's Position Vector
$\vec{v}$	$\mathrm{m\,s^{-1}}$	Vehicle's Velocity Vector
$\vec{c}$	$\mathrm{m\,s^{-1}}$	Flow Velocity Vector
P	W	Power
σ	N	Trajectory
W	N	Weight Force
B	N	Buoyancy Force
L	N	Lift Force
D	N	Drag Force
T	N	Thrust Force
γ	rad	Flight Path Angle
ψ	rad	Yaw Angle
ϕ	rad	Roll Angle

α	rad	Angle of Attack
$\dot{\psi}$	$\mathrm{rad\,s^{-1}}$	Yaw Rate
m	kg	Mass
ρ	$\mathrm{kg\,m^{-3}}$	Density
V	$\mathrm{m^3}$	Volume
g	$\mathrm{m\,s^{-2}}$	Acceleration of Gravity
S	$\mathrm{m^2}$	Wing Area
v_a	$\mathrm{m\,s^{-1}}$	Airspeed
c_L	-	Lift Coefficient
c_D	-	Drag Coefficient
$c_{D,0}$	-	Zero-Lift Drag Coefficient
e	-	Oswald Efficiency Number
$\mathit{Æ}$	-	Wing Aspect Ratio
b	J	Battery Charge
E	J	Energy
P	W	Power
η	-	Efficiency
h	m	Altitude
v	-	Vertex
e	-	Edge
$\mathcal{W}$	-	Workspace
$\mathcal{O}$	-	Obstacles
$\mathcal{X}$	-	State Space
X	-	Position Space
$\dot{X}$	-	Velocity Space
$\mathcal{C}$	-	Configuration Space
$\mathcal{S}$	-	Sensor Space

$\mathbf{F}_c$	-	Current Vector Field
C	-	Cost Function
$\mathbb{R}$	-	Real Numbers
$\mathbb{S}$	-	Circle Group
$\mathcal{G}$	-	Graph
$\mathcal{T}$	-	Tree Graph
V	-	Vertices Set
E	-	Edges Set
$\sqsupseteq$	-	World Frame
r_0	-	Lattice inner radius
N_T	-	Number of trunks
N_B	-	Number of branches
N_L	-	Number of layers
K	-	Growth ratio
r, θ	-	Cylindrical coordinates
t	-	Trunk ordinal number
b	-	Branch ordinal number
l	-	Layer ordinal number
$\cdot$	-	Set of tesselating triangles
$\mathcal{L}$	-	Set of laser beams
$\mathcal{R}_{\cdot}^{\mathcal{E}}$	-	Triangle-to-Edge Map
$\mathcal{R}_{\mathcal{L}}^{\cdot}$	-	Laser-Beam-to-Triangle Map
$\mathbf{F}_v$	-	Guiding Vector Field
$\mathbf{N}$	-	Normal Vector Component
$\mathbf{T}$	-	Tangential Vector Component
τ	-	Triangle
l_m	-	Laser Beam

m	-	Laser Beam cardinal number
r_R	-	Robot radius
r_{tree}	-	Tree radius
d_L^R	-	Sensor to robot frame offset
C_m	-	Circular region around laser measurement
$\mathcal{F}$	-	Vector field alignment cost functional
ξ_i^j	-	Edge from vertex i to vertex j
$\xi_i'^j$	-	First derivative of edge from vertex i to vertex j
s	-	Spatial parameterization
$N_{threads}$	-	number of beams in threads
N_{blocks}	-	number of triangles in blocks
ρ	-	Density
c	-	Convergence parameter
ϕ	-	Potential field
v_x, v_y, v_z	-	Commanded velocities
ω_z	-	Commanded yaw angular rate
x_{sp}, y_{sp}	-	Setpoint Coordinates
ψ_{sp}	-	Setpoint Yaw
K_ψ	-	Proportional Constant for Yaw

Chapter 1

Introduction

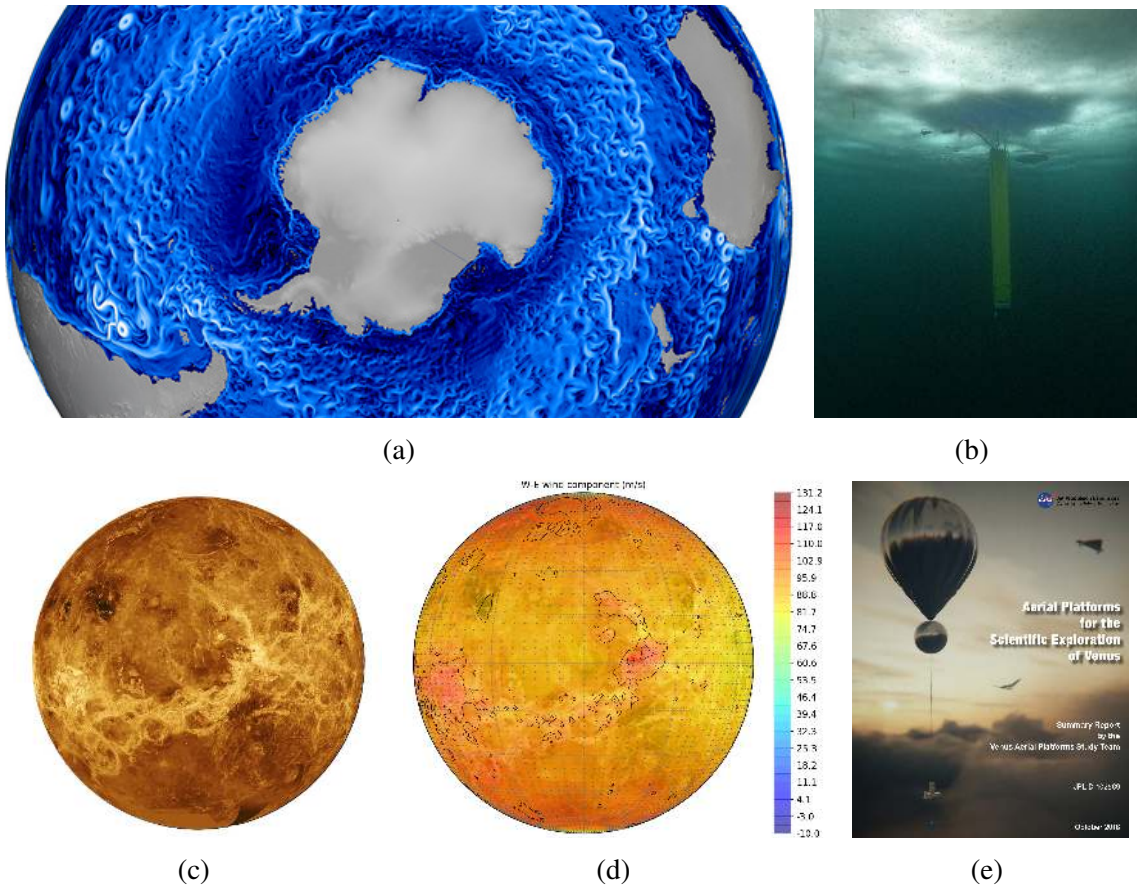
It is not the case anymore that robots are confined to structured and controlled environments. Robots are taming the wild [1]. Their wider use and proliferation were impulsed by recent technological advancements – cheaper, smaller, and more powerful electronics and ubiquitous and seamless connectivity – and by the ripening of Robotics as a scientific field along with the popularization of Artificial Intelligence techniques. With applications that range from responding to accidents in nuclear power plants [2] to investigating the effects of climate change on the Antarctic ice shelves [3, 4], robots enable the exploration of remote or dangerous places. They do not need costly resources such as life support and food, and they can work continuously, often making them more cost-effective than deploying humans, especially for long-term missions. They can take measurements, collect data, and capture images or video footage in environments where it's difficult or unsafe for humans to operate. Being able to perform these repetitive tasks precisely makes them ideal for scientific experiments and sample collection in these hostile locations.

1.1 Motivation

This dissertation focuses on Robot Motion Planning, considered one of the foundation areas for the field of Robotics. Motion planning is the science that studies how robots should

move in the world to accomplish their tasks safely and efficiently. The recent decades have witnessed formidable progress as robots are deployed in diverse, complex, and challenging environments. The primary motivation of this dissertation is to solve some problems encountered by a robot facing the daunting forces of nature. Wind and ocean currents, examples of such forces, can exert external forces on a robot and push it off its intended route, making it difficult to achieve precise and predictable motion. Figure 1.1 illustrates two large-scale examples of environments with natural flows that robots can explore. While the natural currents can help the robot reach its goal faster when moving along with the flow, navigating against them requires extra energy. Robots may need additional power to maintain their desired trajectory, potentially reducing their operational efficiency and lifespan. Strong currents can, in some cases, cause restrictions on which places are accessible. Their direction and strength can change significantly across time and space, and the robots may need to replan their paths to adapt to the new conditions. This constant replanning can be computationally intensive and may slow the robot's progress. This work attempts to optimize the robot motion accounting for some of these elements so that the robots follow short and efficient paths, and replanning is not needed as frequently.

Another essential aspect of this dissertation is equipping the robot to handle unexpected events after a motion plan is traced. For example, consider a robot being deployed in a partially mapped environment. An initial plan for the robot mission is computed, and the robot starts moving. At some point in the motion execution, the robot detects an obstacle that contradicts the previously computed plan. Now, the robot faces an additional challenge of avoiding that obstacle that was not accounted for. Once the robot's perception systems detect the obstacle, it must update its understanding of the world and swiftly react to its presence. In the short term, the robot needs to adapt its plan to avoid running into a collision. Obstacle avoidance techniques compute changes to a robot's motion control to guarantee that the robot is still moving toward the target location but through a sequence of motions that are safe to execute. This can involve altering the robot's speed, steering, or



(a) (b) (c) (d) (e)

Figure 1.1: Be the Earth, Venus, or other remote and inaccessible locations, robots will be continuously deployed to work in various environments where strong flow fields can be present. (a) Antarctic ocean currents and eddies in a high-resolution global ocean simulation. Credit: Los Alamos National Laboratory CC BY-NC-ND 2.0 DEED. (b) Underwater view of IceNode [4] landed on the underside of the ice at Lake Superior, Michigan. Credit: NASA IceNode JPL Core Team. (c) Mosaics showing a global view of the surface of Venus from Magellan synthetic aperture radar image, centered at 180° East longitude. Credit: NASA Goddard Space Flight Center. (d) Predictions of zonal winds in the atmosphere of Venus at 65,000km of altitude on Oct. 29, 2023, centered at 180° East longitude. Generated using Venus Climate Database © LMD/ESA [5, 6, 7]. (e) Concept art of robotic balloons that could be part of a Venus exploration. Credit: NASA/JPL-CalTech/Tibor Balint. NASA images follow this policy.

path in real time to navigate around obstacles safely.

Figure 1.2 illustrates robots moving in environments and detecting obstacles that prevent them from achieving their goals as planned. As shown in Figure 1.2b, in a depiction of our first practical attempt at designing an obstacle avoidance planner for a mobile robot guided by an artificial vector field, the paths in the local space around the robot can be used

(a) (b)

Figure 1.2: Obstacle avoidance capabilities become necessary when a robot explores unknown places. (a) A robot navigating ocean currents can encounter unexpected obstacles that may require deviating from the previously computed optimal path. (b) Our first practical attempt at designing an obstacle avoidance planner for a mobile robot guided by an artificial vector field.

to assess alternatives to the commands. The definition of this vector field is what connects the motion planners presented in this dissertation.

1.2 Research Objectives

The primary goal of this dissertation is to explore and advance state-of-the-art knowledge in motion planning for autonomous vehicles working in complex environments. We specifically focus on autonomous vehicles navigating strong flows or artificial vector fields that encode a desired behavior. The proposed research can be divided into two aims:

- **Aim 1:** To design optimal motion planners for robots navigating in an environment with strong currents, i.e., the flow velocities can be larger than the vehicle's velocity relative to the flow, causing problems related to position reachability and path infeasibility. Many examples of navigation problems can benefit from motion planning algorithms tailored to optimize generic cost functions while accounting for strong flows, including the navigation of exploratory aerobots in the atmosphere of Venus, which is one of the problems solved by our work.

- **Aim 2:** To develop a real-time obstacle avoidance and path planning strategy that allows fast robots to traverse cluttered environments while following an artificial vector field that is oblivious to the presence of obstacles but encodes the high-level task of the robot. Similar to the problem solved by our first aim, several applications would benefit from such a planner, including forest navigation, where the position of the obstacles (trees) cannot be determined before the execution of the task.

1.3 Contributions

The contents of this dissertation resulted in publications in journals and conference proceedings. A list of these publications is provided below:

- **Contribution 1:** The development of a flow-aware sampling-based motion planning strategy, named FlowFMT*, that yields minimum-time or minimum-energy paths for holonomic vehicles in a three-dimensional environment with obstacles and accounts for the limitations in reachability given by the ratio between the flow velocity and the maximum vehicle velocity relative to the flow. Neighborhood set functions are defined using reachability cones to reduce the number of calls to the cost functions. A vector-field policy to encode a feedback motion plan for the vehicle actuation is extracted from the graph generated using the sampling-based solution and the local flow velocity. A paper with the results of this chapter will be submitted to the Autonomous Robots journal shortly:

  - **Bernardo Martinez Rocamora Jr.**, and Guilherme A. S. Pereira. "Optimal Policies for Autonomous Navigation in Strong Flows Using Fast Marching Trees."

- **Contribution 2:** The development of an energy-efficient sampling-based motion planning strategy, named EWRRT, that i) uses wind-deformed Dubins' Airplane tra-

jectories as local planners and ii) includes a semi-buoyant dynamic model for the aircraft; and the development of a cost function that i) accounts for the energy expenditure of the propulsive system, ii) accounts for the vehicle's battery state, and iii) considers battery charging as an opportunity cost, thus avoiding negative costs in the function. This contribution was partially published at the 2022 and the 2023 International Conferences on Unmanned Aircraft Systems (ICUAS) as:

  - ©2022 by IEEE. Reprinted, with permission, from **Bernardo Martinez Rocamora Jr.**, Anna Puigvert i Juan, and Guilherme A. S. Pereira. "Towards Finding Energy Efficient Paths for Hybrid Airships in the Atmosphere of Venus." In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022.
  - Anna Puigvert i Juan, **Bernardo Martinez Rocamora Jr.**, Guilherme A. S. Pereira. "Wind-Aware Path Optimization for an Aerobot in the Atmosphere of Venus Using Genetic Algorithms" In *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 249-256. IEEE, 2023.

- **Contribution 3:** The development of a flow-aware sampling-based motion planning strategy for holonomic vehicles based on Contributions 1 and 2. The algorithm handles regions where the vehicle cannot actuate because of a low battery state. Another paper with a complete solution for the aerobot navigation problem in the Venusian atmosphere will be soon submitted to IEEE Transactions on Aerospace and Electronic Systems (TAES) as:

  - **Bernardo Martinez Rocamora Jr.**, and Guilherme A. S. Pereira. "Circumnavigating Venus with Aerobots: Motion Planning from Dusk to Dawn under Strong Winds."

- **Contribution 4:** The development of a lattice-based local motion planning strategy, named SSLAT, that optimizes a local, vector-field-dependent functional for obstacle

avoidance and field tracking. The strategy includes the development of a method for generating a lattice in the sensor space of the robot that simultaneously tessellates the sensor space and stores in a tree graph the possible paths a robot can take locally and the development of an embarrassingly parallel strategy based on a precomputed mapping between sensor measurements and lattice edges for collision detection in parallel architectures, including CUDA. An initial version of this contribution was published at the 2021 IEEE International Conference on Robotics and Automation (ICRA), and the final version was published in the MDPI Sensors journal.

  -

  -

Additionally, the author contributed to other research projects not included in this dissertation but fundamental to his overall scientific development. The list of publications is provided below.

- **Stone Mine Project:** This project aimed at developing a cooperative robotic system for autonomous stone mine pillar inspection. The system comprised two major components: a rover (ground robot) and a drone (aerial robot). The drone was connected to the rover through a tether system that delivers power from batteries located in the rover, and, though sensing of tether variables such as angles at both endpoints (drone

and rover), tension, and length, aids in localization and landing of the drone. The rover also provides a self-leveling landing platform on the rover side so the drone can take off and land on a leveled surface, even if the rover is on a slope. The author contributed to the system's hardware design and developing software for the autonomous operation of the aerial vehicle. The drone's hardware and software architecture were described in the MDPI Drones journal. A method for landing was published in the IEEE Robotics and Automation Letters (RA-L).

  - **Bernardo Martinez Rocamora Jr.**[1], Rogério R. Lima[1], Kieren Samarakoon, Jeremy Rathjen, Jason N. Gross, and Guilherme A. S. Pereira. "Oxpecker: A Tethered UAV for Inspection of Stone-Mine Pillars." *Drones* 7, no. 2 (2023): 73.
  - Rogério R. Lima, **Bernardo Martinez Rocamora Jr.** and Guilherme A. S. Pereira, "Continuous Vector Fields for Precise Cable-Guided Landing of Tethered UAVs," *IEEE Robotics and Automation Letters* (2023).

- **Space Robotics Challenge Phase 2:** The SRCP2 was a two-year NASA Centennial Challenge in which teams competed to solve challenging problems posed by NASA to advance autonomous robotic operations for space exploration missions on the surface of other celestial bodies, such as the Moon and Mars. The challenge consisted of developing technologies to enable a team of six lunar robots to operate entirely autonomously for two hours, e.g., without human intervention, to locate and excavate resources and return them to a home base in a virtual lunar environment. The author led software development for many areas, such as System Integration, Autonomy, Manipulation, and Driving. The team solution and results for the qualifying round were published in the IEEE Aerospace and Electronic Systems Magazine, and the final round in Frontiers in Robotics and AI journal.

– Cagri Kilic[1], **Bernardo Martinez Rocamora Jr.**[1], Christopher A. A. Tatsch[1], Jared Beard, Jared Strader, Shounak Das, Derek Ross, Yu Gu, Guilherme A. S. Pereira, and Jason N. Gross. "NASA Space Robotics Challenge 2 Qualification Round: An Approach to Autonomous Lunar Rover Operations." *IEEE Aerospace and Electronic Systems Magazine* 36, no. 12 (2021): 24-41.

– **Bernardo Martinez Rocamora Jr.**[1], Cagri Kilic[1], Christopher A. A. Tatsch[1], Guilherme A. S. Pereira, Jason N. Gross "Multi-robot cooperation for lunar In-Situ resource utilization." *Frontiers in Robotics and AI* 10 (2023): 1149080

1.4 Dissertation Outline

The remainder of this dissertation is divided as follows:

- **Chapter 2** presents a literature review covering scientific publications on motion planning in natural flows and motion planning using artificial vector fields.
- **Chapter 3** proposes a methodology for motion planning for holonomic vehicles moving in a flow field where the flow speed can be greater than the vehicle's speed relative to the flow.
- **Chapter 4** proposes a methodology for motion planning of a low-speed semi-buoyant aerobot navigating the atmosphere of Venus, where superrotation wind speeds can be much larger than the aerobot's maximum airspeed.
- **Chapter 5** proposes a novel methodology for real-time motion planning in cluttered environments that allows following artificial vectors and avoiding unaccounted obstacles when selecting the target vector fields.
- **Chapter 6** shows conclusions for the work presented in this dissertation and discusses possibilities for new research directions following up on this work.

[1]These authors contributed equally to this paper.

- **Appendices A, B, and C** contain extensions to the methods developed in Chapters 3 and 4 with some preliminary results. These extensions have been partially developed and still need to be investigated further.

Chapter 2

Literature Review

In Robotics, motion planning algorithms are the key to converting high-level specifications of tasks defined by humans into low-level descriptions of how to move [8]. The most fundamental example is planning a path free of collision from a given starting position to a goal position. This problem, which looks simple, has been the center of attention of many roboticists for decades. This dissertation is interested in using vector fields to create models or support solving the robot motion planning problem. A vector field is a function that assigns a vector to each point in space. Vectors, composed of direction and magnitude, can be used for various applications, including to represent motion. For example, they can encode a velocity at each point in space, resulting in a desired behavior that a human controller wants the robots to execute, like following a path or avoiding obstacles. Alternatively, they can describe essential characteristics of the environment at each point. For example, they can describe how a fluid moves in three-dimensional space.

2.1 Overview

In this dissertation, vector fields are used in two different ways. We call the first type of vector field "natural" since they emerge from the nature of the environment in which the robots navigate, and we call the second type "artificial", given that they are selected or

generated by humans. This chapter surveys the earlier and state-of-the-art developments in the field. It is divided into two sections for the two general categories of vector field usage. The following chapters review the literature with a more focused view of the specific topics they cover.

2.2 Motion Planning in Flowing Environments

Some of the most important categories of autonomous robots can be significantly impacted by environmental flows: Unmanned Aerial Vehicles (UAVs) and Autonomous Underwater Vehicles (AUVs). In this study, "natural vector fields" serve as representations of these environmental flows. For instance, these vector fields associate flow speed and direction with each point in space, describing the movement of fluid where robots navigate. UAVs are agile and fast, and, in general, they can treat wind as a disturbance that alters the stability and control of the vehicle. While winds make flying more complicated, depending on their strength, they can be counteracted as the UAV flies and performs its tasks. However, for a subset of UAVs that are naturally slow or that work in an environment with strong currents, such as underactuated or lighter-than-air UAVs, and also for AUVS in oceanic currents, the environmental flow can bring other issues like infeasibility of the desired path. Dedicated motion planning algorithms can help generate paths that consider the environmental currents, helping to ensure that the vehicles reach their destinations safely and efficiently.

In 1931, Ernst Zermelo [9] proposed a problem (translated to English here) that still attracts the attention of many scientists today:

> In an unbounded plane, where the wind distribution is given by a vector field as a function of place and time, a vehicle moves with constant airspeed relative to the surrounding air mass. How does this vehicle have to be controlled in order to get from a starting point to a given destination in the shortest possible time?

The problem has been expanded to account for the vehicle's dynamics, more generalized flow conditions, and various optimization metrics. Many approaches were taken to work with this problem: optimal control theory, probabilistic methods, evolutionary methods, grid-based, sampling-based methods, and, more recently, reinforcement learning methods. In one of the first path planner papers for AUV, a grid-based search (A*) path planning method was presented by [10] for variable-speed AUVs in ocean environments considering bathymetry, exclusion zones, obstacles, and ocean currents. The three-dimensional environment is represented as quadtrees, and it is considered that the robot can reach the entire workspace. An evolutionary approach is proposed in [11] to solve the path planning problem for a UAV in traversing long distances in wind fields. The energy cost calculations consider the wind, and paths aligned with the wind result in lower energy consumption.

Another method using a genetic algorithm [12] for minimum energy cost path planning was developed for underwater vehicles in two and three-dimensional environments for time and space-varying ocean currents. While the method considers complex ocean currents, it does not restrict the vehicle's maximum speed. It considers that the vehicle can counteract the ocean currents to reach a nominal speed. In [13], a grid-based search (A*) method is applied for finding minimum energy paths for AUVs in 2D ocean environments. The velocities of the flow were considered to be always smaller than the velocity of the vehicle relative to the current. When the speed of the vehicle relative to the fluid is smaller than the speed of the fluid relative to the inertial frame we say that the vehicle is subjected to strong winds or currents. To handle incorrectness and incompleteness issues caused by the presence of strong currents, a sliding wavefront expansion method was proposed in [14]. More recently, other path planning methods based on A* for such 2D environments were introduced in [15, 16]. Minimum time and minimum energy cost functions were developed in [16] using flow-oriented coordinates, which considered the feasibility problem implicitly. Most of the existing research for AUVs has considered 2D or quasi-two-dimensional

environments, but in [17], a level-set solution based on exact differential equations was proposed for time-varying and 3D scenarios.

A method for optimal time paths for aircraft flying in a constant wind environment with bounded turning was developed in [18] by expressing the goal as a moving target and solving a standard iterative root-finding method to find the intersection time. The aircraft's relative velocity was lower than the wind speeds, and the analysis was restricted to 2D. In [19], a path planner for a micro-aerial vehicle (MAV) was proposed considering constant wind field. They formulate the problem as an asymmetric traveling salesman problem and solve it using a mixed-integer linear program. However, no considerations are provided regarding the issue that strong winds can cause. A real-time environment-aware planner has been proposed in [20] using variations of RRT* [21] that approximate the dynamics of the aircraft using Dubins' Airplane [22], and incorporating wind field data in its heuristics.

A minimum energy planner was proposed in [23] for a UAV subject to wind caused by urban canyons. The authors used an A* grid search to find a minimum cost path, and their proposed cost function is based on the change of total energy, composed of potential, kinetic, and stored energy (battery). However, their model assumed constant altitude, constant airspeed, and no recharging. The total energy decreases monotonically with the consumption of the stored energy. Also, their method requires wind speeds to be smaller than the airspeed to converge. Path planners for a gliding aircraft under complex wind fields were studied in [24, 25]. These planners grow a tree of feasible trajectories from a discrete set of allowable inputs and weigh the branches using a cost function that accounts for changes in total energy and distance to the goal. In a recently published paper [26], a time-optimal path planning method was developed for UAVs moving in strong uniform winds with a constrained turning rate for the aircraft. The paper combines maximum-rate turn segments (also known as trochoids) and straight segments under the effect of wind. The method iterates over possible trajectory types to reduce computational load trying to find the optimal connection between two states.

The methods employed for motion planning in natural vector fields, such as wind and ocean currents, share certain characteristics found in artificial vector fields since both can affect the movement of a robot. However, they differ in their essence: natural vector fields are immutable from the vehicle standpoint. They are a disturbance that can be helpful or problematic. So the robot needs to adapt and overcome the constraints imposed by them. Artificial vector fields, on the other hand, are defined by the user or the robot, they are there to help the vehicle meet its goals. In the next section we discuss what were the previous developments in artificial vector fields.

2.3 Implicit Representation of Motion Plans Using Artificial Fields

In the 1930s, the psychologist Clark Hull, studying the behavior of animals when learning a maze, proposed what was called the "goal gradient hypothesis" [27], which states that "animals in traversing a maze will move at a progressively more rapid pace as the goal is approached". This is possibly the first and most straightforward potential field approach recorded for motion guidance [28]. The first application in Robotics was developed by Loef and Soni [29, 30] for controlling the manipulator's arms. In their work, obstacles produce "influences" in the velocity commands and push the robot away from them.

The popularization of this approach dates back to the 1980s when Kathib [31] introduced the concept of artificial potential fields (APFs) for real-time obstacle avoidance of manipulators and mobile robots. The core idea was that the robot would move in a field of forces generated from the composition of an attractive potential function toward the goal and repulsive potential functions calculated based on the surface of the obstacles. Both potential fields are computed using the relative distance from the robot to the goal or obstacles. By taking the gradient of the composed potential function at each point in space, a vector field is obtained to control the robot. Since the vector field assigns a motion for

the robot at any arbitrary configuration, the great advantage of this framework is that it can be considered a feedback control approach, being robust to errors in actuation and sensing [32]. This approach also provides a different paradigm for robot control. Instead of requiring a high-level controller to convert a goal to a sequence of reference commands and then following these commands using a low-level controller, which can create incompatibility issues (it is not guaranteed that the converted reference can be followed) vector fields can combine both controllers into a single module [28].

Given these potential fields, finding the path to the goal from a given start position is implicitly equivalent to a gradient descent procedure. Consequently, it is prone to the same limitation – the method does not guarantee convergence to the global minimum. That happens when the gradient is zero due to the balancing of attractive and repulsive forces anywhere different than the goal. This inherent limitation was noted even in [29], "another feature which is not implemented, but which could be so if needed is logic decision capabilities, which would allow the selection of a different trajectory if the previously generated one leads to a dead end."

From then on, many techniques were developed to improve and extend the capabilities of potential or vector fields. Particular types of artificial potential functions were also proposed to mitigate the problem of converging to local minima. Navigation functions [33], for example, are potential functions specially designed – but not necessarily easy to obtain – to have a single minimum. They guarantee that, from any point in free space, the robot will reach the goal if it is reachable. Random motions were used in the Randomized Path Planner (RPP) [34] to escape local minima in potential fields. Navigation functions can be found numerically when grids can represent the configuration space. Wavefront planners [35] can be used to find a distance-based resolution-complete potential function by incrementally increasing the cost-to-go value of the cells starting from the goal. But, similar to other grid-based methods, the method might become computationally too expensive in higher dimensions. Alternatively, Connolly et al. [36] used harmonic functions, which

rely on the numerical solution of Laplace's equation, to find potential fields.

Compared to sampling-based approaches, a downside of the potential field approach is that they need a representation of the configuration space. Vector fields have also been generated from the discretization of the environment. The trivial example would be considering the wavefront planner cited above and assigning vectors to each of the grid cells in the direction of reducing cost-to-go. Lindemann and LaValle [37] construct smooth vector fields from a cell decomposition of the configuration space, typically convex decomposition of a general polygonal environment, by creating local vector fields for each cell and for the faces connecting cells and then blending them using bump functions, which are functions that transition from zero to one. Pimenta et al. [38] discretizes the configuration space into a sequence of neighboring discrete regions, which is triangulated. Base vectors that provide general direction that should be followed are calculated at the triangle's vertices. Then, to generate the continuous vector field for each point internal to the triangles, they are interpolated using a convex combination.

In this dissertation, we contribute to the field with a method to generate a vector field without an explicit representation of free space. We solve the motion planning problem with a sampling-based approach and use the resulting graph to create a vector field. This vector field transforms the solution from the randomly discretized to continuous space. The resulting policy guides the robot to its goal. This is similar to what the sampling-based neighborhood graph (SNG), proposed by Yang and LaValle [39], does. It uses a sampling-based approach to find balls that cover the free space. Each of these balls is assigned a local navigation function, which is guaranteed to take the robot into a ball closer to the goal. Instead of finding local functions similar to [38], we prefer an interpolation approach.

Vector fields can also be constructed *ad hoc*, handcrafted to provide a robot with a trajectory that has specific properties. Sometimes, these are called "guiding vector fields". Lawrence et al. [40] presented techniques to construct vector fields incorporating Lyapunov stability properties developed for UAVs. The method is first illustrated with a vector field

that makes the robot follow a circle. Then, they provide forms of switching between circular loiter vector fields to encode racetrack loitering and waypoint following. Many works have then defined guiding vector fields [41]. Notably, they can be generated for circulation of time-varying curves for perimeter surveillance [42], following corridors [43, 44], or coordination of multiple robots [45]. One application that has gotten some attention is using vector fields for the landing of UAVs [46, 47, 48, 49, 50]. Besides being a feedback approach, which provides robustness against external disturbances and sensing errors, vector fields have the advantage of specifying a shape for the path to be followed by the UAV. This application was first proposed by Kwon et al. [46], where vector field-guided landing for an airship subject to wind disturbance.

In our research lab, a vector field-based approach was developed by Gonçalves et al. [47] to guide UAVs toward the landing point and enforce a predetermined trajectory shape while landing. The strategy only required the relative position between the landing pad and the UAV. Landing on a moving platform is briefly mentioned but not the focus of that work. Martinez et al. [49] proposed a switched vector field that guides a tethered drone to the landing pad. The transitions between different modes in the vector field were obtained from a combination of tether elevation angle and height. Although the method was able to land the vehicle on static platforms efficiently, the selection of each mode relies on a series of conditions and thresholds that are difficult to tune, especially if the shape of the tether is allowed to change. The wrong selection of the parameters and a very slack tether could lead to undesirable situations that prevent the vehicle from landing. In sequence, a continuous velocity vector field [50] that relies on the tether variables only, namely tether angles (elevation and azimuth at the endpoints) and the tether length, was derived to mitigate this issue. Using Lyapunov stability properties, the proposed vector field was proven to correctly guide the vehicle to its desired landing position. Additionally, experiments showed landing on a moving platform by incorporating a feedforward term.

Many of the techniques mentioned before consider obstacles during the construction

of the vector field. That results in a vector field that needs to be changed every time the environment changes (e.g., obstacles are included or removed). Another approach was proposed by Pereira et al. [51], in which the constructed vector field provides a higher-level task for the robot and ignores some details of the environment, granting a fast field computation. Then, an optimal sampling-based planner (like the RRT*) is used to find a path that follows the vector field as much as possible but now considers the details of the environment. This concept is essential for this dissertation, where we develop a different type of local planner [52, 53], which can return paths that avoid obstacles while trying to follow a predefined vector field.

In the next chapter, we provide a methodology to solve the motion planning problem for vehicles moving in natural vector fields, where the flow can move so fast that the robot cannot counteract its presence. From the sampling-based solution, we also define a guiding vector field that can be used as a feedback motion plan to control a robot moving in these environments while rejecting small disturbances that can be present in their actuation.

Chapter 3

Robots Navigating in Strong Currents

This chapter proposes a motion planning methodology for holonomic vehicles moving in a flow field, where the flow speed can be greater than the vehicle's speed relative to the flow. The contents of this chapter are being submitted to the Autonomous Robots journal.

3.1 Introduction

In diverse applications, autonomous aerial and aquatic vehicles are required to work in environments with strong natural flows. In these applications, the flow can both assist or constrain the vehicle's motion. NASA's IceNode buoyant robots, for instance, exploit ocean currents to move along the basal ice-ocean interface of the Antarctic ice shelves acquiring long-duration melt rate measurements [54]. In another example, balloons developed by Loon [55] provide internet to remote places. Rather than flying against the winds, these balloons take advantage of the atmospheric winds to navigate without propulsion.

In an extraterrestrial application, NASA has considered the use of aerial robots to explore the cloud layer of the Venusian atmosphere [56], where recent observations have found evidence of phosphine [57], a compound associated with microbial presence. The strong winds of Venus, which may be as fast as $100\,\mathrm{m\,s^{-1}}$ [58], can help the vehicle to circulate the planet in five days, but can also prevent some latitudes to be reached sooner

than that [59]. To help with these applications, the methods proposed in this chapter and illustrated in Figure 3.1 compute optimal trajectories or motion policies for vehicles navigating natural flows. Chapter 4 shows how the methodology presented in this chapter, with a few extensions, can be used to navigate an aerobot in the windy atmosphere of Venus.

3.1.1 Related Work

In a previous work of our group, we used a sampling-based approach, based on the RRT algorithm, to create trajectories considering the strong winds of the atmosphere of Venus [59]. We first calculate connections without considering the wind and then integrate the wind along the calculated paths to calculate a wind drift. Although efficient, our method finds sub-optimal trajectories. An extension of the method to find optimal trajectories would require rewiring the search tree, as is done by RRT*, which is difficult to perform under the influence of strong winds. Very recently, a few research groups focused on applying the Fast Marching Tree star (FMT*) algorithm [60], which can find optimal paths faster and without tree rewiring. FMT* is used in [61] for energy-optimal planning of underwater gliders navigating time-invariant flows. The connection between nodes is performed by calculating the "trim states" required to achieve straight-line motion. In [62], FMT* is adapted to time-optimal planning of ultralight gliders navigating in time-invariant flows. The connections between nodes are calculated using a simplified optimal control problem that relies on a two-point boundary solver. Both papers consider 3D scenarios and raise reachability problems due to strong opposing flows. However, different from the analytical procedure proposed in this chapter, these methods numerically calculate the reachability set by solving the vehicle's equation of motion.

This chapter presents a different sampling-based algorithm based on FMT* to solve the optimal motion planning problem for holonomic vehicles navigating strong 3D environmental flows. Unlike previous FMT*-based methods, our algorithm explicitly handles the feasibility problem created by situations where the flow speed is larger than the vehicle's

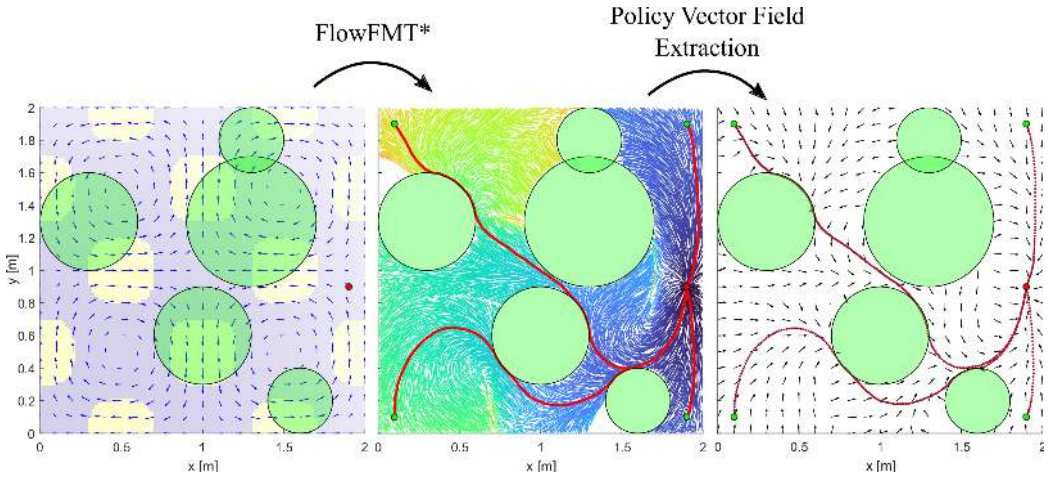


Figure 3.1: Given an environment with a time-invariant flow (→) and prohibited regions (■), the Policy-FlowFMT* algorithm proposed in this work creates a tree (represented in the center by vertices colored by their costs) that encodes the optimal motion policy to a specified goal (●). Besides being directly used to provide optimal trajectories to the goal (—), the tree can also generate a vector field (→), which can be applied online for robot control (· · ·).

maximum speed relative to the flow. To achieve that, we extended the use of reachability cones defined in [14] to 3D and used them to remove unreachable vertices in constructing the search tree. We also use reachability cones to determine accurate minimum-time and minimum-energy cost functions for the problem. Additionally, based on a few modifications to the proposed path planning algorithm, we present a method that finds a motion policy for the vehicle. The policy is represented as a vector field that serves as a feedback planner that optimally guides the vehicle from any valid configuration to its goal. An illustration of the method is shown in Figure 3.1. In summary, the main contributions of this chapter are (1) a motion planner based on FMT* that accounts for the feasibility of the trajectory in the presence of strong flows and prohibited regions (e.g., obstacles), (2) a method to obtain minimum-time and minimum-energy cost functions based on the reachability condition, (3) a method that creates a vector-field policy for optimal navigation in strong flows. We provide the source code of our methods and, for comparison, the setup environment for the equivalent optimal control solver (OCS) (https://bitbucket.org/wvufarolab/flowfmtstar).

3.1.2 Organization

The rest of this chapter is organized as follows. The next section introduces the problem we are solving in the chapter. The proposed methods and algorithms are explained in Sect. 3.3, and experiments are provided in Sect. 3.4. Conclusions and future work are presented in Sect. 3.5.

3.2 Problem Definition

This section specifies the robot motion planning problem solved in this work. We start by defining the environment and the model of our robot.

3.2.1 Environment Model

In this chapter, a single robot navigates within a fluid (most commonly air or water) situated in a tri-dimensional (3D) environment $\mathcal{W} \subset \mathbb{R}^3$. The fluid is assumed to be flowing and its motion is modeled as a time-independent vector field $\mathbf{F}_c : \mathcal{W} \mapsto \mathbb{R}^3$ that maps each position $\vec{x} = [x, y, z]^\mathsf{T} \in \mathcal{W}$ to a flow velocity vector $\vec{c} = [c_x, c_y, c_z]^\mathsf{T} \in \mathbb{R}^3$. The vector field $\mathbf{F}_c$ that represents the flow is continuous, i.e., given any point $\vec{x}_0 \in \mathcal{W}$, there is a ball $\mathcal{B}_r(\vec{x}_0) = \{\vec{x} \in \mathcal{W} \mid \|\vec{x} - \vec{x}_0\| < r\}$ of radius $r \in \mathbb{R}^+$ centered in $\vec{x}_0$, where $\|\mathbf{F}_c(\vec{x}) - \mathbf{F}_c(\vec{x}_0)\| < \epsilon$, for every arbitrary $\epsilon \in \mathbb{R}^+$. Intuitively, if we consider two points in space that are sufficiently close to each other, the velocities of the flow at these points are also similar. Finally, we assume that the environment may contain prohibited regions, $\mathcal{O}$, which may be seen as obstacles for the vehicle but not for the fluid (e.g., “no-fly zones” for an aircraft). Therefore, the region where the vehicle can safely navigate, or free space, is given by $\mathcal{W}_{free} = \mathcal{W} \backslash \mathcal{O}$.

3.2.2 Robot Model

We assume a holonomic vehicle, whose state space is given by $\mathcal{X} = X \times \dot{X}$, where $X \subseteq \mathcal{W}$ is the space of vehicle positions and $\dot{X}$ is its respective tangent space. We consider that the vehicle is small (meter scale) compared to the environment and the distances traveled (kilometer scale). With this assumption in mind, we use a point with no orientation to represent the vehicle. In practice, this point can be seen as the vehicle itself or, for vehicles with more complicated dynamic models, as a setpoint that could be tracked with the aid of a non-linear controller. Therefore, the vehicle's configuration space is identical to the workspace $\mathcal{W}$, where the vehicle is represented by its position $\vec{x}$ in the global reference frame. The velocity of the vehicle in the global reference frame is given by

$$\vec{v}_g = \vec{v}_r + \vec{c}, \tag{3.1}$$

where $\vec{v}_r$ is the velocity of the vehicle relative to the flow and $\vec{c} = \mathbf{F}_c(\vec{x})$ is the velocity of the flow at the current robot position $\vec{x}$. The relative velocity is bounded by a maximum speed v_r^{max}, such that $\|\vec{v}_r\| \leq v_r^{max}$. We assume that the flow speed can be greater than the vehicle's speed relative to the flow ($\|\mathbf{F}_c(\vec{x})\| \geq v_r^{max}$), which means that the vehicle may not be able to counteract the flow all over the environment.

3.2.3 Problem Statement

The motion planning problem solved in this chapter is to find the best feasible trajectory $\sigma(t)\colon [0, t_f] \rightarrow \mathcal{W}_{free}$ parameterized by $t \in [0, t_f]$ that moves the vehicle from an initial configuration $\vec{x}_S$ at $t = 0$ to a set of admissible goal configurations $\vec{x}_G \in X_{goal}$ in a finite time $t = t_f$. The cost function $C(\sigma)$ to be minimized by our problem maps the trajectory to a positive real value $C : \mathbb{R}^3 \times \mathbb{R} \mapsto \mathbb{R}^+$ and respects the constraints imposed on the vehicle by the environment. Time and energy cost functions will be minimized in this chapter.

Once these functions are defined, our problem is posed as:

$$\begin{aligned} \min_{\sigma} \quad & C(\sigma) \\ \text{s.t.} \quad & \vec{v}_g = \vec{v}_r + \vec{c}\,, \\ & \sigma(0) = \vec{x}_S\,, \quad \sigma(t_f) = \vec{x}_G\,, \quad \sigma(t) \in \mathcal{W}_{free}. \end{aligned} \tag{3.2}$$

3.3 Methodology

This section presents our motion planning algorithm. First, we introduce minimum-time and minimum-energy cost functions necessary to specify our problem completely. Then, we define neighborhood functions that are created using reachability cones. Finally, we describe the Flow-Aware Fast Marching Tree algorithm and an adaptation to compute motion policies.

3.3.1 Flow-Aware Cost Functions

The problem presented in the previous section is not entirely specified without a cost function. The literature uses two main categories of cost functions, depending on the task [63]. The first specifies the minimum time problem, which tries to guide the vehicle to its goal as fast as possible, regardless of the energy spent. The other type specifies the minimum energy problem, which is useful when the vehicle has a finite amount of energy available (e.g., battery) and is required to spend this energy efficiently. Functions in these two main categories are defined next.

Minimum-Time Cost Function

In early attempts to solve the minimum-time problem, the cost functions used were incorrect in the presence of strong flows (i.e., $\|\vec{c}\| \geq \|\vec{v}_r\|$). That meant that trajectories found using such cost functions were not necessarily feasible. The limited reachability problem due to strong flows is illustrated in Figure 3.2. In this chapter, we provide an

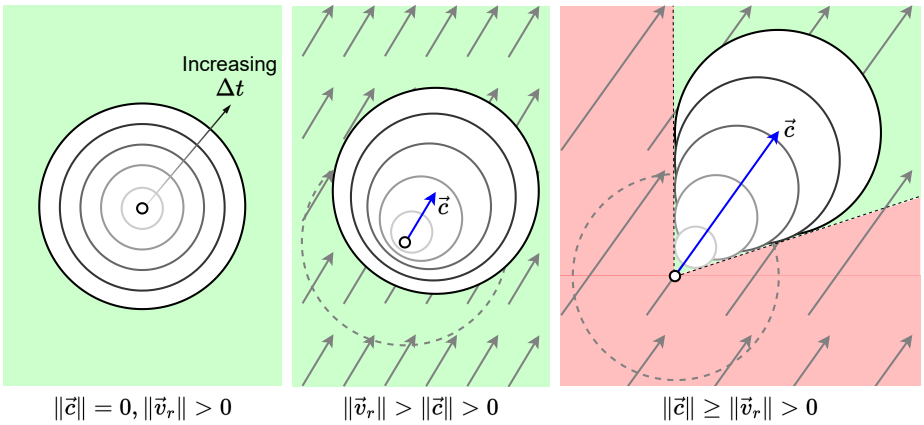


Figure 3.2: The reachable region for a vehicle moving on a constant flow is greatly affected by the ratio between the flow velocities and the vehicle's velocity relative to the flow. The circles represent iso-temporal lines given by the position reached by the vehicle after moving towards any direction for a fixed period at a constant speed $\|\vec{v}_r\|$. When the flow velocity $\|\vec{c}\| = 0$, the circles are concentric and expand outwards with time, thus allowing the vehicle to reach the entire space (green). With $\|\vec{v}_r\| > \|\vec{c}\| > 0$, the circles are drifted accordingly, indicating that the vehicle travels less when moving against the flow but still reaches the entire space (green). When $\|\vec{c}\| \geq \|\vec{v}_r\| > 0$, the vehicle cannot counteract the flow, which creates a region that cannot be reached (red) even if the vehicle's velocity relative to the flow is directly opposed to the flow.

extension to 3D of the correct minimum-time cost function proposed in [14] to solve 2D problems (we recommend the reader to read the discussion about cost function incorrectness in [14]). Minimum-time cost functions are usually defined for continuous trajectories and n-waypoints discretizations, respectively, as:

$$C_t(\sigma) = \int_0^{t_f} dt = t_f \quad \text{and} \quad C_t(\sigma) \approx \sum_{k=0}^{n} \Delta t^{(k)} \,. \tag{3.3}$$

Although these equations are correct, previous work computed the travel time between two sufficiently close configurations (Δt) with simple heuristics that accounted for the traveled distance and an additional term that considered the relative heading of the flow and the vehicle [64] (thus assuming constant speed) or a ratio between the traveled distance and the sum of the vehicle speed and the speed of the flow opposing the movement [13, 65]. None of these heuristics were able to handle the strong flow condition.

When the flow speed gets larger than the maximum vehicle's speed relative to the flow, the drift caused by the flow cannot be compensated, thus creating regions that cannot be reached by the vehicle (Figure 3.2).

To create discrete functions that consider this characteristic, we follow the same procedure of [14] but extend it to 3D. Because the vehicle is holonomic, we assume it can change direction instantaneously and that the minimum time path between two waypoints under a constant flow vector is a straight line. We also assume that two consecutive waypoints are close enough so the flow is constant between them. Thus, considering two positions $\vec{x}(t_A)$ and $\vec{x}(t_B)$, we can define a directional edge as $\vec{d} = \vec{x}(t_B) - \vec{x}(t_A) = [d_x, d_y, d_z]^\mathsf{T}$. The time interval between these two positions is $\Delta t = t_B - t_A$. By integrating the model in (3.1) and considering the vehicle's relative velocity $\vec{v}_r$ and flow velocity $\vec{c}$ to be constant along this edge we have:

$$\vec{d} = \vec{v}_g \Delta t = (\vec{v}_r + \vec{c})\Delta t. \Leftrightarrow \begin{cases} (v_{r,x} + c_x)\Delta t = d_x \\ (v_{r,y} + c_y)\Delta t = d_y \\ (v_{r,z} + c_z)\Delta t = d_z. \end{cases} \tag{3.4}$$

By incorporating this equation on the inner product of the relative velocity ($\vec{v}_r \cdot \vec{v}_r = v_{r,x}^2 + v_{r,y}^2 + v_{r,z}^2$), we obtain an equation for the 3D cone shown in Figure 3.3 as:

$$(\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)\Delta t^2 - 2(\vec{d} \cdot \vec{c})\Delta t + \vec{d} \cdot \vec{d} = 0\,. \tag{3.5}$$

For some given $\vec{v}_r$, $\vec{c}$, and $\vec{d}$, the cone angle β is derived from the geometry of the problem as:

$$\cos\beta = \frac{\sqrt{\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r}}{\|\vec{c}\|} = \frac{\vec{d} \cdot \vec{c}}{\|\vec{d}\|\|\vec{c}\|}\,. \tag{3.6}$$

To check if a straight line connection between two points is feasible, we can check if $\cos(\beta) > \cos(\beta_{max})$, where $\cos(\beta_{max})$ is found when $\|\vec{v}_r\| = v_r^{max}$.

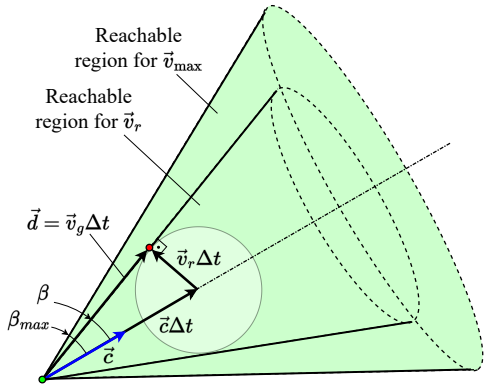


Figure 3.3: Computation of the reachable region. Given a time interval Δt, the displacement due to the flow is given by $\vec{c}\Delta t$, while the displacement due to the vehicle's velocity relative to the flow is the surface of a sphere of radius $\vec{v}_r \Delta t$. A three-dimensional cone is formed by taking different time intervals. The axis of the cone is aligned with the flow velocity ($\vec{c}$), and the angle β is calculated from both the flow velocity and $\vec{v}_r$. The largest reachable cone is defined by β_{max}, which is obtained when $\|\vec{v}_r\| = v_r^{max}$.

Finally, the time cost for the straight line path between the two vertices is obtained by solving for Δt:

$$\Delta t = \frac{\vec{d} \cdot \vec{c} \pm \sqrt{(\vec{d} \cdot \vec{c})^2 - (\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)(\vec{d} \cdot \vec{d})}}{(\vec{c} \cdot \vec{c} - \vec{v}_r \cdot \vec{v}_r)} . \tag{3.7}$$

If the path is infeasible, the computation of Δt in (3.7) would return complex numbers without physical meaning. However, as mentioned above, this case can be ruled out *a priori* using β_{max}. If the path is feasible, regardless of the flow velocity and the vehicle's relative velocity magnitudes ratio, (3.7) returns the minimum time between the two points for both the 2D and 3D cases if the vehicle moves with its maximum speed (i.e., $\vec{v}_r \cdot \vec{v}_r = (v_r^{max})^2$) and we consider the negative sign solution. If the flow velocity is smaller than the relative velocity, the positive sign would have led to a negative Δt, which can be disregarded. Interestingly, when the flow velocity is stronger than the relative velocity, the two possible values for Δt are positive and valid. However, the lower-time solution should be chosen for the minimum-time problem. To compute the cost of a trajectory, the time cost Δt in (3.7) is used in (3.3) for each segment that composes the n piece-wise linear path traversed by the vehicle from start to goal.

Minimum-Energy Cost Functions

Common energy-based cost functions for continuous and discrete trajectories are, respectively, given by:

$$C_e(\sigma) = \int_0^{t_f} P(t)dt \quad \text{and} \quad C_e(\sigma) \approx \sum_{k=0}^{n} P^{(k)}\Delta t^{(k)}\,, \tag{3.8}$$

where P is the power required to overcome the drag. This required power can be modeled as a polynomial of the vehicle's speed relative to the flow to represent diverse forms of drag (e.g., viscous, pressure, or lift-induced drag):

$$P(t) = \sum_{i=0}^{n} \alpha_i \|\vec{v}_r(t)\|^i\,. \tag{3.9}$$

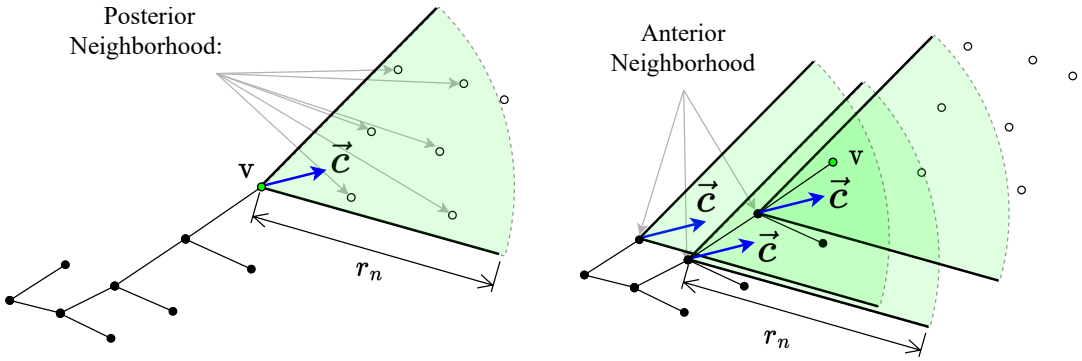
From (3.9) and (3.8), we observe that to minimize the cost function, it is necessary to minimize the vehicle's speed relative to the flow. However, as mentioned before, small relative speeds may generate feasibility problems. We propose a solution for this problem by taking the limit situation that allows the vehicle to reach the goal defined by a given edge $\vec{d}$. We then use the cone of reachability defined in the previous subsection to find the speed v_r^{min} that minimizes the cost as:

$$\frac{\sqrt{\vec{c}\cdot\vec{c} - (v_r^{min})^2}}{\|\vec{c}\|} = \frac{\vec{d}\cdot\vec{c}}{\|\vec{d}\|\|\vec{c}\|} \Leftrightarrow v_r^{min} = \sqrt{\vec{c}\cdot\vec{c} - \left(\frac{\vec{d}\cdot\vec{c}}{\|\vec{d}\|}\right)^2}\,, \tag{3.10}$$

which can be used with (3.9) and (3.7) to calculate $P(t)^{(k)}$ and $\Delta t^{(k)}$, which are then used in (3.8) to calculate the energy cost for each edge of the path.

3.3.2 Flow-Aware Neighborhood Sets

Sampling-based motion planners, such as the one proposed in this chapter, usually rely on the concept of neighborhood, which is usually used to specify a set of vertices (samples)



(a) (b)

Figure 3.4: Two types of neighborhood sets are required by FlowFMT*, substituting the standard Near() function used by the original FMT* algorithm. Given a vertex v, we define (a) the posterior neighborhood, with vertices that can be reached from v, and (b) the anterior neighborhood, with vertices that can reach the vertex v.

that are candidates to be connected to the current vertices of a graph during its construction. One of the novelties of our method is the division of the neighborhood set into posterior and anterior sets. The posterior neighborhood set of a given vertex v, as shown in Figure 3.4(a), is the set of vertices that respect two conditions: 1) the vertices can be reached from v (i.e., they are contained in the reachability cone defined by v) and 2) the vertices are within a distance r_n from v. On the other hand, the anterior neighborhood set of a vertex v, as shown in Figure 3.4(b), is the set of vertices that respect two conditions: 1) the reachability cone defined by these vertices contains v and 2) the vertex v is within a distance r_n from these vertices. The definition of these two sets allows us to handle the reachability constraint caused by the presence of strong currents when creating a sample-based algorithm, as explained next.

3.3.3 Flow-Aware Fast Marching Tree Star

Using the previously defined cost functions and neighborhood sets, this section presents specialized Fast Marching Tree Star algorithms (FMT*) [60] that consider the reachability constraints imposed by the strong currents. We first show the standard version of the algorithm, for which the tree grows from the start position until it reaches the goal region. In

the second part of the section, we slightly change the algorithm so it expands the tree from the goal position to every feasible point sampled in the environment and creates a policy.

Standard Flow-Aware FMT*

The Flow-Aware FMT* algorithm (FlowFMT*) proposed in this chapter is shown in Algorithm 1. Changes with respect to the original FMT* algorithm [60] are shown in red. The algorithm creates a tree graph with an empty edge set and the vertex set defined by m random samples of the space, the start, and the goal positions (line 2). Three sets of vertices (line 3) are created: the set of unvisited vertices, initialized with all vertices but the start position; the open set, which initially has the start position; and the closed set, which is empty at the beginning. Then, two structures (N^A and N^P) that save two sets of neighbors (see Sect. 3.3.2) for each vertex are initialized (lines 5–8). The distance r_n is automatically defined by the number of samples, dimension of the free space, size of a unit ball for the corresponding dimension, and a tuning parameter η. The complete derivation for this distance can be found in [60]).

The algorithm executes until no vertices are found on the open set or a path to the goal is found (lines 9 and 26). At each iteration, the minimum cost vertex of the open set is computed (line 29), and the posterior neighborhood of this vertex is calculated or retrieved (lines 30–31). This set is intersected with the set of unvisited vertices (line 11), and for each of the vertices x in the intersection, a search for a locally-optimal one-step connection is performed by testing connections from all the vertices y in the intersection of the anterior neighborhood and the open set to x (lines 12–16). Up to this point, no collision checks are done. Once the locally-optimal one-step connection is found, it is tested for collision (line 17). If no collisions are detected, the edge is added to the tree (line 18). The newly added vertex is added to the open set and removed from the unvisited set (lines 19–20). After looping through all vertices in the posterior neighborhood, the query vertex is removed from the open set and added to the closed set (lines 24–25). The neighborhoods are found

Algorithm 1 Flow-Aware FMT* Algorithm

function $\sigma = \text{FLOWFMT*}(\mathrm{x}_S, \mathrm{x}_G, V_{pool})$
  $V \leftarrow \mathrm{x}_S \cup V_{pool} \cup \mathrm{x}_G$, $E \leftarrow \varnothing$
  $V_{unvisited} \leftarrow V \backslash \{\mathrm{x}_S\}$, $V_{open} \leftarrow \mathrm{x}_S$, $V_{closed} \leftarrow \varnothing$
  $\mathrm{z} \leftarrow \mathrm{x}_S$
  $N^P_{\mathrm{z}} \leftarrow \mathit{PosteriorNeighborhood}(\varnothing, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$
  $N^P \leftarrow \mathit{Save}(\varnothing, N^P_{\mathrm{z}}, \mathrm{z})$
  $N^A_{\mathrm{z}} \leftarrow \mathit{AnteriorNeighborhood}(\varnothing, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$
  $N^A \leftarrow \mathit{Save}(\varnothing, N^A_{\mathrm{z}}, \mathrm{z})$
  while $\mathrm{z} \neq \mathrm{x}_G$ **do**
    $V_{open,new} \leftarrow \varnothing$
    $X_{near} = N^P_{\mathrm{z}} \cap V_{unvisited}$
    for $\mathrm{x} \in X_{near}$ **do**
      $N^A_{\mathrm{x}} \leftarrow \mathit{AnteriorNeighborhood}(N^A, V \backslash \{\mathrm{x}\}, \mathrm{x}, r_n)$
      $N^A \leftarrow \mathit{Save}(N^A, N^A_{\mathrm{x}}, \mathrm{x})$
      $Y_{near} \leftarrow N^A_{\mathrm{x}} \cap V_{open}$
      $\mathrm{y}_{min} \leftarrow \text{argmin}_{\mathrm{y} \in Y_{near}} \{c(\mathrm{y}) + \mathit{Cost}(\mathrm{y}, \mathrm{x})\}$
      if $\mathit{CollisionFree}(\mathrm{y}_{min}, \mathrm{x})$ **then**
        $E \leftarrow E \cup \{(\mathrm{y}_{min}, \mathrm{x})\}$
        $V_{open,new} \leftarrow V_{open,new} \cup \{\mathrm{x}\}$
        $V_{unvisited} \leftarrow V_{unvisited} \cap \{\mathrm{x}\}$
        $c(\mathrm{x}) = c(\mathrm{y}_{min}) + \mathit{Cost}(\mathrm{y}_{min}, \mathrm{x})$
      end if
    end for
    $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \backslash \{\mathrm{z}\}$
    $V_{closed} \leftarrow V_{closed} \cup \{\mathrm{z}\}$
    if $V_{open} = \varnothing$ **then**
      return $\varnothing$
    end if
    $\mathrm{z} \leftarrow \text{argmin}_{\mathrm{y} \in V_{open}} \{c(\mathrm{y})\}$
    $N^P_{\mathrm{z}} \leftarrow \mathit{PosteriorNeighborhood}(N^P, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$
    $N^P \leftarrow \mathit{Save}(N^P, N^P_{\mathrm{z}}, \mathrm{z})$
  end while
  return $\mathit{GetPath}(\mathrm{x}_G, \mathcal{T}(V_{open} \cup V_{closed}, E))$
end function

using 2.

Computing a Policy Using Flow-Aware FMT*

More valuable than simply obtaining a path from the start to the goal, a policy defines paths from all positions in the space to the goal. A policy can be represented by a vector

Algorithm 2 Anterior/Posterior Neighborhood Function

1: **function** $N_{\mathrm{q}} = \textsc{Neighborhood}(\text{'type'}, N, V, \mathrm{q}, r_n)$
2: **if** $\exists N(\mathrm{q})$ **then**
3: **return** $N(\mathrm{q})$
4: **end if**
5: $N_{\mathrm{q}} \leftarrow \varnothing$
6: $\vec{q} \leftarrow GetPosition(\mathrm{q})$
7: $\vec{c} \leftarrow \mathbf{F}_c(\vec{q})$
8: $\cos(\beta_{max}) \leftarrow \sqrt{\vec{c} \cdot \vec{c} - (v_r^{max})^2} / \|\vec{c}\|$
9: **for** $\mathrm{v} \in V$ **do**
10: $\vec{x} \leftarrow GetPosition(\mathrm{v})$
11: **if** 'type' = 'A' **then** ▷ Anterior Neighborhood
12: $\vec{d} \leftarrow \vec{q} - \vec{x}$
13: **else** ▷ Posterior Neighborhood
14: $\vec{d} \leftarrow \vec{x} - \vec{q}$
15: **end if**
16: $\cos\beta \leftarrow (\vec{d} \cdot \vec{c}) / (\|\vec{d}\| \|\vec{c}\|)$
17: **if** $\|\vec{d}\| < r_n$ **then**
18: **if** $\|\vec{c}\| \geq v_r^{max}$ **then**
19: **if** $\cos\beta \geq \cos(\beta_{max})$ **then**
20: $N_{\mathrm{q}} \leftarrow N_{\mathrm{q}} \cup \mathrm{v}$
21: **end if**
22: **else**
23: $N_q \leftarrow N_q \cup \mathrm{v}$
24: **end if**
25: **end if**
26: **end for**
27: **return** N_{q}
28: **end function**

field, which can be used as a feedback system to compensate for disturbances [8]. Since the FMT* method is a graph-based level-set method that computes increasing levels of the cost function as the tree marches to cover the environment, it can be easily used to compute a policy. We then propose a few modifications to the Flow-Aware FMT* algorithm (Algorithm 1) to make this possible: (i) the root of the tree is set to the goal position (lines 2–4 of Algorithm 1); (ii) the anterior (lines 13–14) and posterior (lines 30–31) neighborhood sets are swapped; (iii) the order of the vertices on the cost function and collision checking functions are inverted (lines 16–17, and 21); (iv) the stopping criteria (lines 9 and 26 of Algorithm 1) is changed to only interrupt the procedure if the open set becomes empty.

Algorithm 3 Flow-Aware FMT* Algorithm (Policy Version)

1: **function** $\mathcal{T} = \text{POLICY FLOWFMT*}(\mathrm{x}_G, V_{pool})$
2: $V \leftarrow \mathrm{x}_G \cup V_{pool}$, $E \leftarrow \varnothing$
3: $V_{unvisited} \leftarrow V \backslash \{\mathrm{x}_G\}$, $V_{open} \leftarrow \mathrm{x}_G$, $V_{closed} \leftarrow \varnothing$
4: $\mathrm{z} \leftarrow \mathrm{x}_G$
5: $N^P_{\mathrm{z}} \leftarrow PosteriorNeighborhood(\varnothing, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$
6: $N^P \leftarrow Save(N^P, N^P_{\mathrm{z}}, \mathrm{z})$
7: $N^A_{\mathrm{z}} \leftarrow AnteriorNeighborhood(\varnothing, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$
8: $N^A \leftarrow Save(N^A, N^A_{\mathrm{z}}, \mathrm{z})$
9: **while** $V_{open} \neq \varnothing$ **do**
10: $V_{open,new} \leftarrow \varnothing$
11: $X_{near} = N_{\mathrm{z}} \cap V_{unvisited}$
12: **for** $\mathrm{x} \in X_{near}$ **do**
13: $N^P_{\mathrm{x}} \leftarrow PosteriorNeighborhood(N^P, V \backslash \{\mathrm{x}\}, \mathrm{x}, r_n)$
14: $N^P \leftarrow Save(N^P, N^P_{\mathrm{x}}, \mathrm{x})$
15: $Y_{near} \leftarrow N^P_{\mathrm{x}} \cap V_{open}$
16: $\mathrm{y}_{min} \leftarrow \text{argmin}_{\mathrm{y} \in Y_{near}} \{c(\mathrm{y}) + Cost(\mathrm{x}, \mathrm{y})\}$
17: **if** $CollisionFree(\mathrm{y}_{min}, \mathrm{x})$ **then**
18: $E \leftarrow E \cup \{(\mathrm{y}_{min}, \mathrm{x})\}$
19: $V_{open,new} \leftarrow V_{open,new} \cup \{\mathrm{x}\}$
20: $V_{unvisited} \leftarrow V_{unvisited} \cap \{\mathrm{x}\}$
21: $c(\mathrm{x}) = c(\mathrm{y}_{min}) + Cost(\mathrm{x}, \mathrm{y}_{min})$
22: **end if**
23: **end for**
24: $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \backslash \{\mathrm{z}\}$
25: $V_{closed} \leftarrow V_{closed} \cup \{\mathrm{z}\}$
26: $\mathrm{z} \leftarrow \text{argmin}_{\mathrm{y} \in V_{open}} \{c(\mathrm{y})\}$
27: $N^A_{\mathrm{z}} \leftarrow AnteriorNeighborhood(N^A, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$
28: $N^A \leftarrow Save(N^A, N^A_{\mathrm{z}}, \mathrm{z})$
29: **end while**
30: **return** $\mathcal{T}\,(V_{open} \cup V_{closed}, E)$
31: **end function**

The outcome of these modifications is shown in Algorithm 3. Changes with respect to the standard FlowFMT* algorithm (Algorithm 1) are shown in blue. Notice that Algorithm 3 builds a tree from the goal position but still considers that the direction of motion of the vehicle is towards the goal. The dynamic programming procedure (line 16) considers only valid connections given the sets defined in Sect. 3.3.2.

A policy is encoded in the resulting tree. Once this tree is found, the vehicle's velocity in the global reference frame $\vec{v}_g$ can be calculated using a multivariate interpolation around

each position $\vec{x}$ in the continuous space. We propose using an inverse distance weighting interpolation of the vertices of the tree that are close to $\vec{x}$ to extract the policy vector field as:

$$\vec{v}_r(\vec{x}) = \frac{\sum_{i=0}^{k} w_i(\vec{x})\, \vec{v}_{g,i}}{\sum_{i=0}^{k} w_i(\vec{x})} - \mathbf{F}_c(\vec{x})\,, \tag{3.11}$$

where $\vec{v}_{g,i}$, $0 \leq i \leq k$ are the vehicle's velocities in the global reference frame at the k vertices v_i of the tree that are within a distance r from $\vec{x}$ (i.e., $\mathrm{v}_i \in V \cap \mathcal{B}_r(\vec{x})$). The weight function is given by $w_i(\vec{x}) = 1/(\vec{x} - \vec{x}_i)^p$ where p is a parameter (commonly, $p = 2$). Notice that we assume that $\vec{x}$ is located in a region from which the goal can be reached. We do not include any further development in this chapter, but simple heuristics, like counting the number of vertices of the tree in the ball $\mathcal{B}_r(\vec{x})$ can help identify whether $\vec{x}$ belongs to a feasible region or not. From (3.1) the vehicle's velocity relative to the flow can be computed by subtracting the interpolated value from the flow velocity at this point.

3.4 Experimental Results

In this section, the performance of the proposed planner is evaluated in 2D and 3D scenarios, with and without obstacles. The resulting paths are compared to the paths obtained by a state-of-the-art optimal control solver (OCS), the Imperial College London Optimal Control Software (ICLOCS2) [66]. All the simulations were performed using an Intel® Core™ i9-9900K CPU at $3.6\,\mathrm{GHz}$, with 16 cores and 32GB of RAM. We consider a vehicle that can move at a maximum speed $v_r^{max} = 0.05\,\mathrm{m\,s^{-1}}$ relative to the flow. The environment is modeled using a three-dimensional (3D) version of double-gyre flow, which is usually used to evaluate motion planning on flows (see [16, 61]), as:

$$\vec{c}(x,y,z) = \begin{bmatrix} -\pi A \sin(\pi x/s)\cos(\pi y/s)\cos(\pi z/s) \\ \pi A \cos(\pi x/s)\sin(\pi y/s)\cos(\pi z/s) \\ \pi A \sin(\pi z/s) \end{bmatrix} \tag{3.12}$$

where A is the amplitude scaling factor that controls the maximum flow speed, $c_{max} = \max_{\vec{x}\in\mathcal{W}} \|\vec{c}(\vec{x})\|$, and s determines the characteristic length of the gyres. As suggested in [16], we choose $A = 0.02$ so that the maximum flow speed is $0.0625\,\mathrm{m\,s^{-1}}$, which is greater than the vehicle's maximum speed relative to the flow. We limit the environment to a box of side $2\,\mathrm{m}$, i.e. $\mathcal{W} = [0,2] \times [0,2] \times [0,2]$ (the dimensional unit 'meters' is omitted in the rest of this chapter for simplicity), and choose $s = 1$. To obtain the standard two-dimensional (2D) double-gyre flow, the slice where $z = 0$ is used.

3.4.1 Two-Dimensional Environment

In a 2D environment, the Flow-Aware FMT* (FlowFMT*) is used to find paths from the start position $\vec{x}_{start} = [0.1, 0.1]^\mathsf{T}$ to a few different goal positions $X_{goal} = \{[0.1, 1.9]^\mathsf{T}, [1.5, 1.0]^\mathsf{T}, [1.9, 0.9]^\mathsf{T}, [1.9, 1.1]^\mathsf{T}, [1.9, 1.9]^\mathsf{T}\}$, as shown in Figure 3.5. This figure shows the optimal-time paths for both minimum-time and minimum-energy cost functions and the resultant trees used to obtain these paths. The edges of the tree were colored by the cost of the vertices. The solution from the equivalent optimal control problem is shown for comparison. The general topology of the path for both methods is the same. Obtaining the best path from the OCS requires some tuning and an appropriate initial guess. Without these two requirements, the OCS can take too long to converge (several minutes) or converge to a local optimum. We noticed that as we increase the number of samples, the path generated by FlowFMT* approaches the best path obtained by the OCS without the extra work. Numerical comparisons for all the paths shown are presented in Table 3.1.

The absolute values of the minimum-time cost obtained by FlowFMT* and the OCS solution agree with the results shown in [16], despite the fact that the piece-wise time cost between vertices was obtained using different formulations (ours is based on [14]). For comparison, considering the path from $[0.1, 0.1]^\mathsf{T}$ to $[1.9, 0.9]^\mathsf{T}$, the OCS solution obtained by [16] presented a minimum-time cost of $C_t(\sigma) = 32.87\,\mathrm{s}$ while their proposed solution had $C_t(\sigma) = 32.92\,\mathrm{s}$. Meanwhile, we observed a cost of $C_t(\sigma) = 32.86\,\mathrm{s}$ using OCS

(set up with ICLOCS2), and $C_t(\sigma) = 32.88\,\mathrm{s}$ using FlowFMT*. Our method ran with 40,000 samples to match the spatial resolution of [16]. This indicates that both approaches generate similar valid solutions for the minimum-time problem.

Finding the true absolute values for the optimal paths considering the minimum-energy cost function is a more difficult task in the scenario involving gyres. Since the double-gyre flow model has parallel streamlines that circulate around the center of each gyre, a vehicle without actuation would drift around the center of its current gyre, carried by the flow. To reach a position in a different streamline while minimizing the energy spent, the vehicle must move from streamline to streamline with the minimum speed possible. In an ideal scenario, this speed and its resultant energy cost could be as small as we wanted. Clearly, minimal costs would also result in very long-time solutions. A discussion related to this problem is found in [63]. We then conclude that the true minimum energy cost is dependent on the minimum speed that the vehicle produces relative to the flow. This issue is not elucidated in [16] when their OCS solution was explained. To allow the OCS to find a solution with a finite and reasonable time, we set the inferior limit of the vehicle's speed relative to the flow to $0.0001\,\mathrm{m\,s^{-1}}$ (0.02% of the maximum relative speed). For FlowFMT*, we do not enforce a minimum speed since it is calculated using (3.10), which will be zero only in the rare cases where the next waypoint is completely aligned with the flow. Figure 3.5 shows that our method can find solutions similar to the ones found by the OCS.

Table 3.1: Performance metric of the trajectories shown in Figure 3.5 computed by FlowFMT* and ICLOCS2 [66].

Goal $\vec{x}_g$	**Time cost** $C_t(\sigma)$ [s]		**Energy cost** $C_e(\sigma)$ [μJ]	
	FlowFMT*	ICLOCS2	FlowFMT*	ICLOCS2
$[1.9, 0.9]^\mathsf{T}$	32.89	32.86	1.120	2.532
$[1.9, 1.1]^\mathsf{T}$	35.12	35.06	1.779	1.747
$[1.5, 1.0]^\mathsf{T}$	34.47	34.43	0.834	0.737
$[1.9, 1.9]^\mathsf{T}$	30.15	30.11	1.681	1.204
$[0.1, 1.9]^\mathsf{T}$	27.58	27.62	1.584	2.143

The average computational time spent by FlowFMT* to compute the paths in Figure 3.5 was $74\,\mathrm{s}$, while the OCS was "generally" able to finish in a few seconds. However, we also observed that the OCS computational time can increase drastically (up to minutes) if obstacles are added and, consequently, more active constraints are in place or if the problem is not set up correctly (poor initial guess, poor constraint limits, etc.). Furthermore, OCS solutions tend to converge to local minima differently from our graph-based solution. Depending on the initial guess, the OCS solutions either get stuck in suboptimal homotopies or show additional loops around gyres, introducing suboptimality. An example can be seen in Figure 3.7(a), which shows the solution of FlowFMT* in the presence of prohibited regions (obstacles for the robot but not for the flow) and two OCS solutions considering different initial guesses. With a good initial guess, the OCS shows an optimal cost of $C_t(\sigma) = 40.24\,\mathrm{s}$, which is similar to the FlowFMT*'s solution cost of $C_t(\sigma) = 40.91\,\mathrm{s}$. However, with a "bad" initial guess, OCS finds a solution in a different homotopy and with an optimal cost of $C_t(\sigma) = 43.46\,\mathrm{s}$.

The Policy-FlowFMT* algorithm was also compared to the solutions obtained with ICLOCS2. First, we found trajectories from 20 random initial positions to a single final position $\vec{x}_{goal} = [1.9, 0.9]^\mathsf{T}$ in the double-gyre environment without prohibited regions. We provided a straight line between the start and goal as the initial guess for the OCS. Our algorithm was tested in different sample densities, doubling the number of samples from 100×2^0 to 100×2^{10}. The results are shown in Figure 3.6. It is possible to see that 2 of the 20 trajectories were dissonant, with FlowFMT* finding lower-cost solutions (in fact, better solutions). As mentioned, this could be solved by providing a better initial guess for the OCS. Figure 3.6(b) shows how close the cost found by FlowFMT* gets to the "true" cost found by ICLOCS2 on the other 18 trajectories in terms of cost ratio (i.e., $C_t(\sigma_{\text{FlowFMT*}})/C_t(\sigma_{\text{OCS}})$). As expected, the cost ratio approaches one when the number of samples increases. Notice that, due to the local constant flow assumptions, FlowFMT* does not necessarily finds trajectories with a cost that is higher than the OCS solutions.

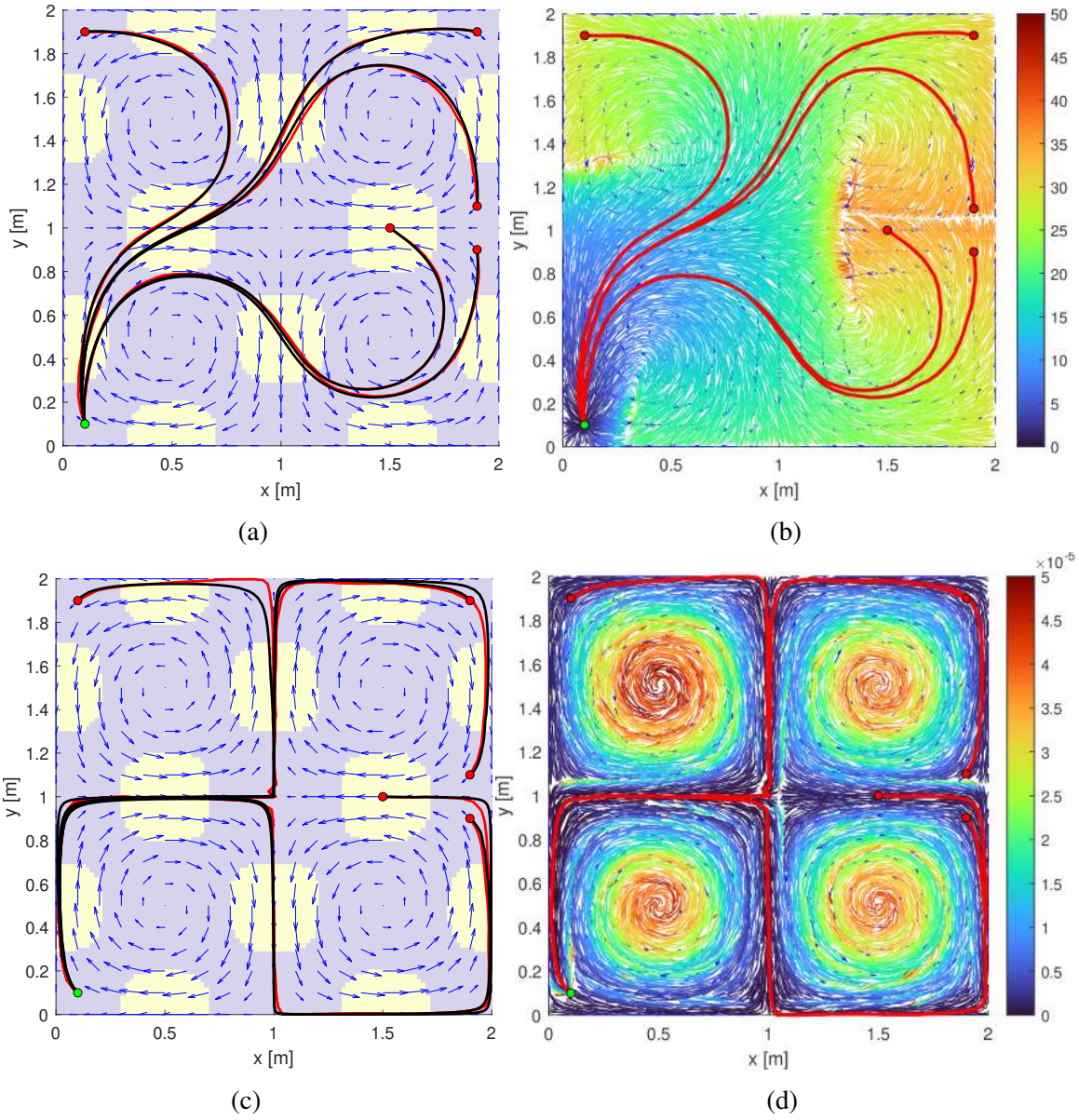
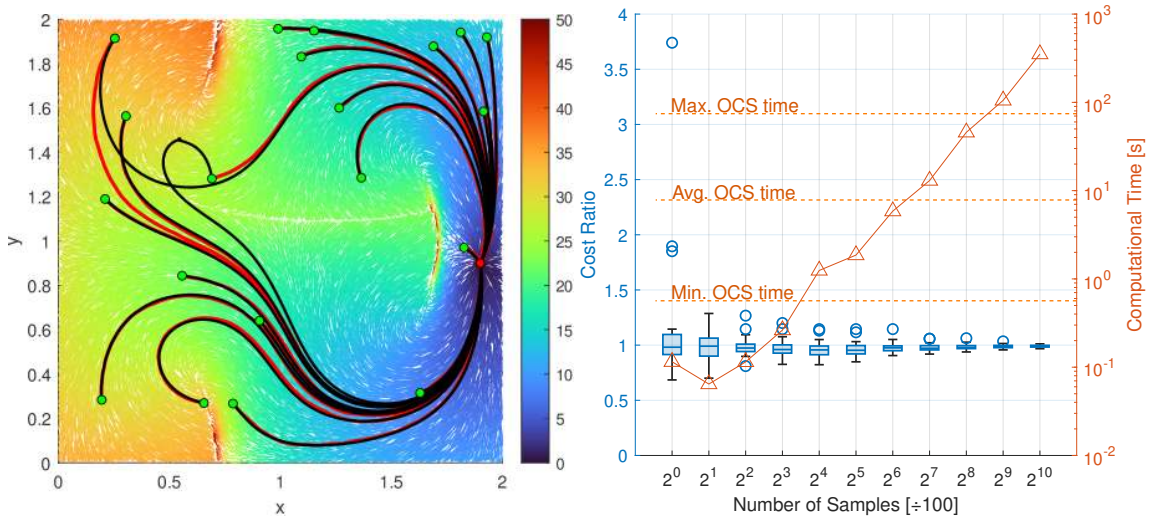


Figure 3.5: Paths planned by FlowFMT* (—) in a two-dimensional double-gyre flow (→) compared to the associated optimal control problem (—). On the left, we show the regions where the speed of the flow is greater (■) and smaller (■) than the maximum speed of the vehicle relative to the flow. We show FlowFMT* trees on the right, with edges colored by their costs. The minimum-time cost function was used in (a) and (b), and the minimum-energy cost function in (c) and (d).

However, as the number of samples increases, the assumption becomes more valid and the cost estimate becomes more realistic. Figure 3.6(b) also shows how the computational time to compute the policy increases as the number of samples increases. Despite that,



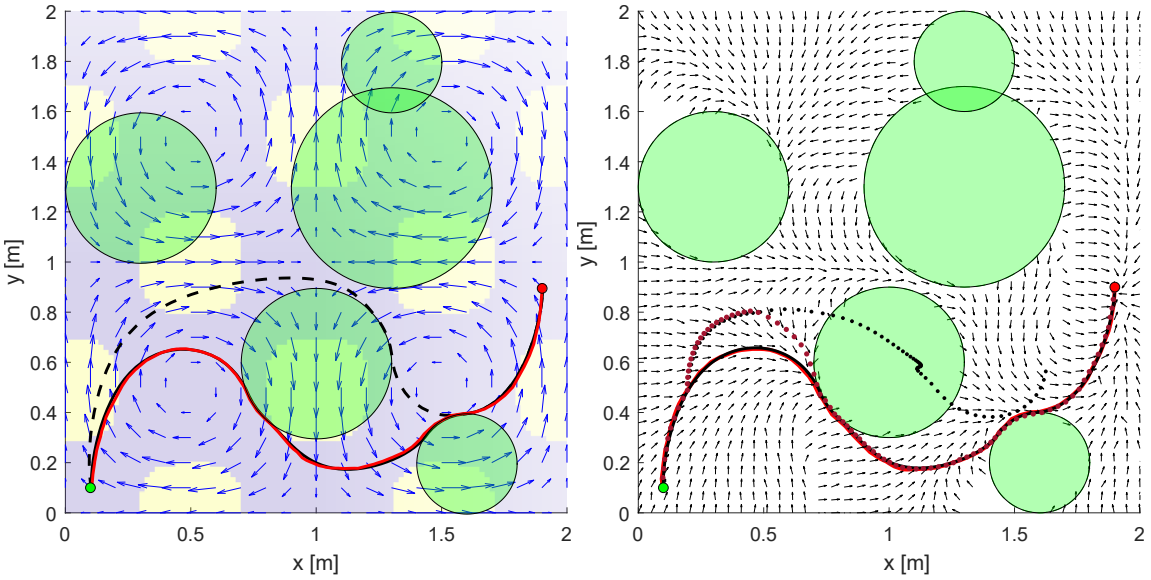
(a) (b)

Figure 3.6: Policy-FlowFMT* computed over the 2D double-gyre flow environment. In (a) we compare the solution for 20 random initial positions using our method and ICLOCS2. In (b) we show that our solution approaches the optimal solution with more samples at the cost of more computational time.

remember that a policy is only computed once for each goal position, which can dilute this time among the number of future trajectories that need to be computed using such a policy.

The Policy-FlowFMT* algorithm can also compute a guiding velocity vector field, as shown in Figure 3.7(b) for an environment with prohibited regions. This figure also shows simulated trajectories that illustrate how the vector field can help recover the vehicle after an actuation fault, while the OCS solution may lead to a collision. We considered that the OCS solution and the vector field are used in real-time to control the vehicle.

We then simulated that vehicle lost power for $10\,\mathrm{s}$ starting at $t = 5\,\mathrm{s}$. Figure 3.7(b) shows that, during this period, the vehicle drifts with the flow due to the lack of actuation. When the actuation is recovered, the OCS solution is not valid anymore, leading the vehicle to a prohibited region. In the best-case scenario, if a trajectory tracker were available, the vehicle may be able to retake the previously computed path at the cost of suboptimality of the resultant trajectory. In comparison, the vector field, no matter where the control is regained, would guide the vehicle through an optimal path from that position to the goal, as can be verified in Figure 3.7(b).



(a) (b)

Figure 3.7: Comparison with optimal control: (a) Comparison of FlowFMT* and optimal control in a two-dimensional double-gyre flow with prohibited regions (■). Both algorithms provide similar solutions when optimal control has a good initial guess (—). However, optimal control with a bad initial guess (– –) can return in paths with a different homotopy. (b) Comparison of Policy-FlowFMT* and optimal control for real-time control in the presence of a disturbance. After an actuation fault, the vector field (→) created using Policy-FlowFMT* brings the vehicle to the goal following an optimal trajectory (· · ·) while optimal control (· · ·) leads the vehicle to cross a prohibited region.

3.4.2 Three-Dimensional Environment

Figure 3.8 shows the vehicle's trajectories from start, $\vec{x}_{start} = [0.1,\ 0.1,\ 0.1]^{\mathsf{T}}$, to goal, $\vec{x}_{goal} = [1.9,\ 0.9,\ 0.9]^{\mathsf{T}}$, moving in a 3D double-gyre flow (as defined in (3.12)) with prohibited regions. We compared the performance of the optimal trajectories obtained by Policy-FlowFMT* using 102,400 samples and OCS (ICLOCS2). The OCS trajectory resulted in a time cost of $C_t(\sigma) = 28.18\,\mathrm{s}$. The cost of the trajectory obtained directly from the tree built using Policy-FlowFMT* is $C_t(\sigma) = 28.91\,\mathrm{s}$. By integrating the vector field resultant from the tree generated by Policy-FlowFMT* with a $0.1\,\mathrm{s}$ time step, we obtained a trajectory that reaches the goal in $29.4\,\mathrm{s}$.

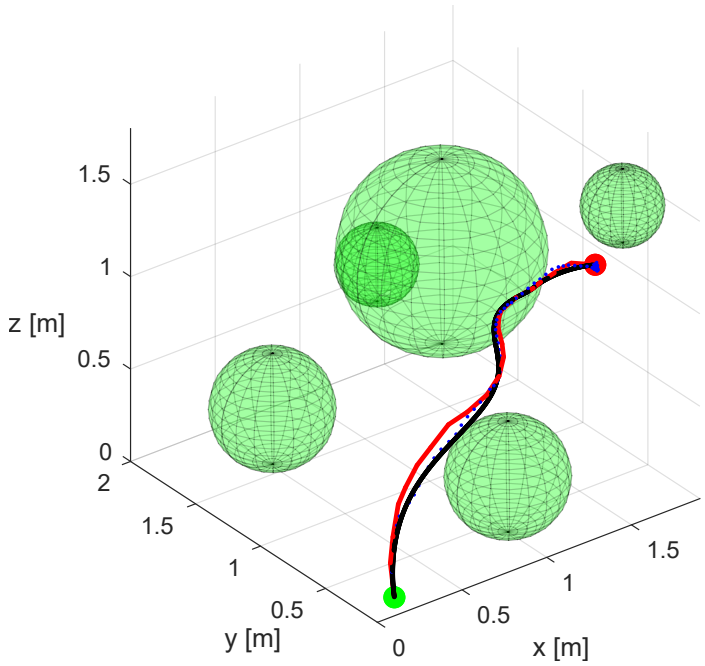


Figure 3.8: Navigating in a 3D double-gyre flow with prohibited regions. The Policy-FlowFMT* solution (—) using 102,400 samples is compared to the solution (· · ·) obtained from a simulation using the policy vector field (→), and the solution obtained from an optimal control problem solver (—).

3.5 Summary

This chapter presented a method for finding optimal paths and policies for holonomic vehicles moving in the presence of strong currents. We modified FMT* by defining new neighborhood sets that consider the directionality of the vehicle's movement due to the flow. The original FMT* algorithm, which relies on ball-shaped neighborhood sets, could eventually be used to solve our problem at the expense of testing the feasibility of connecting a node of the tree to all the samples in the neighborhood set before computing the connecting cost. Our method avoids this test and reduces the number of calls to the cost function by defining smaller, fluid-based neighborhood sets. These sets are defined using a reachability cone that restricts the space reached when strong currents are found (i.e., the flow speed is greater than the vehicle's relative speed to the flow). The reachability cone is also used to define a minimum-energy cost function for this type of environment.

Additionally, we provide simple modifications to our original path-planning algorithm

to calculate global policies to a goal. By using these methods, we can obtain paths with costs similar to those obtained using optimal control techniques without significantly increasing the computation time of the solution. Moreover, the proposed methods highly simplify the introduction of obstacles and other constraints, guaranteeing that the optimal homotopy is found when the number of samples is high enough. Finally, we show through a simulation that the policy generated by our method is inherently more resilient to actuation failures than optimal control. Therefore, if the environment contains obstacles and replanning is not desirable, the methods in this chapter should be preferred over previous solutions.

In the next chapter, we extend the methods proposed in this chapter to allow the presence of regions in the workspace where the vehicle cannot be actuated. This will be useful for solving the problem of navigating an aerobot in the atmosphere of Venus, where the vehicle does not have thrusting power during the night. To solve this problem, we propose new cost functions considering the vehicle's energy budget. On the other hand, Chapter 5 presents a method that can be used to navigate the vector-field policy developed here when unmodeled obstacles are present.

Chapter 4

Aerobots Flying Under the Strong Winds in the Atmosphere of Venus

This chapter proposes a methodology for the motion planning of a low-speed aerial robot (aerobot) navigating the atmosphere of Venus, where wind speeds can be much larger than the aerobot's speed relative to the wind. Part of this chapter was published as a conference paper at the 2022 International Conference on Unmanned Aircraft Systems (ICUAS) [59]. The most recent results, which complete this chapter, will be submitted as another paper to the IEEE Transactions of Aerospace and Electronics Systems (TAES).

4.1 Introduction

Venus, the third brightest object in our sky after the Sun and the Moon, has intrigued humankind for millennia. Comparisons to our planet are unavoidable because, in many ways, Venus is very similar to Earth. Together with Mars, these three planets are the ones that can potentially be in the habitable zone (Goldilocks Zone) of our Solar System. It is also a rocky planet with a similar size (95% of Earth's volumetric mean radius) and mass (81.5% of Earth's mass). However, its environmental conditions could not be more different than ours. On its surface, the conditions are incredibly hostile. The surface has an average tem-

perature of $464\,^{\circ}\mathrm{C}$, air pressure of $92\,\mathrm{bar}$, and density of $65\,\mathrm{kg\,m^{-3}}$ [67], making it nearly impossible to sustain life and be explored, due to the complexity of designing machines that can withstand this extreme environment even for a short duration.

Temperature and pressure fall when we go up on the Venusian atmosphere. From 50 to $70\,\mathrm{km}$ altitude, the pressure ranges from 1.066 to $0.0369\,\mathrm{bar}$, the temperatures from 76.85 to $-43.15\,^{\circ}\mathrm{C}$, and the air density from 1.594 to $0.084\,\mathrm{km\,m^{-3}}$ [68], which are similar to the environmental conditions found on Earth. However, other factors make it hard to explore the atmosphere of Venus. Most of Venus' atmosphere is composed of carbon dioxide (96.5%), but between 45 and $70\,\mathrm{km}$ of altitude, there is a layer of clouds composed of drops of sulfuric acid – a corrosive substance that can damage pieces of equipment very quickly. This cloud layer traps the heat in the inner atmosphere, maintaining the high temperature and pressure on the surface. Although not very well understood, it is proposed that the convection generated by these gradients of temperature and pressure generate what are known as superrotation winds – zonal winds that can largely exceed $100\,\mathrm{m\,s^{-1}}$ [69].

Despite these challenges, there is a potential for many discoveries and groundbreaking science coming from exploring this planet. Venus's atmosphere experienced a transformation due to the greenhouse effect. Thus, understanding what happened to Venus in the past can give insight into what can happen on Earth as greenhouse gases accumulate in our atmosphere. Recent scientific observations have found evidence of phosphine [70] in the cloud layer of the Venusian atmosphere. Along with other reasons [71], this leads some scientists to believe that microbial life can exist in the atmosphere of Venus. Hence, a renovated interest in sending missions to our neighboring planet's atmosphere has grown [56], and new missions have been announced [72].

4.1.1 Venus *In-Situ* Exploration History

The exploration of the Venusian atmosphere was first accomplished with Soviet atmospheric probes Venera 4, 5, and 6, which consisted of probes that entered the atmosphere

and parachuted to the surface, sampling the atmospheric composition along the descent. The main findings included the measurement of a high percentage of carbon dioxide on the atmospheric composition, low magnetic field, and confirmation of extreme temperatures and pressures [73]. Later, with the VeGa missions in 1983 and 1984, balloons were designed to float at approximately $54\,\mathrm{km}$ from the surface [74]. These balloons functioned for over two days, sending back data about the atmosphere. These balloons not only ascertained the existence of high-speed zonal winds but also verified the presence of vertical winds.

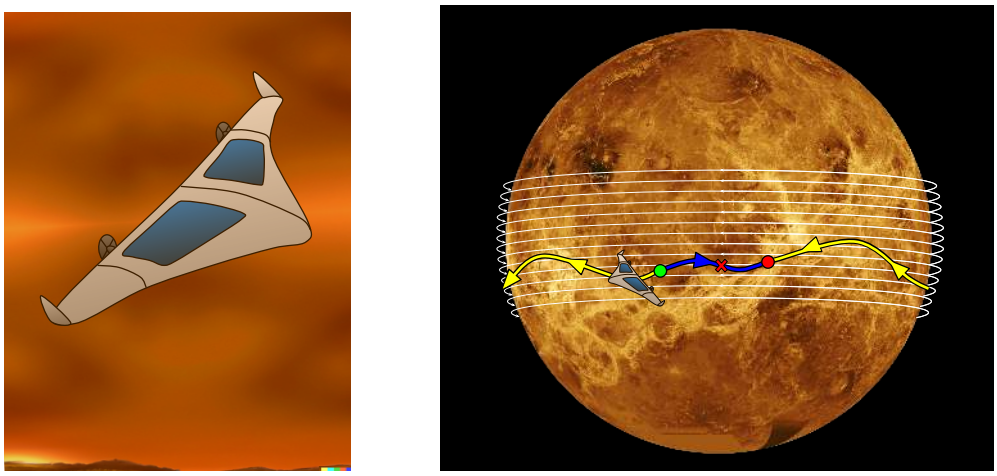
Over the years, NASA also sent a few missions to Venus. The first NASA mission was a flyby with Mariner 10 in 1973, which took over 4,000 photos of Venus and was able to identify general patterns of the atmospheric circulation from them [75]. During NASA's Pioneer program [76, 77], two Pioneer missions were dedicated to Venus exploration in 1978: Pioneer Venus 1 was an orbiter that used radar to map the surface of the planet, measured a very weak magnetic field, and confirmed that sulfuric acid was present in the cloud layer; Pioneer Venus 2 consisted of a set of probes that were sent to land on the surface, probing the atmosphere on their descent. NASA's Magellan Orbiter [78] was a space probe launched in 1989 to map the planet's surface and measure its gravitational field into a much higher resolution. More recently, in 2005, the European Space Agency (ESA) launched Venus Express [79] to study the interaction of the planet's upper atmosphere with solar winds and lower atmosphere with the surface. In 2010, the Japan Aerospace Exploration Agency (JAXA) launched a space probe called the Venus Climate Orbiter, also known as *Akatsuki*, to study the weather patterns on Venus, characterize its thick clouds, and search for volcanic activity.

A few missions have been scheduled to explore Venus in the next few years. NASA's DAVINCI (Deep Atmosphere Venus Investigation of Noble Gases, Chemistry, and Imaging) [80], scheduled for 2029, consists of an orbiter and a descent probe. The probe, which will work for approximately an hour, will be the first to enter Venus's atmosphere since

1985. It will take measurements of the lower atmosphere and acquire high-resolution images of the surface. NASA's VERITAS (Venus Emissivity, Radio Science, InSAR, Topography, and Spectroscopy) [81] is a spacecraft that will orbit Venus. It will try to answer the question "What caused Earth and Venus to diverge down different evolutionary paths?" by providing radar maps of the surface at a higher resolution than the ones returned by the Magellan missions in the 1990s and describing the topography and surface rock composition. These missions will either acquire short-time measurements of the atmosphere or deployment of instruments from a distance.

Innovative approaches and conceptual vehicle designs for exploring Venus have been proposed for *in-situ* exploration of Venus [82] for more extended periods. These include solar airplanes (heavier-than-air) and aerobots (lighter-than-air). Solar airplanes, as proposed in [83, 84, 85], could be deployed over the cloud layer where atmospheric pressure is similar to Earth and the exoatmospheric solar intensity (upwards and downwards) is sufficiently high, allowing the top and bottom surfaces to generate power continuously for the aircraft through solar panels installed in its wings and matching the power required by the propulsive system. The main advantage of this approach is that if the aircraft can counteract the strong winds, it would have continuous solar power by avoiding the night and would not need a large energy storage capacity. The main disadvantage of this approach is the higher complexity of transporting and deploying such a vehicle.

For these reasons, lighter-than-air aerobots have been the preferred concept for an aerial platform on Venus. Aerobots can leverage their buoyancy to transport heavier loads, cover a large area with lower power requirements, and extend the mission's duration, significantly increasing the science these missions could achieve. Three types of aerobots were considered to explore the atmosphere of Venus according to VEXAG Roadmap [56]: 1) fixed altitude balloons, 2) variable altitude balloons, and 3) airships. Most of the work developed by NASA recently is related to the first two types of aerobots due to their simplicity and higher technology readiness levels (TRL) [86, 87]. However, in this work, we focus on



(a) Venus Aerobot (b) Venus Circumnavigation

Figure 4.1: Venus *In-Situ* Aerial Platform navigation problem. In (a), we illustrate an airship concept inspired by the VAMP [88] (background generated with assistance of DALL·E 2). In (b), the solution of a motion planning problem in the atmosphere of Venus can result in the most efficient path to a desired goal being very different from the shortest path when considering the vehicle dynamic constraints and environmental properties such as high-speed winds and available solar power. The green and red dots represent the start and goal locations, respectively. The yellow path is more energetically efficient than the blue path, which is shorter. The thin arrows represent the superrotation winds. Image adapted from mosaics showing a global view of the surface of Venus from Magellan synthetic aperture radar image, centered at 180° East longitude. Credit: NASA Goddard Space Flight Center

the airship concept. Specifically, we consider a hybrid airship concept similar to the Venus Atmospheric Maneuverable Platform (VAMP), a semi-buoyant inflatable plane proposed in [88] for Venus exploration. Figure 4.1a shows an illustration, made by the author, of an aerobot inspired by the VAMP.

Just like solar airplanes, airships provide the possibility of 3D-controlled flight. However, even with the reasonable climate conditions of higher altitudes, the atmosphere of Venus still imposes challenges on this type of fixed-wing aircraft. Due to the high-speed zonal winds, a vehicle left to drift in the atmosphere will circulate the planet in as few as five Earth days. Unless the vehicle is powerful enough to counteract these wind speeds, it is expected to experience long periods without solar power, making it dependent on charged batteries for controlled flight during these periods [89]. These challenges introduce the problem of obtaining the optimal paths for an aircraft navigating these environments. As

illustrated in Figure 4.1b, the energy-optimal path in such an environment may differ significantly from the shortest path. For this concept, we want to design motion planners that: 1) create minimum energy paths, since the vehicles have limited battery and will fly for long periods; 2) account for battery re-charging; and 3) account for the strong winds in which the aircraft will fly.

4.1.2 Organization

This chapter is organized as follows. A compilation of related work is presented in Section 4.2. Problem definitions are listed in Section 4.3. Some background on the problem and the framework developed in our first attempt to solve the problem are provided in Section 4.4. Adaptations were required to use our previously introduced motion planner (FlowFMT*) to solve this problem; they are explained in detail in Section 4.5. Numerical results for the validation of the method and the complete problem are presented in Section 4.7. Finally, a summary of the work is presented in Section 4.8.

4.2 Related Work

A flight controller for station-keeping of a stratospheric balloon navigating a wind field on Earth was developed using reinforcement learning by [90]. The method leads to a solution where the balloon stays close to the station by changing altitude and seeking favorable winds. Variable altitude light gas balloons were assessed in [91] for their use in Venus' and Titan's atmospheres. Models and simulations were introduced for the balloon's energy consumption and dynamics equations.

A probabilistic motion planner was developed for balloons in strong and uncertain winds for Montgolfieré balloons in the atmosphere of Titan [92], a moon of Saturn. A planner for altitude-controlled balloons navigating the atmosphere of Venus was developed in [58]. The motivation for their work is to monitor volcanic activity using these aerobots.

The authors propose a method for autonomous guidance of these vehicles considering uncertainty in the wind field. The balloons can control their buoyancy to change their altitude in the atmosphere and take advantage of favorable winds. The problem is framed as a continuous-state Markov Decision Process (MDP), and an optimal policy is calculated to solve the MDP problem with approximate dynamic programming. This framework was derived from the author's previous work on buoyancy-controlled underwater vehicles' guidance in uncertain ocean currents under the ice shelves in Antarctica [54].

The most significant advantage of this type of solution is the straightforward incorporation of uncertainty in the motion models, which is relevant for robots moving in environmental flows (which are uncertain by nature). The solution is suitable to the conditions proposed by the authors since the vehicle has few options for actuation and a limited number of states. However, MDP formulations do not scale well with an increasing number of states and actions, a problem known as the "curse of dimensionality". As we are dealing with an aircraft with more degrees of freedom, we have opted for sampling-based motion planners that can better handle higher dimensional issues.

4.3 Problem Definitions

This section describes the problem we are dealing with in this chapter, especially the environment and the airship.

Environment Models

We targeted a region of the atmosphere of Venus within $50\,\mathrm{km}$ to $70\,\mathrm{km}$ altitude, where exploration is viable. We simplified the atmospheric properties as temperature, $\Theta(h)$, pressure, $p(h)$, air density, $\rho(h)$, and wind field, $\mathbf{w}(\lambda, \varphi, h)$, where λ and φ represent latitude and longitude respectively and h represents the altitude. Figure 4.2 shows the density, temperature, and pressure as a function of the altitude. The data was obtained from tables

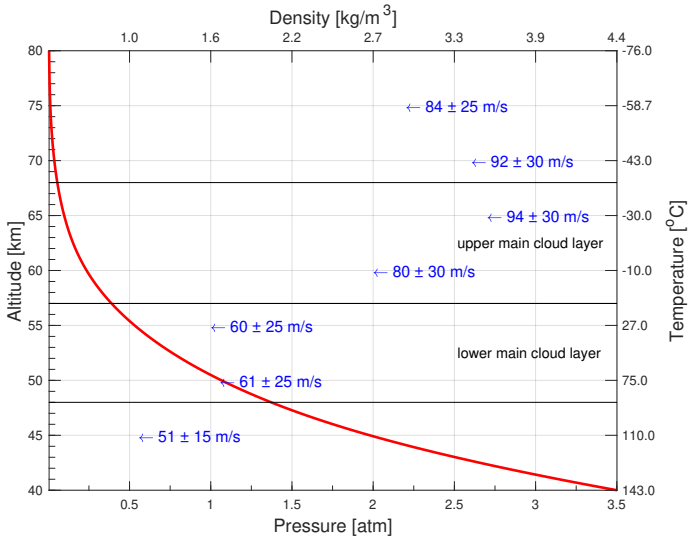


Figure 4.2: Atmospheric characteristics of Venus. The wind is illustrated by the blue left arrows, where each value represents the average wind speed and its possible variation in function of the altitude.

presented in [68], compiled from data from several past missions to Venus [93]. The wind field is dominated by the superrotation winds, which is compatible with the observations near the equatorial region [69]. Wind and cloud layers are also displayed according to the altitude. The farther we get from the Venusian surface, the lower density, temperature, and pressure we get. Wind speeds vary from around $61 \pm 25\,\mathrm{m\,s^{-1}}$ at $50\,\mathrm{km}$ altitude to $92 \pm 30\,\mathrm{m\,s^{-1}}$ at $70\,\mathrm{km}$ altitude. The gravitational acceleration, $g(h)$, is approximately $8.87\,\mathrm{m/s^2}$, changing only slightly with the altitude. For simplicity, all properties are assumed to be time-invariant. Furthermore, we disregard the retrograde rotation of Venus since the duration of a day in Venus is equivalent to 243 Earth days, and the winds can circulate the atmosphere in as few as five Earth days.

Another factor considered in our model is the availability of solar intensity ($I_{solar}(z)$) to charge the batteries and power the aircraft. Venus is exposed to a solar flux of $2600\,\mathrm{W/m^2}$ (this is also called exoatmospheric solar flux), almost double the solar flux that Earth is exposed to. However, this flux is attenuated by the atmosphere. The available solar intensity (ratio between available flux and the exoatmospheric) is 95% at the top of the clouds (at $65\,\mathrm{km}$ altitude) and 20 to 50% at the bottom of the cloud layer (at $40\,\mathrm{km}$ altitude), de-

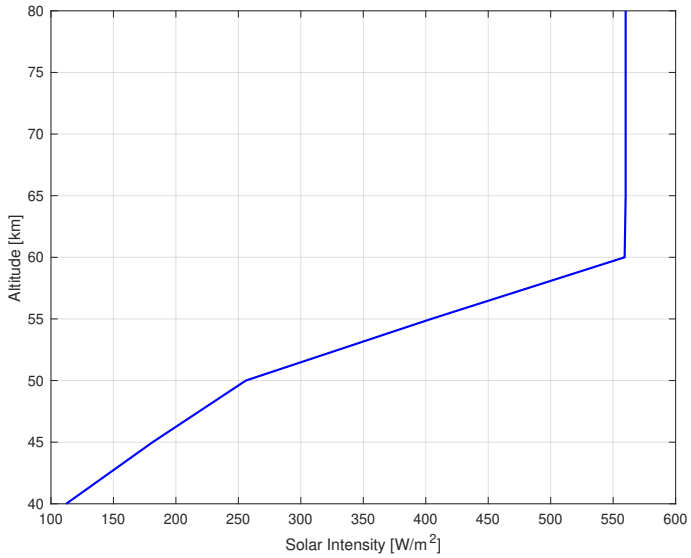
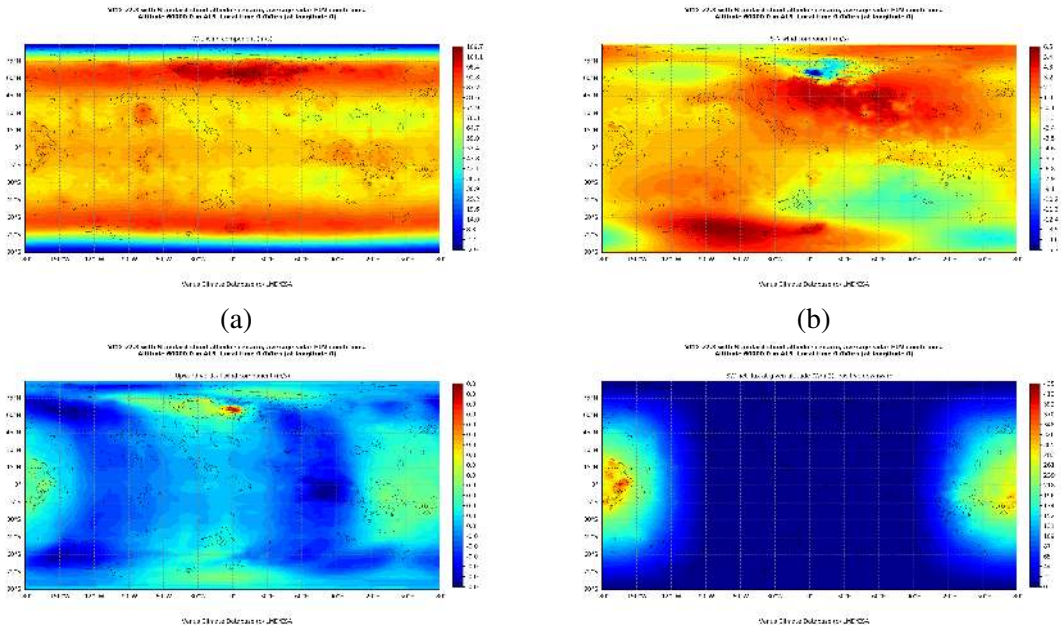


Figure 4.3: Venus solar intensity as a function of altitude. The graph also considers the conversion efficiency of the solar panels.

pending on the considered wavelength. Further, solar panels have an intrinsic efficiency in converting solar power to stored energy in the batteries. A model for the solar intensity in Venus considering this energy conversion efficiency was obtained from [89] and is shown in Figure 4.3.

A more detailed model of the environment can obtained from a database of meteorological properties, the Venus Climate Database (VCD), derived from the state-of-the-art General Circulation Model (GCM) simulations of the Venusian atmosphere, developed by the Laboratoire de Météorologie Dynamique (LMD) and the Laboratoire Atmosphères, Observations Spatiales (LATMOS), members of the Institute Pierre Simon Laplace (IPSL) [5, 6, 7]. The GCM serves as a computational model that calculates the changes in a planet's atmospheric conditions over time. The model employed for statistical analysis has undergone rigorous validation, drawing upon existing observational data. Its primary goal is to offer a representation that aligns closely with the most up-to-date understanding of the state of the Venusian atmosphere, taking into account the observations made and the fundamental physical principles governing atmospheric circulation and surface conditions on the planet.



(a) (b)

(c) (d)

Figure 4.4: Venus Climate Database simulations can be used to get the values of the three wind components and the solar intensity on August 24, 2023. (a) Zonal winds (W-E) [$\mathrm{m\,s^{-1}}$]. (b) Meridional winds (S-N) [$\mathrm{m\,s^{-1}}$]. (c) Vertical winds [$\mathrm{m\,s^{-1}}$]. (d) SW net flux [$\mathrm{W\,m^{-2}}$]. Generated using Venus Climate Database © LMD/ESA [5, 6, 7].

Variables in the database are stored on the grid on which they are obtained from the general circulation model (GCM) runs: a regular, equispaced horizontal 96 $\times$ 97 in East longitude vs. latitude. Longitudes thus range from -176.25°to 180.0°in steps of 3.75°and latitudes from 90°to -90°in steps of 1.875°. A large number of variables is available, but for this work, we used: “Atmospheric density”, “Meridional wind” (South-North), “Zonal wind” (West-East), “Vertical wind” (down-up), “SW net flux” (positive downward) at a given altitude. All these are available in three spatial and time dimensions. Figure 4.4 shows an example of the data obtained from the database at $60\,\mathrm{km}$ of altitude for the three wind components and the solar intensity on August 24, 2023.

Airship Model

In this work, we are considering a vehicle based on the concept proposed by [88]. This vehicle is a semi-buoyant unmanned propelled aircraft. At $70\,\mathrm{km}$ above Venus’s surface,

Table 4.1: Vehicle Design Parameters

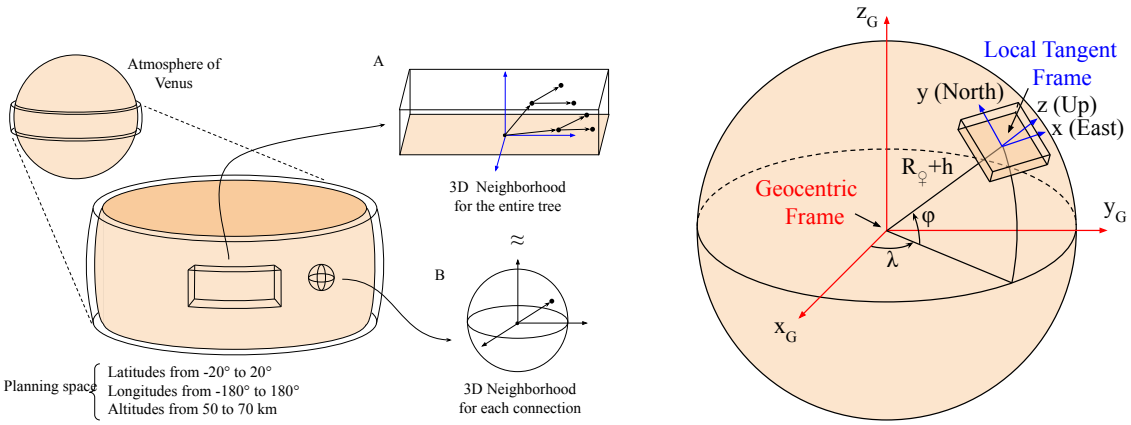
Parameters	Values	Parameters	Values
Aircraft Mass	$450\,\mathrm{kg}$	Solar Panel Area	$50\,\mathrm{m}^2$
Aircraft Volume	$490\,\mathrm{m}^3$	Battery Max. Energy	$50\,\mathrm{MJ}$
Wing Span	$50\,\mathrm{m}$	Propellers Efficiency	0.8
Mean Aerodynamic Chord	$10\,\mathrm{m}$	Shaft Efficiency	0.8
Wing Area	$500\,\mathrm{m}^2$	Airspeed Limits	0-$30\,\mathrm{m/s}$
Wing Aspect Ratio	5	Roll Angle Limits	$\pm 30^\circ$
Oswald Efficiency Number	0.85	Flight Path Angle Limits	$\pm 45^\circ$
Zero-lift Drag Coeff.	0.02		

the vehicle is expected to be 10% buoyant with 90% lift from the propellers. At low altitudes, the vehicle is 100% buoyant. The aircraft is solar-powered and its solar panels are distributed over its body. The maximum airspeed of the vehicle is $30\,\mathrm{m/s}$. Table 4.1 shows the main design parameters used in this chapter.

4.3.1 Motion Planning Problem

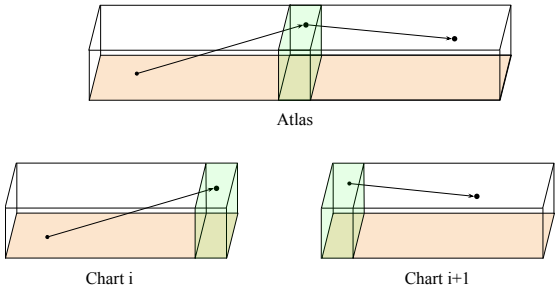
The goal of the motion planning problem is to find a feasible and optimal path that guides a hybrid airship from the initial configuration $\mathbf{x}_{start}$ to the final configuration $\mathbf{x}_{goal}$ in a finite time, subject to the dynamic equations given by $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w})$ and respecting the constraints in the configuration and input spaces. Solving the motion planning problem in a planet-size environment is not trivial. The topology of the environment in which the aerobot moves is not simple. Given constraints in latitude and altitude, the environment can be seen as a truncated spherical shell, like the one shown in Figure 4.5a.

This environment has very specific characteristics. The thickness (altitude component) of this shell is small compared to the other dimensions, making the problem almost 2D. Still, the gradient of the wind in this direction is the largest, so this dimension cannot be disregarded. Additionally, since the wind speeds are much higher than the vehicle's airspeed, the vehicle cannot move freely in all directions. Hence, depending on the start-goal pair, the vehicle must loop around the planet to find a feasible plan. This periodicity in



(a) Truncated spherical shell manifold

(b) Environment Frames



(c) Charts and atlas

Figure 4.5: Representation of the environment where the aerobot will navigate. The complete space is given by a truncated spherical shell, as shown in (a, top-left). The space is a manifold that can be portioned (a, right) into local subspaces like A and B that behave like Euclidean space. These spaces, charts, can be combined to form the complete space. The collection of charts is called an atlas.

space makes the motion planning problem harder. By considering that the environment is a "manifold", i.e., a topological space that locally behaves like an Euclidean space (Figure 4.5a, panels A and B), we can create smaller topological spaces, known as charts, that cover a small region of the manifold with a local coordinate system. Figure 4.5b shows the different reference frames involved in this problem: geocentric and local tangent frames. In our case, we are considering a three-dimensional manifold that, locally, behaves as $\mathbb{R}^3$. The charts are combined by an overlap region in which transition functions describe how to smoothly change coordinates from one chart to another. When a collection of charts that covers the entire manifold is combined, as shown in Figure 4.5c, then we have created an atlas.

In the following section, we share our first attempt at solving this problem. We decided to restrict our analysis to a coordinated chart in $\mathbb{R}^3$, with axes x (East), y (North), and z (Up) locally aligned respectively with longitude, latitude, and altitude. The kinematic model $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w})$ for the aircraft motion under the influence of the wind presented in [24] is extended to account for the buoyancy of the vehicle:

$$\begin{aligned}
\dot{x} &= v_a \cos\gamma \cos\psi + w_x \\
\dot{y} &= v_a \cos\gamma \sin\psi + w_y \\
\dot{z} &= v_a \sin\gamma + w_z \\
\dot{\psi} &= \frac{(m - \rho V) g \cos\gamma \tan\phi}{m v_a}.
\end{aligned} \tag{4.1}$$

where $\dot{\psi}$ is obtained from $L \sin\phi = m v_a \dot{\psi}$.

4.4 Background

In our earlier attempt to create a motion planner for this problem, we proposed a sampling-based path planner for a fixed-wing, hybrid airship that relies on Dubins' Airplane as a local planner and accounts for the influence of strong winds. The planner optimizes a cost function based on energetic transactions. This cost function includes not only expenditure – in the form of thrust or drag – but also accumulation – in the form of charging using solar panels and gains in potential energy (e.g., with upward directional winds). It is essential to mention that the naive inclusion of such influx energy could generate negative costs for some paths, which would violate the requirements for most optimizers [21]. We then used the notion of *opportunity cost* [94], the loss of gains caused by choosing not to follow an alternative that can offer the highest benefit, to avoid incorporating negative costs. In our case, when considering solar power, the cost is computed as the difference between the maximum charging energy the vehicle could gain (e.g. when flying above the clouds) and

the energy gained at a less favorable altitude. The methodology consisted of

- An energy-efficient sampling-based motion planning strategy that i) uses as local planners Dubins' Airplane trajectories deformed by the wind field, ii) includes a semi-buoyant dynamic model for the aircraft, iii) accounts for the vehicle's battery state.

- A cost function that i) accounts for the energy expenditure of the propulsive system and ii) considers battery charging as an opportunity cost, thus avoiding negative costs in the function.

4.4.1 Energy and Wind Aware RRT

Our sampling-based algorithm, the EWRRT, was inspired by the RRT* [95]. It builds and maintains a tree graph $\tau(V, E)$ using a set of vertices, V, and a set of edges, E. The algorithm samples a random point in the configuration space, finds its nearest vertex in the current tree, uses an approximated steering function to create a vertex based on the random point and its nearest vertex, and, finally, selects the actual parent for this vertex using a more appropriated steering function that also returns the cost of the path between a vertex and its potential parent.

One important difference from other comparable methods [20] is that, since we are not considering an environment with obstacles, we do not have strict requirements for the path and the vertices that are going to be added to the tree. Therefore, instead of forcing the aircraft to reach the selected vertex, we allow the wind to deform the trajectory created when connecting a new vertex and its potential parents. Unlike RRT* [95], we do not rewire the tree when a new node is inserted, which would not be straightforward due to wind. This fact may lead our approach to generate sub-optimal paths since rewiring is necessary to prove the asymptotically optimally of RRT*.

Our motion planner is shown in Algorithm 4, and its functions are described in the

Algorithm 4 Energy-efficient and wind-aware RRT planner

function τ = EW-RRT($\mathbf{x}_{start}, \mathbf{x}_{goal}, d, d_{min}$)
  $\mathrm{v}_{init}, \mathrm{v}_{goal} \leftarrow \mathbf{x}_{start}, \mathbf{x}_{goal}$
  $\tau \leftarrow InitializeTree()$
  $\tau \leftarrow InsertNode(\tau, \varnothing, \mathrm{v}_{init})$
  for $i \leftarrow 1, N$ **do**
    $\mathrm{v}_{rand} \leftarrow Sample()$
    $\mathrm{v}_{nearest} \leftarrow Nearest(\tau, \mathrm{v}_{new})$
    $\mathrm{v}_{new} \leftarrow ApproxSteer(\mathrm{v}_{nearest}, \mathrm{v}_{rand}, d)$
    $V_{near} \leftarrow Near(\tau, \mathrm{v}_{new})$
    $\mathrm{v}_{min}, \mathrm{v}_{new} \leftarrow BestParent(V_{near}, \mathrm{v}_{nearest}, \mathrm{v}_{new})$
    $\tau \leftarrow InsertNode(\tau, \mathrm{v}_{min}, \mathrm{v}_{new})$
    if $Distance(\mathrm{v}_{new}, \mathrm{v}_{goal}) < d_{min}$ **then**
      return
    end if
  end for
end function

sequence.

Initialization, Sampling, Nearest Neighbor and Steering

The method is initialized with an empty graph using the function $InitializeTree$. After the initial vertex is added to the tree, for a given number of iterations N, the $Sample$ function randomly chooses a configuration $\mathbf{x}_{rand} \in \mathbb{C}$ from the configuration space and creates a vertex v_{rand} with it. Given the vertex v_{rand} and the tree $\tau(V, E)$, the $Nearest$ function returns the nearest vertex $\mathrm{v}_{nearest} \in V$ according to a distance function. A simple Euclidean metric is used to select $\mathrm{v}_{nearest}$ to keep a low computational load. The $ApproxSteer$ function creates a new vertex v_{new} in the direction given by the nearest vertex and the vertex randomly sampled. This new vertex is created at a distance given by the step size d. The heading angle is set as aligned with this direction.

Given the new vertex v_{new} and the tree $\tau(V, E)$, the $Near$ function returns a set the vertices $V_{near} \in V$ that are at a certain distance l from the new vertex. The Euclidean distance is used as a metric to obtain this set. The threshold l is calculated using the optimal formulation proposed in [21]. The function $BestParent$, shown in Algorithm 5, selects the

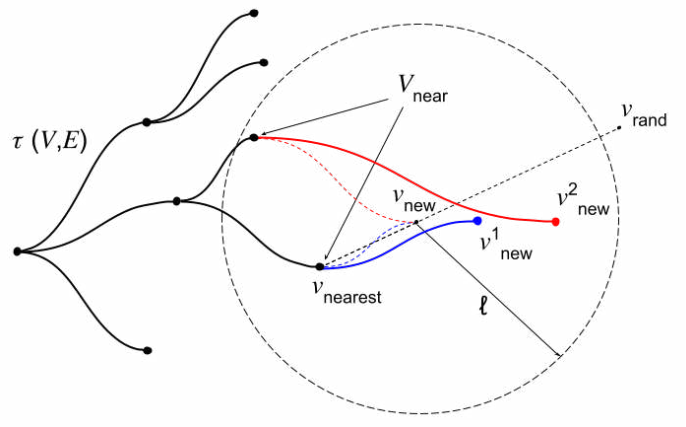


Figure 4.6: Energy- and Wind-Aware RRT connections. The new vertex v_{new} is connected using Dubins' Airplane to all vertices in set V_{near} (dashed colored curves). These curves are deformed due to the effect of the wind (solid-colored curves). The cost for each of the wind-deformed curves is calculated and ranked. The vertex v^i_{new} with the lowest cost is added to the tree $\tau(V, E)$.

Algorithm 5 Choosing the best parent for a new vertex

function $\mathrm{v}_{min}, \mathrm{v}_{new} = \textsc{BestParent}(V_{near}, \mathrm{v}_{nrst.}, \mathrm{v}_{new})$
  $\mathrm{v}_{new}, c_{min} \leftarrow DubinsSteer(\mathrm{v}_{nrst.}, \mathrm{v}_{new})$
  for $\mathrm{v}_{near} \in V_{near}$ **do**
    $\mathrm{v}'_{new}, c' \leftarrow DubinsSteer(\mathrm{v}_{near}, \mathrm{v}_{new})$
    if $c' < c_{min}$ **then**
      $\mathrm{v}_{new} \leftarrow \mathrm{v}'_{new}$
      $c_{min} \leftarrow c'$
    end if
  end for
end function

best parent vertex from the set of near vertices V_{near} by comparing the costs of steering from each of them to the new vertex, v_{new}, using the $DubinsSteer$ function (Algorithm 6). Figure 4.6 illustrates how selecting the best parent process works.

Steering With Dubins

The local planner $DubinsSteer$, shown in Algorithm 6, calculates a path connecting a given vertex v_{near} to v_{new} using an extended 3D version of the Dubins' paths. As described in [96] we consider that the airspeed, v_a, is constant and that the roll angle ϕ can only assume three values $\{\phi_{min}, 0, \phi_{max}\}$ (in other words, the aircraft is capable of turning in

Algorithm 6 Steering using Dubins' Airplane and wind

function $\mathrm{v}_{new}, c_{min} = \textsc{DubinsSteer}(\mathrm{v}_{near}, \mathrm{v}_{new})$
  $\sigma \leftarrow DubinsPath(\mathrm{v}_{near}, \mathrm{v}_{new})$
  $S_{wind} \leftarrow DeformTrajectory(\sigma, N, v_a)$
  $\mathrm{v}_{new} \leftarrow S_{wind}(end)$
  $T \leftarrow RequiredThrust(S_{wind})$
  $c_{min} \leftarrow Cost(\mathrm{v}_{near}, \sigma_{wind}, T)$
end function

Algorithm 7 Deforming Dubins' trajectory due to wind

function $\sigma_{wind} = \textsc{DeformTrajectory}(\sigma, N, v_a)$
  $L \leftarrow GetLength(\sigma)$
  $\Delta t \leftarrow (L/N)/v_a$
  $S_{wind} \leftarrow Discretize(\sigma, \Delta t)$
  $D \leftarrow (0, 0, 0)$
  for $i \leftarrow 1, N$ **do**
    $S_{wind}(i) \leftarrow S_{wind}(i) + D$
    $(w_x, w_y, w_z) \leftarrow WindField(\sigma(i))$
    $D \leftarrow D + (w_x, w_y, w_z) \cdot \Delta t$
  end for
end function

maximum rate or follow a straight heading angle), and the flight path angle, γ, can assume any value within its limits $[\gamma_{min}, \gamma_{max}]$. Therefore, $DubinsPath$ finds the sequence of inputs $u = [v_a,\ \gamma,\ \phi]^T$ that takes the system from v_{near} to v_{new} in the wind reference frame, i.e., without considering the effect of the wind.

Based on the method presented in [20], the function $DeformTrajectory$ (Algorithm 7) discretizes the Dubins' path at regular time intervals Δt. Then, the loop deforms the trajectory from the wind reference frame to the ground frame by adding the integrated wind drift at each time step using a $WindField$ function to obtain a wind vector for each ground-relative position. This process is illustrated in Figure 4.7.

The new trajectory, now in the ground reference frame, is used to calculate the required thrust at every instant ($RequiredThrust$) using the formulation presented in the following section. With the necessary thrust and the ground-relative trajectory, the cost for this potential edge is calculated using the $Cost$ function, explained in the next section. Figure 4.6

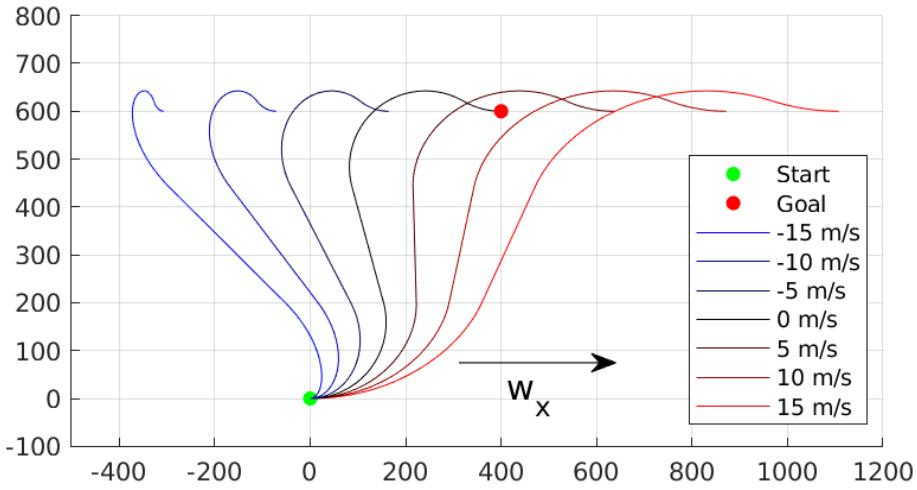


Figure 4.7: A trajectory is deformed due to the effect of wind. The black trajectory is obtained by connecting the start and goal configurations using a Dubins' Airplane path and discretizing this path over time. Using this discretization, we calculate the red and blue trajectories by adding the drift caused by the wind field at each time step.

shows how this step is used in our algorithm.

Energy-Based Cost Function

For this planner, we proposed an energy-based cost function that considers the energy expenditure due to the propulsive system and a heuristic to account for the changes in potential energy and energy accumulation due to battery charging with the solar panels. The proposed cost function is given by

$$C_E(\sigma) = \begin{cases} E_{prop} + E_{pot}^{opp} + E_{solar}^{opp}, & \text{if } b + \Delta b \geq 0 \\ \infty, & \text{if } b + \Delta b < 0 \end{cases}. \tag{4.2}$$

The first component, E_{prop}, is straightforward since there is a natural relationship between energy expenditure and cost. The cost is given by the energy obtained by the integration of the instantaneous power required by the propellers:

$$E_{prop} = \int \frac{P_{prop}}{\eta_{prop}\eta_{shaft}}\, dt = \int \frac{T v_a}{\eta_{prop}\eta_{shaft}}\, dt, \tag{4.3}$$

where T is the thrust calculated for each point in the path, and η_{prop} and η_{shaft} are the propellers and shaft efficiencies, respectively. We derived the equations that describe the force equilibrium during the aircraft's flight to find the thrust. The hypothesis for this work was that the aircraft is capable of two coordinated flight modes. Given a desired flight path angle, the first mode is flying in a straight line, and the second is turning with a constant bank angle, both with respect to the wind reference frame. Similarly to what is proposed in [24], we considered that switching between steady turn and steady climb/descent is fast compared to the duration of the command itself and that a point-mass model is sufficient to represent the vehicle. Further, we assume that the aircraft maintains its airspeed constant for this work. Figure 4.8 shows a free-body diagram of a point-mass model of the aircraft where W is the weight of the aircraft, B is the buoyancy force, L is the lift force, D is the drag force, T is the thrust force, α is the angle of attack of the aircraft with respect to the airflow (represented by v_a), γ is the flight path angle, ψ is the heading angle, and ϕ is the roll angle.

For our planning purposes, we must obtain an expression for the thrust required at any given point of the aircraft trajectory. By equating the forces parallel and perpendicular to the vehicle's airspeed, we have

$$(W - B)\cos\gamma = L\cos\phi + T\sin\alpha \tag{4.4}$$

$$(W - B)\sin\gamma = T - D\cos\alpha. \tag{4.5}$$

The term $(W - B)$ can also be written as $(m - \rho V)g$, where m is the mass of the aircraft, ρ is the density of the fluid, V is the volume of the aircraft, and g is the acceleration of gravity. Both density and gravity are variable with the altitude of the vehicle. Considering that the angle of attack is usually small, we assume the approximations $T\sin\alpha \approx 0$ and $T\cos\alpha \approx T$. Given a specific input $\mathbf{u} = [v_a, \gamma, \phi]^T$ and the wing area (S), we can

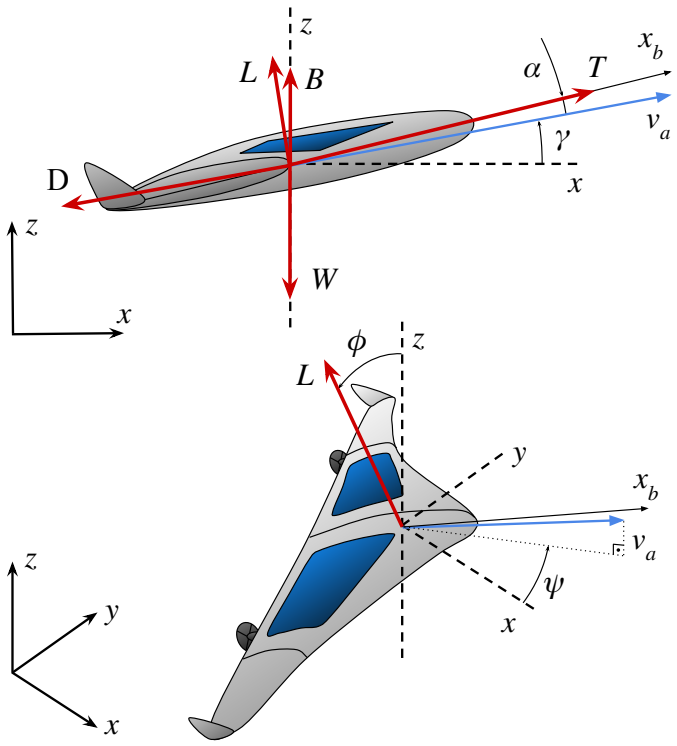


Figure 4.8: Coordinate frames and free-body diagram of the vehicle. The flight path, heading, and roll angles (γ, ψ, ϕ) are given with respect to the inertial frame (x, y, z). The thrust force is assumed to be aligned with x-axis of the vehicle's body frame, denoted by x_b.

rewrite (4.4) and obtain the lift force and lift coefficient as

$$L = \frac{(m - \rho V) g \cos\gamma}{\cos\phi} \Rightarrow c_L = \frac{L}{\frac{1}{2}\rho v_a^2 S}. \tag{4.6}$$

We represent the drag coefficient with a second-order polynomial using the lift coefficient, the wing aspect ratio ($\mathcal{R}$), and the Oswald efficiency number (e). This allows us to calculate the drag force exerted on the vehicle:

$$c_D = \sum_{i=0}^{N} a_i c_L^i = c_{D,0} + \frac{c_L^2}{\pi \mathcal{R} e} \Rightarrow D = \frac{1}{2}\rho v_a^2 S\, c_D. \tag{4.7}$$

Finally, by rewriting (4.5) and including our model for the drag force from (4.7), we

obtain the required thrust as

$$\begin{aligned} T &= D + (W - B)\sin\gamma \Leftrightarrow \\ T &= \frac{1}{2}\rho v_a^2 S c_D + (m - \rho V) g \sin\gamma. \end{aligned} \tag{4.8}$$

For the other two components of (4.2), the inclusion of energy gains with solar energy and potential energy is problematic because, if the energy influx is greater than the energy outflux, negative costs can be encountered, which is unsuitable for optimization. Since there are paths that can maximize the value for both energy components (paths with the maximum altitude), we propose their inclusion as a complement to their respective maximum value. Like in Microeconomics, this can be thought of as opportunity costs: the loss of a prospective gain when one chooses not to use the action that maximizes its benefits [94]. Thus, the opportunity costs for the potential and solar energies are

$$\begin{aligned} E_{pot}^{opp} &= E_{pot}^{max} - E_{pot} = mg(h_{max} - h) \quad \text{and} \\ E_{solar}^{opp} &= E_{solar}^{max} - E_{solar} = \int (P_{solar}^{max} - P_{solar})\, dt, \end{aligned} \tag{4.9}$$

where h_{max} is $70\,\mathrm{km}$, h is the height of the local goal, and $P_{solar} = I_{solar}\eta_{s.p.}A_{s.p.}$ is the solar power accounting for the available solar intensity, I_{solar}, solar panel efficiency, $\eta_{s.p.}$, and solar panel area, $A_{s.p.}$. In the atmosphere of Venus, P_{solar}^{max} is the maximum power that the solar panels can absorb, $560\,\mathrm{W/m^2}$ (reached at the highest altitude, as shown in Figure 4.3).

The change in the battery level Δb in 4.2 is calculated using the difference between influx and outflux energies as

$$\Delta b = -E_{prop} + E_{solar}. \tag{4.10}$$

Node Insertion and Termination

Given the current tree, $\tau(V, E)$, a vertex $\mathrm{v}_{min} \in V$, and a new vertex, v_{new}, the $InsertNode$ function adds the new vertex, v_{new}, to V and adds an edge $(\mathrm{v}_{min}, \mathrm{v}_{new})$ to E. It also assigns the cost of this new edge (local trajectory) and adds this cost to the cost of the parent vertex to create a new total cost associated with the new vertex. Similarly, the child (v_{new}) updates the state of the battery by adding the change in the battery level required to follow the added trajectory.

After the new vertex, v_{new}, is added to the tree $\tau(V, E)$, we check if it is in the goal region, $X_{goal} = \{\mathbf{x}_{new} : Distance(\mathrm{v}_{new}, \mathrm{v}_{goal}) < d_{min}\}$, specified by d_{min}, which is a hyper-parameter to control how close the solution needs to get from the actual desired position. The algorithm is terminated if the condition is met, and the tree $\tau(V, E)$ is returned. Otherwise, the loop is continued until the maximum number of samples N is reached.

4.4.2 Findings

We defined the position of the aircraft as given by $\mathbf{p} = [x,\, y,\, z]^T$, where x, y, and z represent, respectively, the Cartesian coordinates in the local chart aligned with longitude, latitude, and altitude. The heading angle of the vehicle is given by ψ. The configuration vector is then given by $\mathbf{x} = [x,\, y,\, z,\, \psi]^T$. We also keep a separate state, b, to account for the vehicle's battery level. We defined an input vector $\mathbf{u} = [v_a,\, \gamma,\, \phi]^T$ that includes the airspeed, v_a, flight path angle, γ, and roll angle, ϕ, of the vehicle. The wind field experienced by the aircraft $\mathbf{w} = [w_x,\, w_y,\, w_z]^T$ varies with its position as it moves through the wind field.

For the analysis presented in this work, we were more concerned with checking the behavior of our planner in a smaller environment, in the order of some hundreds of kilometers (Table 4.2). In our first example, shown in Figure 4.9, we show the relevance of considering high-speed winds in this motion planning problem. In this experiment, we set the airship's airspeed to its maximum value $v_a = 30\,\mathrm{m/s}$ and then used a wind model

Table 4.2: Configuration Space Limits

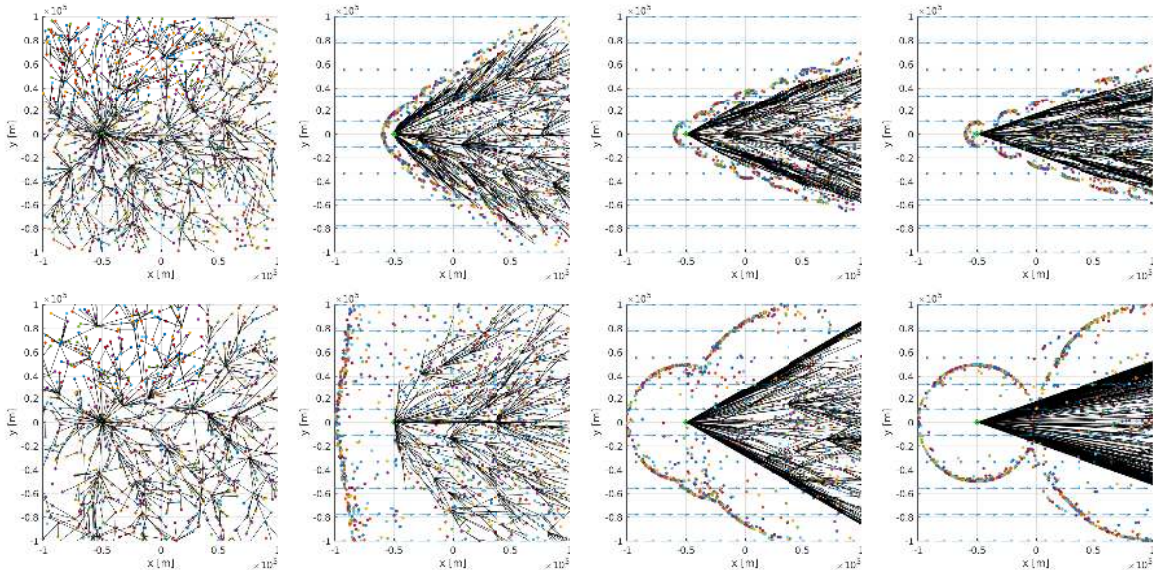
Parameters	Values	Parameters	Values
x_{min}	-100 km	x_{max}	100/300 km
y_{min}	-100 km	y_{max}	100 km
z_{min}	55 km	z_{max}	70 km
ψ_{min}	-π rad	ψ_{max}	π rad
b_{min}	0 MJ	b_{max}	50 MJ

$\mathbf{w} = [w_x,\ 0,\ 0]^T$, varying the values of the longitudinal component w_x. The motion planner runs using $N = 1000$ random samples (represented by the colored dots in the figure). For each value of w_x, we also compared the effect of two different steering step sizes $d = 10$ km and $d = 50$ km.

In Figure 4.9(a) we can see a scenario with $w_x = 0$. Since there is no wind, the Dubins' paths are not deformed, and the random samples coincide with the vertices of the tree. Furthermore, because $v_a < ||w||$, the vehicle can move in any direction. From Figure 4.9(b) to Figure 4.9(d), we increase the wind speed and observe that the reachable configurations become limited by a "cone"-like geometry (similar to what is described in [14]).

In Figure 4.9(b), the wind speed is equal to the airspeed of the vehicle, and we can see how the step size can be an important parameter. In an ideal scenario, the vehicle could maintain its x coordinate at most. However, if it chooses to move in the other axes, it is carried by the wind. When a small step size is chosen, the curved parts of the Dubins' trajectories are relatively large compared to its straight part and the cone. If the step size is big, the straight parts are prevalent, and the airship can move in the y coordinate with low drift in the x coordinate. As the wind speeds become much larger than the airspeed, the vehicle cannot counteract the wind speeds that much, and the reachable regions are further reduced. This is an essential fact for motion planning in Venus. If the distance to our desired goal in the y direction (latitude) is too big and the vehicle cannot produce speeds to counteract the wind drift, the only possible way to reach it is to circle the planet.

In our second simulation, shown in Figure 4.10, we present a solution for a well-



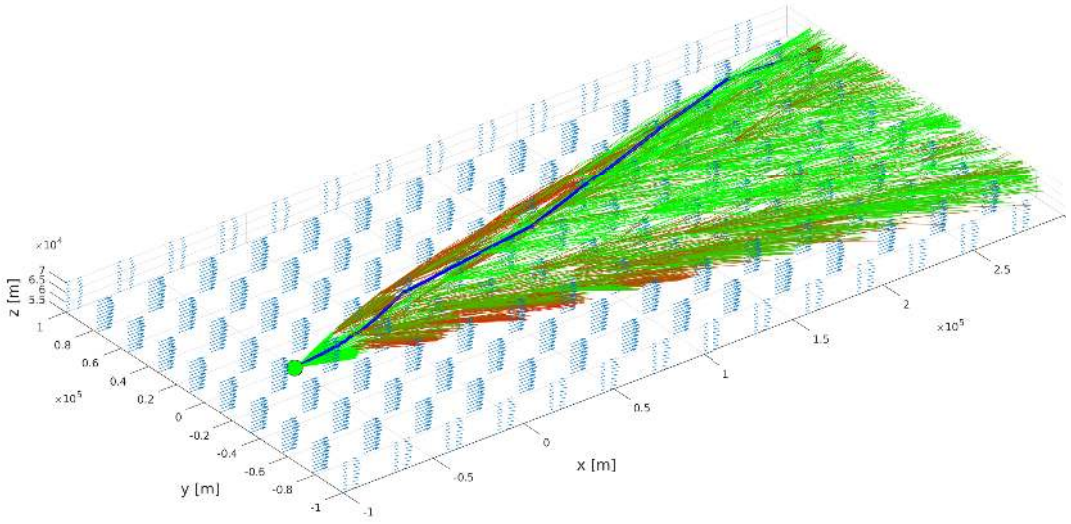
(a) $w_x = 0\,\mathrm{m/s}$ (b) $w_x = 30\,\mathrm{m/s}$ (c) $w_x = 60\,\mathrm{m/s}$ (d) $w_x = 90\,\mathrm{m/s}$

Figure 4.9: Effect of the winds in the planning tree. Airspeed $v_a = 30\ \mathrm{m/s}$. Steering step-size $d = 10\ \mathrm{km}$ in the top figures and $d = 50\ \mathrm{km}$ in the bottom figures.

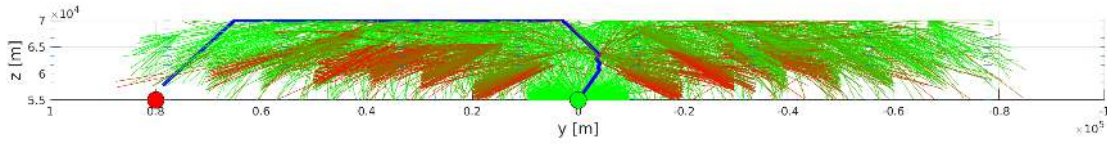
defined motion planning problem. The start position of the aircraft was set to $\mathbf{p}_{start} = [-100,\, 0,\, 55]^T\,\mathrm{km}$ with initial heading set to zero. The goal position was set to $\mathbf{p}_{goal} = [300,\, 80,\, 55]^T\ \mathrm{km}$. The final heading was also set to zero, but our planner does not guarantee this heading since the planner finds paths to a region around the target. The aircraft airspeed was set to be $v_a = 30\,\mathrm{m/s}$. The 3D wind model used in this setup was again simplified to $\mathbf{w} = [w_x,\, 0,\, 0]^T$, but now w_x was computed in function of the altitude of the vehicle, as shown in Figure 4.2.

The motion planner runs with $N = 5000$ random samples and a steering step size of $10\,\mathrm{km}$. In the path the planner finds, the vehicle travels most of the route at higher altitudes. This is expected since, in higher altitudes, we find lower air density (i.e., less drag force) and higher availability of solar intensity. A solution was reached at a distance of $3490.9\,\mathrm{m}$ from the goal, where the vehicle would arrive with a full battery ($50\,\mathrm{MJ}$), having traveled a distance of $361.35\,\mathrm{km}$ in $4065.0\,\mathrm{s}$. The cost of the path is $221.9\,\mathrm{MJ}$.

For comparison, we connected the start and goal configurations with a single Dubins path using an iterative method to account for the wind drift. This solution converged to a



(a) 3D view



(b) YZ view

Figure 4.10: Motion planning for airship flying in the atmosphere of Venus with start and goal positions given by the green circle and red circles. A tree of kinematically feasible trajectories is created considering the effect of the wind drift. Higher altitudes are preferred by the solution trajectory (in blue) because, at these altitudes, the aircraft finds less air resistance and the highest solar energy for charging its batteries. The colored edges on the tree show the battery capacity after the vehicle follows the edge: the color changes from green to red according to the battery level (green is full, and red is empty).

position at a distance of $1920.2\,\mathrm{m}$ from the goal. The traveled distance would be $360.84\,\mathrm{km}$ with total time $3405.4\,\mathrm{s}$. However, the path would not be energetically feasible because the battery would fully discharge before reaching the end ($-E^{baseline}_{prop} + E^{baseline}_{solar} = -598.4\,\mathrm{MJ}$, which is more than the battery capacity $b_{max} = 50\,\mathrm{MJ}$).

4.4.3 Limitations

The simulation results shown in the previous subsection demonstrate a few limitations of the EWRRT. Although our motion planner can find low-energy trajectories and is an excellent first step for a more complete system that will be used in future exploration missions,

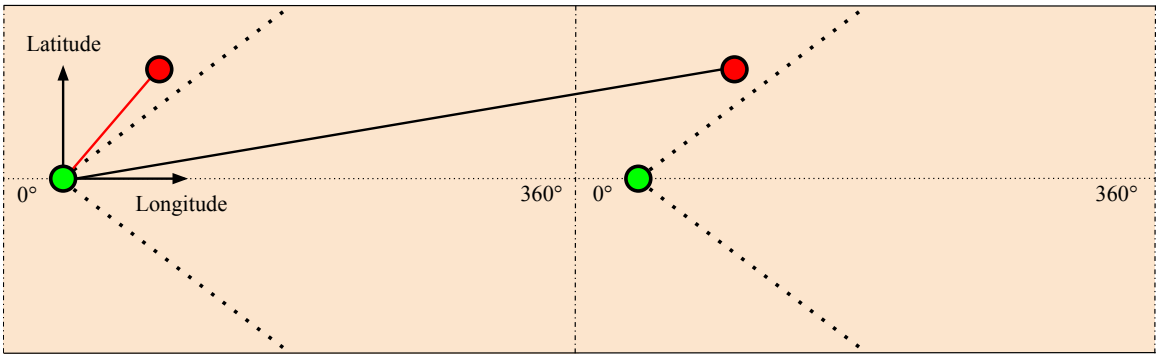


Figure 4.11: The superrotation winds of the Venusian atmosphere create a reachability issue even without considering other issues like running out of battery. Depending on whether or not the goal (●) is inside of the reachable region from the start (●), the aerobot is forced to go around the planet to solve the motion planning problem.

when the aerobot is limited to airspeeds smaller than the wind speed, planning must account for the wrapping nature of the atmospheric environment. If a goal is set against the wind or at higher latitudes, the aircraft must go around the planet at least once to reach it. Figure 4.11 illustrates this issue. The goal (●) is outside of the region (defined with dotted lines) that is immediately reachable from the start (●). Thus, the aerobot is forced to go around the planet to solve the motion planning problem.

Also, EWRRT did not consider the possibility of motion without thrust (i.e., gliding) when the battery is empty since we do not expand the search tree using edges that would result in a negative battery level. However, when considering the problem at the planet scale, there is a large region where solar power is unavailable, and this behavior becomes relevant. Additionally, our current simulations consider the wind field as time-invariant and primarily unidirectional. We want to consider a more complete model of Venus's atmospheric wind that curves with the planet.

4.5 Extending FlowFMT* to Non-Actuation Regions

Due to the limitations of the previous approach, we abandoned the EWRRT in favor of the FlowFMT* because, with a few adaptations, the latter could return optimal paths and

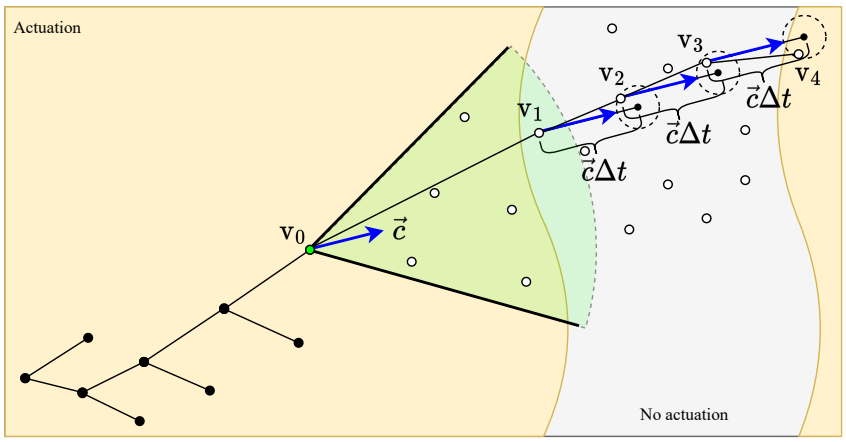


Figure 4.12: Adaptation of the FlowFMT* to regions without actuation. Once the vertex v_1 is connected to the rest of the tree through v_0, a function that checks if the vehicle is active at v_1 returns that it is not active. The vertex is moved to the "closed" set, and the vehicle's state is advanced in time (by Δt). At each step in time, the nearest vertex to the position found through integration is found (in this case, v_2), and a connection is made. This process is repeated until the vehicle reaches active condition or a maximum number of time steps are executed.

overcome some of the limitations mentioned. In this section, we develop tools that permit extending the FlowFMT* algorithm to achieve our final objective: obtain a motion planner capable of finding optimal trajectories in a planet-sized environment.

Under environmental currents, imposing restrictions on where the vehicle can be actuated may make the motion planning problem harder to solve. To achieve the desired behavior, the FlowFMT* algorithm presented in Algorithm 1 was adapted to solve situations where the vehicle does not actuate. In summary, the algorithm is changed so that when a node is connected to the tree and its battery state is below a certain threshold level, it is moved to the closed set, and vertices are sequentially connected to this vertex, trying to expand the tree until actuation is possible again, as shown in Figure 4.12.

During the non-actuating period, the vehicle behaves similarly to a pseudo-Lagrangian drifter that tracks the fluid through the atmosphere. A full explanation of the assumptions necessary to consider a balloon a Lagrangian drifter can be seen in [97]. For our Venus problem, once actuation is lost, the vehicle cannot generate lift and sustain its altitude, and

gravity, buoyancy, and other aerodynamic forces interact in its descent to the altitude where it is neutrally buoyant. Assuming we have a model for the evolution of the vehicle's state after drifting with the fluid, the algorithm looks for the nearest sample in the unvisited set. Presumably, the nearest state found in the samples differs from the drifted state, but it is added to the tree as an approximation. This approximation improves as the overall number of samples increases because the probability of finding samples closer and closer to the drifted state increases. The cost function and battery state are also updated for this new connection. Algorithm 8 shows the required changes.

Next, we validate the performance of this motion planner against a well-understood and benchmark scenario, the crossing of a jet stream. Variations of this problem are described in [98, 99, 100, 101]. By starting with a problem with known solutions and characteristics, we can evaluate the planner's accuracy, efficiency, and adaptability while having a clear reference point for comparison. This validation helps ensure that the results for the Venus navigation problem are reliable.

Validation Setup

The crossing of a canonical uniform jet problem, as shown in Figure 4.13a, is an illustrative problem that brings possibilities to validate FlowFMT* on different aspects. The problem statement from the last chapter is used again, but we consider a different environment and vehicle. We consider a vehicle that can move at a maximum speed $v_r^{max} = 10.0\,\mathrm{m\,s^{-1}}$ relative to the flow. We limit the environment to a box of side $100\,\mathrm{m}$, i.e., $\mathcal{W} = [0, 100] \times [0, 100]$ (the dimensional unit 'meters' is omitted in the rest of this chapter for simplicity). The flow environment is modeled as

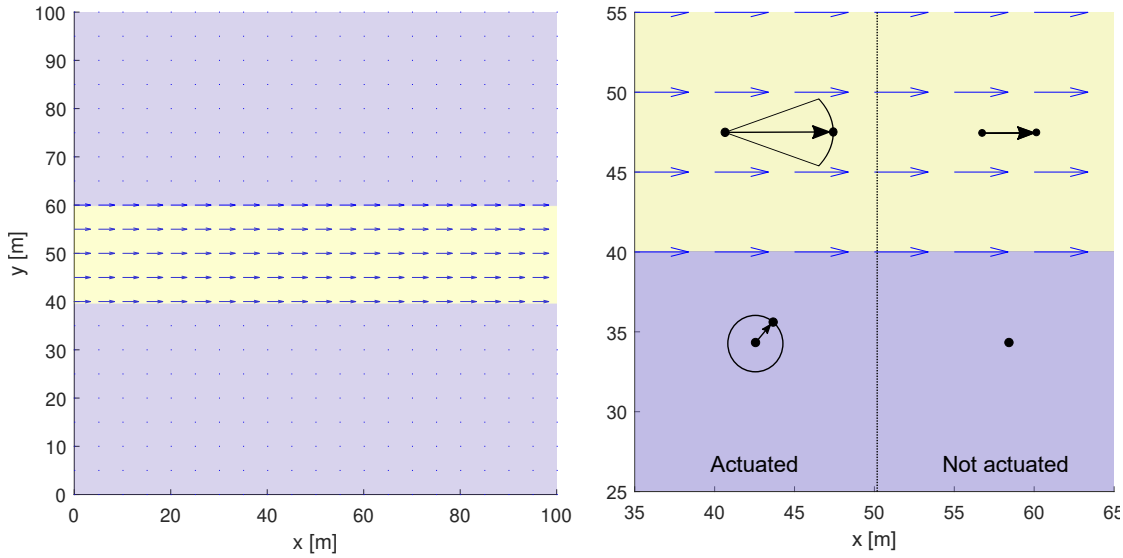
$$\vec{c}(x, y) = \begin{cases} [20.0, 0.0]^\mathsf{T} \text{ if } y \geq 40 \text{ and } y \leq 60\,, \\ [0.0, 0.0]^\mathsf{T} \text{ otherwise}\,. \end{cases} \tag{4.11}$$

Algorithm 8 Flow-Aware FMT* Algorithm for Non-Actuation Regions

function $\sigma = \text{FLOWFMT*}(\mathrm{x}_S, \mathrm{x}_G, V_{pool})$
  $V \leftarrow \mathrm{x}_S \cup V_{pool} \cup \mathrm{x}_G$, $E \leftarrow \varnothing$
  $V_{unvisited} \leftarrow V \backslash \{\mathrm{x}_S\}$, $V_{open} \leftarrow \mathrm{x}_S$, $V_{closed} \leftarrow \varnothing$
  $\mathrm{z} \leftarrow \mathrm{x}_S$
  $N_{\mathrm{z}}^P \leftarrow PosteriorNeighborhood(\varnothing, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$, $N^P \leftarrow Save(\varnothing, N_{\mathrm{z}}^P, \mathrm{z})$
  $N_{\mathrm{z}}^A \leftarrow AnteriorNeighborhood(\varnothing, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$, $N^A \leftarrow Save(\varnothing, N_{\mathrm{z}}^A, \mathrm{z})$
  while $\mathrm{z} \neq \mathrm{x}_G$ **do**
    $V_{open,new} \leftarrow \varnothing$
    $X_{near} = N_{\mathrm{z}}^P \cap V_{unvisited}$
    for $\mathrm{x} \in X_{near}$ **do**
      $N_{\mathrm{x}}^A \leftarrow AnteriorN'd(N^A, V \backslash \{\mathrm{x}\}, \mathrm{x}, r_n)$, $N^A \leftarrow Save(N^A, N_{\mathrm{x}}^A, \mathrm{x})$
      $Y_{near} \leftarrow N_{\mathrm{x}}^A \cap V_{open}$
      $\mathrm{y}_{min} \leftarrow \text{argmin}_{\mathrm{y} \in Y_{near}} \{c(\mathrm{y}) + Cost(\mathrm{y}, \mathrm{x})\}$
      if $CollisionFree(\mathrm{y}_{min}, \mathrm{x})$ **then**
        $E \leftarrow E \cup \{(\mathrm{y}_{min}, \mathrm{x})\}$
        $V_{unvisited} \leftarrow V_{unvisited} \cap \{\mathrm{x}\}$
        $c(\mathrm{x}) = c(\mathrm{y}_{min}) + Cost(\mathrm{y}_{min}, \mathrm{x})$
        if $Active(\mathrm{x})$ **then**
          $V_{open,new} \leftarrow V_{open,new} \cup \{\mathrm{x}\}$
        else
          $\mathrm{x}_{new} \leftarrow \mathrm{x}$
          for $1 : N_{ts}$ **do**
            $V_{closed} \leftarrow V_{closed} \cup \{\mathrm{x}_{new}\}$
            $\mathrm{x}_{drift} \leftarrow Nearest(Drift(\mathrm{x}_{new}), V_{unvisited})$
            if $\mathrm{x}_{drift} = \varnothing$ **then break**
            end if
            $E \leftarrow E \cup \{(\mathrm{x}_{new}, \mathrm{x}_{drift})\}$
            $V_{unvisited} \leftarrow V_{unvisited} \cap \{\mathrm{x}_{drift}\}$
            $c(\mathrm{x}_{drift}) = c(\mathrm{x}_{new}) + Cost(\mathrm{x}_{new}, \mathrm{x}_{drift})$
            if not $Inactive(\mathrm{x}_{drift})$ **then**
              $V_{open,new} \leftarrow V_{open,new} \cup \{\mathrm{x}_{drift}\}$
              break
            end if
            $\mathrm{x}_{new} \leftarrow \mathrm{x}_{drift}$
          end for
        end if
      end if
    end for
    $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \backslash \{\mathrm{z}\}$
    $V_{closed} \leftarrow V_{closed} \cup \{\mathrm{z}\}$
    if $V_{open} = \varnothing$ **then return** $\varnothing$
    end if
    $\mathrm{z} \leftarrow \text{argmin}_{\mathrm{y} \in V_{open}} \{c(\mathrm{y})\}$
    $N_{\mathrm{z}}^P \leftarrow PosteriorN'd(N^P, V \backslash \{\mathrm{z}\}, \mathrm{z}, r_n)$, $N^P \leftarrow Save(N^P, N_{\mathrm{z}}^P, \mathrm{z})$
  end while
  return $GetPath(\mathrm{x}_G, T(V_{open} \cup V_{closed}, E))$
end function

Validation Results

To match what is found in the literature, in this section, we plan paths from the start position $\vec{x}_{start} = [20.0, 20.0]^\mathsf{T}$ to the goal positions $\vec{x}_{goal} = [80.0, 80.0]^\mathsf{T}$. The results are compared



(a) Jet Crossing

(b) Reachability constraints

Figure 4.13: Reachability cases for motion planning problem of crossing a uniform jet. In (a), we illustrate the validation scenario, the regions where the speed of the flow is greater (■) and smaller (■) than the maximum speed of the vehicle relative to the flow. In (b), on the left, we show the reachable space for a regular holonomic vehicle, as seen in Figure 3.2. On the right, we show the different reachability conditions imposed by the flow or lack of actuation. If there is a flowing environment, the vehicle is carried by it. If not, the vehicle is stagnant.

to the optimal results found using the methodology explained in [101]. This reference shows how to formulate an optimization problem to use a simple gradient-based method to find the optimal solution to the jet crossing problem. The optimization is solved using Matlab (*fmincon* function). All the simulations were performed using an Intel® Core™ i9-9900K CPU at 3.6 GHz, with 16 cores and 32GB of RAM.

In Figure 4.14, we show the optimal solution obtained numerically using the gradient method with cost $C_t(\sigma) = 6.2523\,\mathrm{s}$. A very accurate solution was found using FlowFMT* using 409,600 samples. We obtained an optimal path with cost $C_t(\sigma) = 6.2569\,\mathrm{s}$ in $30.7315\,\mathrm{min}$ of computation time. The method found the solution after 277,597 iterations and 11,650,852 calls to the cost function.

For the regions where the vehicle does not actuate, we need to integrate the flow during a time interval Δt_{drift}. To find reasonable values for these two parameters, we defined a

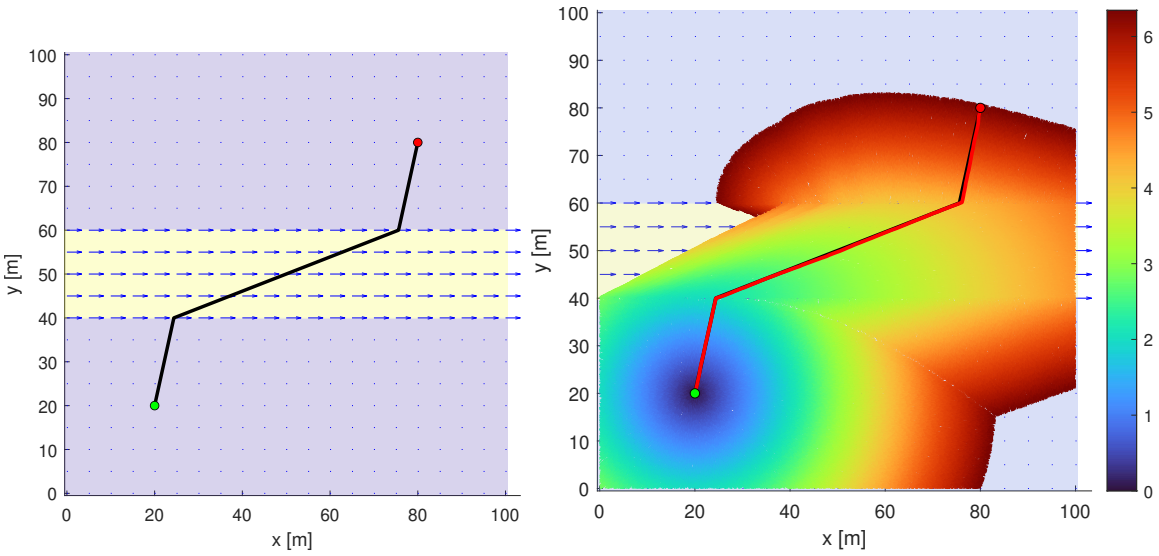


Figure 4.14: Motion planning for a vehicle crossing a jet flow including a region with no actuation. The robot moves from the start position (●) to the goal (●). We show the regions where the speed of the flow is greater (■) and smaller (■) than the maximum speed of the vehicle relative to the flow. The tree is colored based on the cost of the arrival vertices. The numerical optimal solution is shown in black (—), and the solution found with the adapted FlowFMT* is shown in red (—). The small error caused by the discontinuity causes a shift in the solution, but it does not affect the cost significantly.

characteristic length L_c as the size of the environment (in this scenario, $L_c = 100$). We use the maximum flow speed (c_{max}) to find the characteristic time as $T_c = L_c/c_{max}$. We obtain a reference drifting time interval, $\Delta t^*_{drift} = T_c/N_{ts}$, where N_{ts} is the number of maximum time steps. One problem of this scenario is that it has discontinuities in the flow speeds, which contradicts our assumption that in the limit, as the sample density gets higher, the flow speed between two neighbor samples tends to be constant. Although increasing the number of samples minimizes the effect of the discontinuity on the entire path, we also included an adaptive neighborhood radius according to a metric based on the difference between the flow velocities in the start and goal vertices ($\vec{c}_{parent}$ and $\vec{c}_{child}$). The adaptive neighborhood radius is given by

$$r_n^* = r_n \left(1 - \frac{\|\vec{c}_{parent} - \vec{c}_{child}\|}{2c_{max}}\right), \tag{4.12}$$

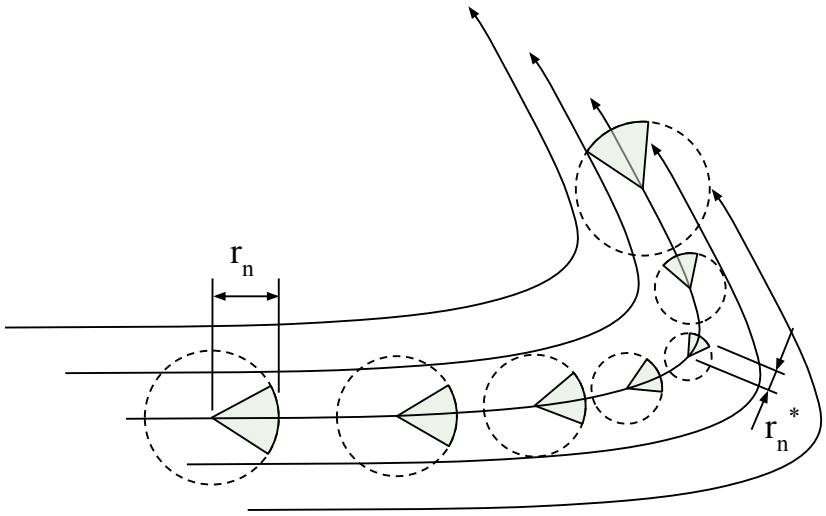
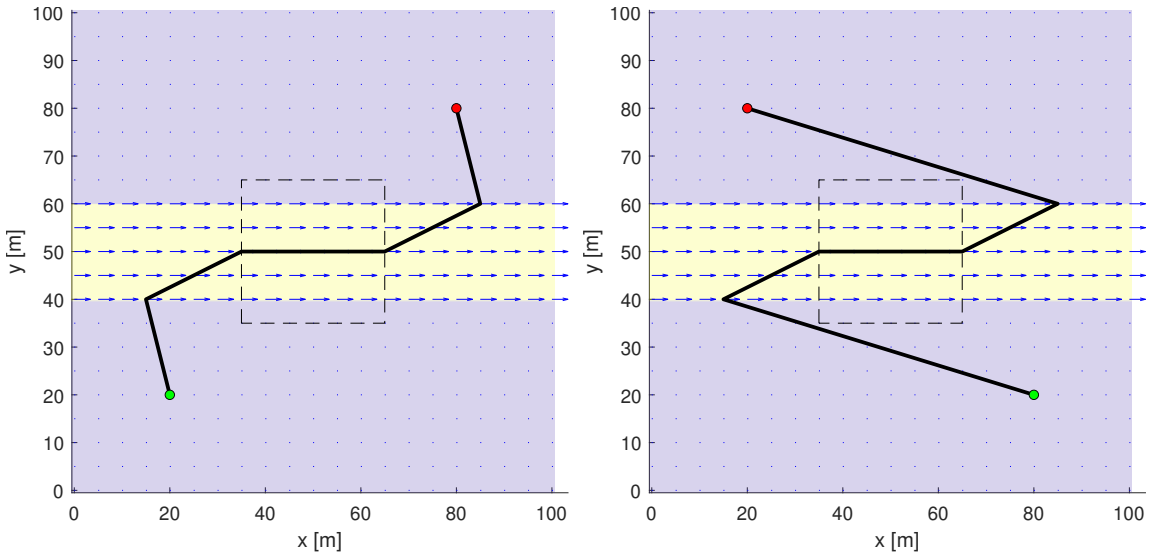


Figure 4.15: Adaptive Neighborhood Radius for the FlowFMT*. The neighborhood size is decreased when the streamlines curve to improve on the assumption that the flow velocity is constant inside of the neighborhood.

where r_n is the original radius proposed for the FlowFMT*. Figure 4.15 illustrates how this process works. Notice that r_n^* tends to r_n as our assumption of constant flow within the neighborhood region is better satisfied. If a discontinuity or large gradients are present, this change in size will reduce the chances of connecting samples far from each other.

Now, we move on from the examples in the literature and define a region where actuation is not permitted, as we wanted to study. In this jet crossing scenario, we define a square in the center of the map $W_{no\,actuation} = [35, 65] \times [35, 65]$ to be the region where the vehicle cannot actuate. Using the β_{max} angle shown in Figure 3.3, finding the minimum distance in the x direction required to cross the jet region is possible. This square was chosen sufficiently big so the vehicle could not cross the jet from side to side without entering this region. This calculation is omitted for brevity and suffices to say that its value is greater than 35, the distance from the non-actuating region to the environment's edges.

Considering the problem described above, we now try to find paths from the start position $\vec{x}_{start} = [20.0, 20.0]^\mathsf{T}$ to the goal positions $\vec{x}_{goal} = [80.0, 80.0]^\mathsf{T}$ and from the start position $\vec{x}_{start} = [80.0, 20.0]^\mathsf{T}$ to the goal positions $\vec{x}_{goal} = [20.0, 80.0]^\mathsf{T}$. In Figure 4.17, we show the optimal solution obtained numerically using the gradient method with cost



(a) Along case (b) Against case

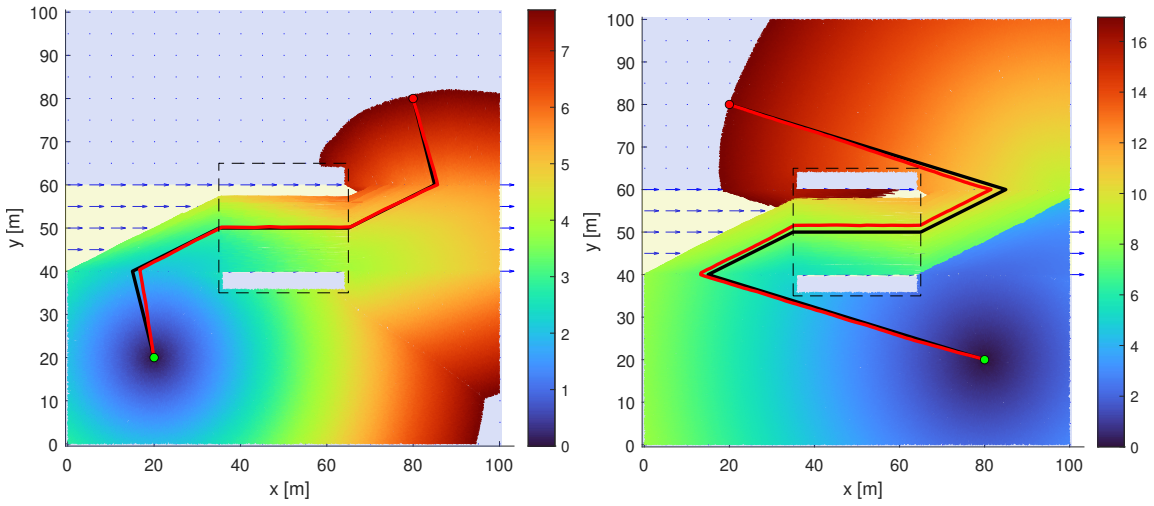
Figure 4.16: True solution of the jet flow crossing problem with regions of no actuation. The path in black is the optimal solution, found numerically.

$C_t(\sigma) = 7.6231\,\mathrm{s}$ for the "along" case and $C_t(\sigma) = 17.1015\,\mathrm{s}$ for the "against" case.

Again, very accurate solutions were found using FlowFMT*, which was run with 409,600 samples. For the "along" case, we obtained an optimal path with cost $C_t(\sigma) = 7.6332\,\mathrm{s}$ in $24.5869\,\mathrm{min}$ of computation time. The method found the solution after 228,927 iterations and 9,971,123 calls to the cost function. For the "against" case, we obtained an optimal path with cost $C_t(\sigma) = 16.9530\,\mathrm{s}$ in $35.5336\,\mathrm{min}$ of computation time. The method found the solution after 328,290 iterations and 14,831,087 calls to the cost function. For both cases, the discontinuity causes a slight difference between the correct solution and the one found with FlowFMT*, observable as a shift in the solution in Figure 4.17. Yet, it does not affect the cost significantly.

4.6 Experiment Setup

After validating the methodology, we present some results for motion planning of an aerobot navigating the atmosphere of Venus.



(a) Along Case (b) Against Case

Figure 4.17: FlowFMT* solution for a vehicle crossing a jet flow including a region with no actuation. The robot moves from the start position (●) to the goal (●). We show the regions where the speed of the flow is greater (■) and smaller (■) than the maximum speed of the vehicle relative to the flow. The tree is colored based on the cost of the arrival vertices. The numerical optimal solution is shown in black (—) and the solution found with the adapted FlowFMT* is shown in red (—).

Some changes were made to the design parameters of the aerobot shown in Table 4.1. The aircraft is buoyant at $55\,\mathrm{km}$, and its volume is $488.75\,\mathrm{m}^3$. The solar panel area is twice the wing area, $1000\,\mathrm{m}^2$. The maximum battery capacity is $1\,\mathrm{MJ}$. The solar panel efficiency is set to $\eta_{s.p.} = 20\%$. Also, to avoid chattering, i.e., switching on and off about the same battery level, the functions $IsActive$ and $IsInactive$ from Algorithm 8 that control the aerobot's actuation behavior were implemented with different threshold levels. The function $IsActive$ now returns false if the battery level is below 20%, and the function $IsInactive$ returns false if the battery level is above 80%.

We use the environmental properties obtained from the more realistic environment model from the Venus Climate Database (VCD) and attempt to solve the planet-scale problem. Table 4.3 shows the limits of the configuration space. Wind speed and solar intensity data were retrieved from the VCD for each sample for longitudes from $-180°$ to $180°$ in steps of $3.75°$, latitudes from $-90°$ to $90°$ in steps of $1.875°$, and altitudes from $50\,\mathrm{km}$ to $70\,\mathrm{km}$ in steps of $5000\,\mathrm{m}$. These properties (wind speed and solar intensity) were interpo-

Table 4.3: Configuration Space Limits

Parameters	Values	Parameters	Values
φ_{min}	$-20°$	φ_{max}	$20°$
λ_{min}	$-180°$	λ_{max}	$180°$
z_{min}	$50\,\mathrm{km}$	z_{max}	$70\,\mathrm{km}$
b_{min}	$0\,\mathrm{MJ}$ (0%)	b_{max}	$1\,\mathrm{MJ}$ (100%)

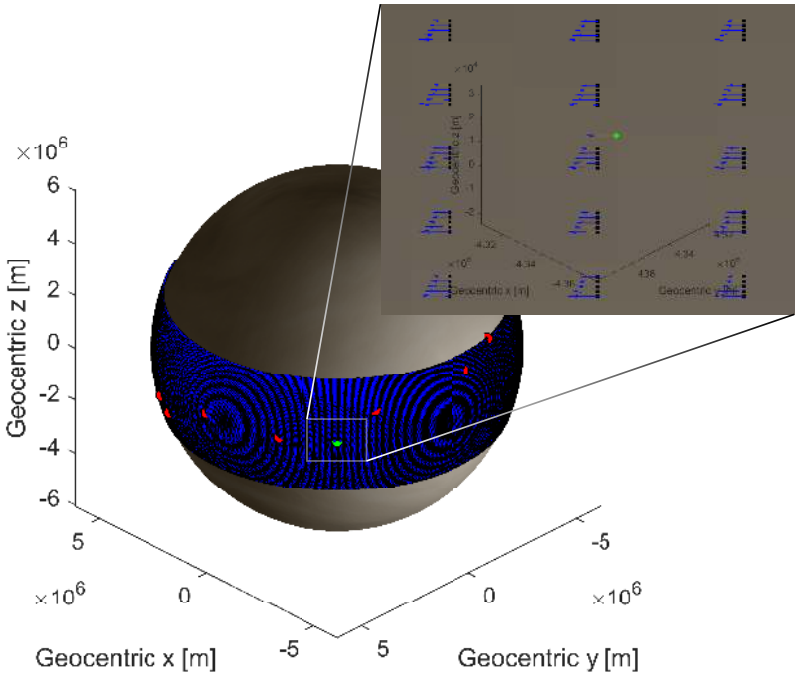
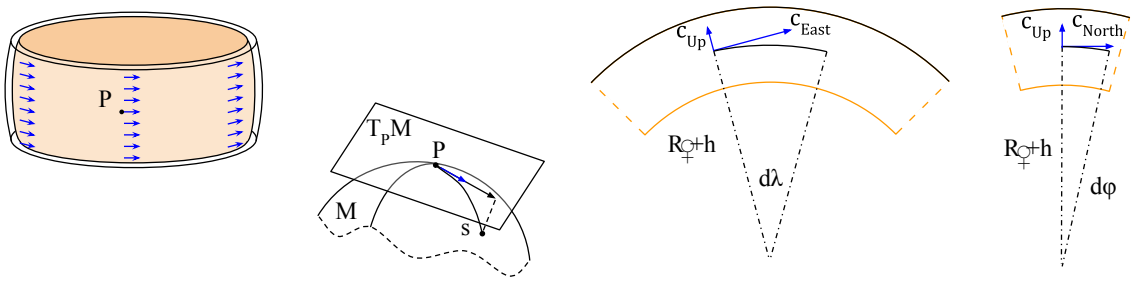


Figure 4.18: Workspace regular sampling. The superrotation winds mostly define the wind field as seen in the zoomed-in region.

lated using Matlab's *interpn* function, precomputed, and saved for each sample and start and goal location. They were also precomputed and saved with respect to the geocentric frame.

In Figure 4.18, we show the wind field and the sampling we used for our experiments. Latitudes are sampled at $0.25°$, longitudes are sampled at $0.5°$, and altitude are sampled at $3000\,\mathrm{m}$. The sample coordinates were also precomputed and saved with respect to the geocentric frame. Given this discretization, a total of $576,000$ samples were used. We chose to sample the workspace uniformly for better visualization of the trajectories.



(a) Manifold and its tangential space

(b) Latitude and longitude arcs

Figure 4.19: Drifting with the flow model. Approximating geodesic paths as a composition of independent motions. Changes in latitude and longitude due to tangential speeds are approximated as arc lengths with constant radii, and changes in altitude are approximated as a linear contribution due to vertical speeds.

During the execution of Algorithm 8, when the vehicle is active, we connect samples using the geocentric frame, as shown in panel B of 4.5a.

4.6.1 Drifting with the Wind and Geodesics

The next step to apply the FlowFMT* to the Venus problem is to create a motion model for the vehicle drifting with the winds when it runs out of battery. Considering that the upward wind speeds are lower than the horizontal wind speeds ($c_z << \sqrt{c_y^2 + c_x^2}$), it is reasonable to assume that the wind moves tangentially to a spherical surface at each altitude of the atmosphere of Venus. When Algorithm 8 goes into the no-actuation mode, the wind needs to be integrated. Points are not "forced" to be connected with a straight line until the integration is complete. As demonstrated by Figure 4.19a, blind integration of the wind in the manifold M at each point P would lead to an accumulation of error towards the tangential direction. This integration is done by the $Drift$ function, which evolves the aerobot's state over time when the vehicle does not have actuation.

To ground the integration into a more reasonable path for the vehicle in the atmosphere when drifting with the flow, we consider that the vehicle's movement is affected independently by the three wind components in the local tangential frame. We define an approximation for the motion as a composition of constant radius arcs for changes in latitude and

longitude. Their curvature is obtained from the planet's radius and the vehicle's altitude at the parent vertex $(R_{\varphi} + h)$. For the change in altitude, we used a simple integration of the vertical wind speed and the apparent acceleration of gravity. Considering that the vehicle is fully buoyant at $55\,\mathrm{km}$, we limited the descent to $55\,\mathrm{km}$ if $h + \Delta h$ gives a value smaller than that. To summarize, the equation that defines the change in the vehicle is

$$\begin{bmatrix} \Delta\varphi \\ \Delta\lambda \\ \Delta h \end{bmatrix} = \begin{bmatrix} c_x/(R_{\varphi} + h) \\ c_y/(R_{\varphi} + h) \\ c_z \end{bmatrix} \Delta t_{drift} + \begin{bmatrix} 0 \\ 0 \\ (\rho(h)V/m - 1)g(h) \end{bmatrix} \Delta t_{drift}^2 \,, \tag{4.13}$$

where Δt_{drift} is the period we allow the vehicle to drift. This period is calculated based on the environment's characteristic length is $L_c = 2\pi R_{\varphi}$ and the maximum flow speed c_{max} =$150\,\mathrm{m\,s^{-1}}$. From them, we obtain the characteristic time as $T_c = L_c/c_{max}$ =$150\,\mathrm{s}$. The reference drifting time interval, considering $N_{ts} = 100$ is $\Delta t^*_{drift} = T_c/N_{ts}$ = 100. Finally, to guarantee some randomness and avoid vertices being added to the closed set for being too close to each other, the drifting time interval is obtained by sampling a uniform interval around the reference: $\Delta t_{drift} = (0.5 + a)\Delta t^*_{drift}$, where a is sampled for a uniform distribution $U(0, 1)$.

In the following section, we run experiments with this setup. In contrast to the approach taken with EWRRT (see Section 4.4), where we considered that the airship state included its orientation and Dubins' paths were found. The deformed using the wind field, now, using the FlowFMT*, we assume that the vehicle is holonomic and that the connection between each pair of vertices is a straight line. We use the same cost function introduced in Section 3.3 to optimize.

Table 4.4: Minimum-Time Goal Set ($\mathbf{X}_{goal}$)

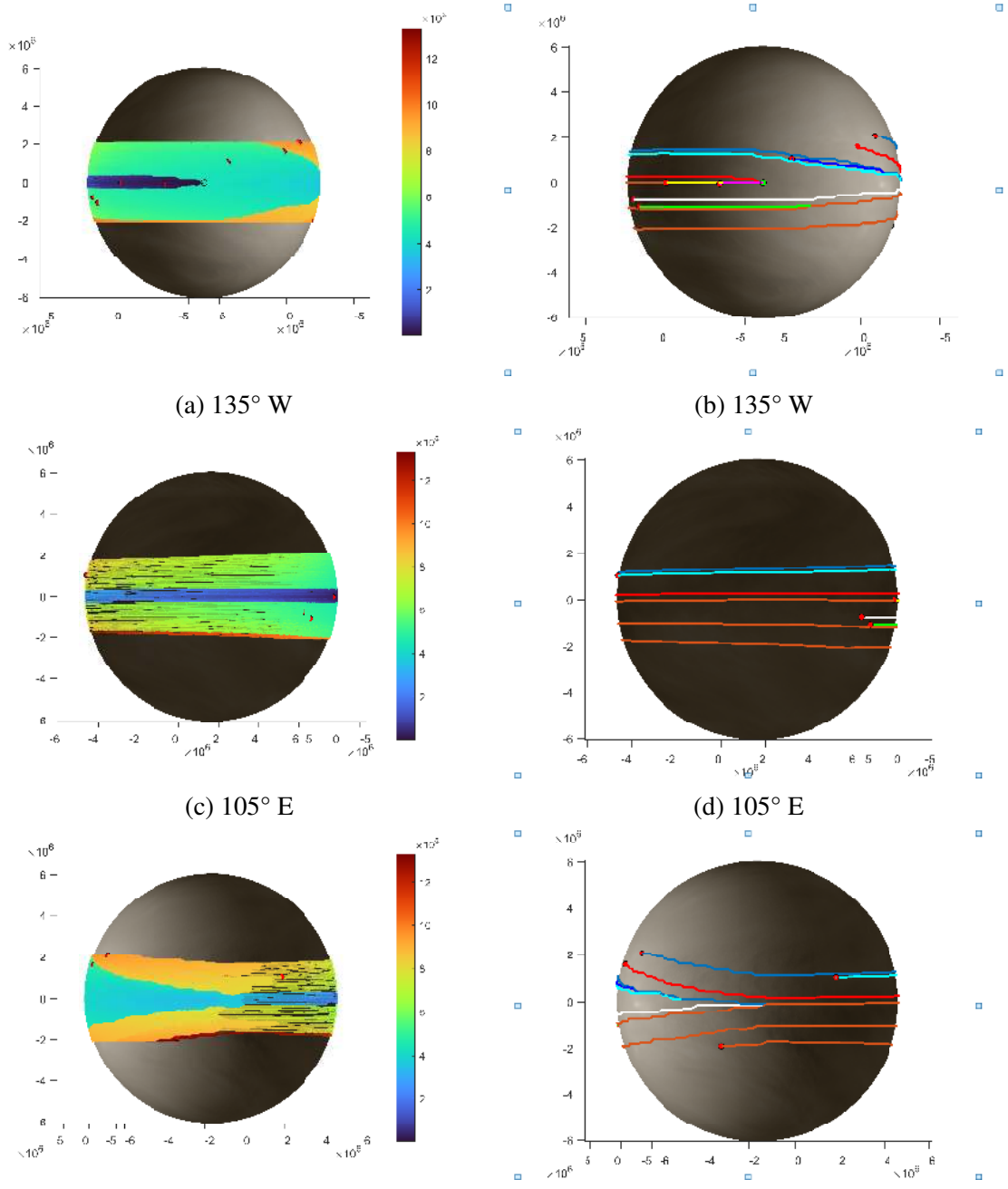
Goal	Latitude [°]	Longitude [°]	Altitude [km]
Goal 1	$14.71°$	$-178.9°$	$55.0\,\mathrm{km}$
Goal 2	$9.434°$	$147.9°$	$69.0\,\mathrm{km}$
Goal 3	$-10.44°$	$69.34°$	$69.0\,\mathrm{km}$
Goal 4	$-0.1257°$	$90.87°$	$69.0\,\mathrm{km}$
Goal 5	$9.937°$	$-59.33°$	$55.0\,\mathrm{km}$
Goal 6	$-0.1257°$	$117.4°$	$65.5\,\mathrm{km}$
Goal 7	$-7.421°$	$63.33°$	$69.0\,\mathrm{km}$
Goal 8	$19.74°$	$-164.4°$	$69.0\,\mathrm{km}$
Goal 9	$-18.23°$	$-119.9°$	$69.0\,\mathrm{km}$

4.7 Experiment Results

The starting location of the aerobot was set to $(\varphi, \lambda, h)_{start} = [135,\ 0,\ 70000]^T\,\mathrm{km}$ and the aerobot was initially given full battery. The starting location is still on the day side but already close to the night side. Nine different goal locations were considered and compiled in Table 4.4.

Figure 4.20 shows the search tree and the path solutions found by FlowFMT* in this experiment with an orthographic view centered at longitudes 135° W, 105° E, and 15° W, respectively. The edges are colored by the costs of the children's vertices. The tree expands in latitude, coming out from the starting vertex until the aerobot runs out of battery (Figure 4.20a). During the night side, the tree is predominantly parallel to the Equator line (Figure 4.20c). That is the case until the aerobot reaches the day side when the tree can extend in latitude again. It is possible to see in Figure 4.20c a region where the tree is colored red, which was only reached after going around the planet three times. Figure 4.21 shows a Robinson map projection of trajectories from the start to each of the nine goals.

The evolution of battery, latitude, longitude, and altitude are shown in Figure 4.22 for goals 4, 5, 8, and 9. These goals were selected because they span cases where the aerobot has to travel increasingly more distances to reach these destinations. For goal 4, the aerobot travels for less than a day, not enough time for the battery to discharge and



(a) 135° W (b) 135° W

(c) 105° E (d) 105° E

(e) 15° W (f) 15° W

Figure 4.20: Time-Optimal FlowFMT* solution for an aerobot navigating the atmosphere of Venus.

affect its behavior. It moves straight at higher altitudes, where the winds are faster, and the solar intensity is higher. For goal 5, the vehicle moves in higher altitudes as its battery discharges. When the battery levels reach under 20%, the aerobot stops actuating. The

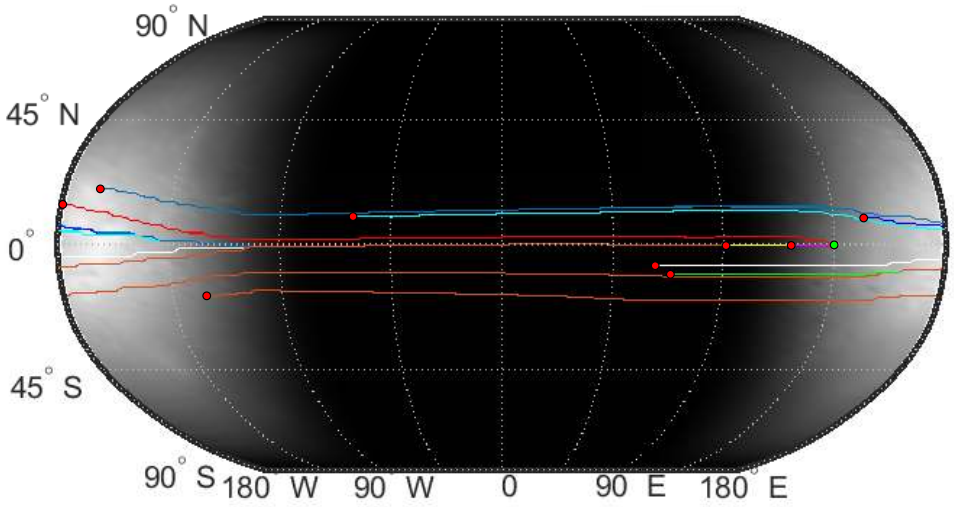
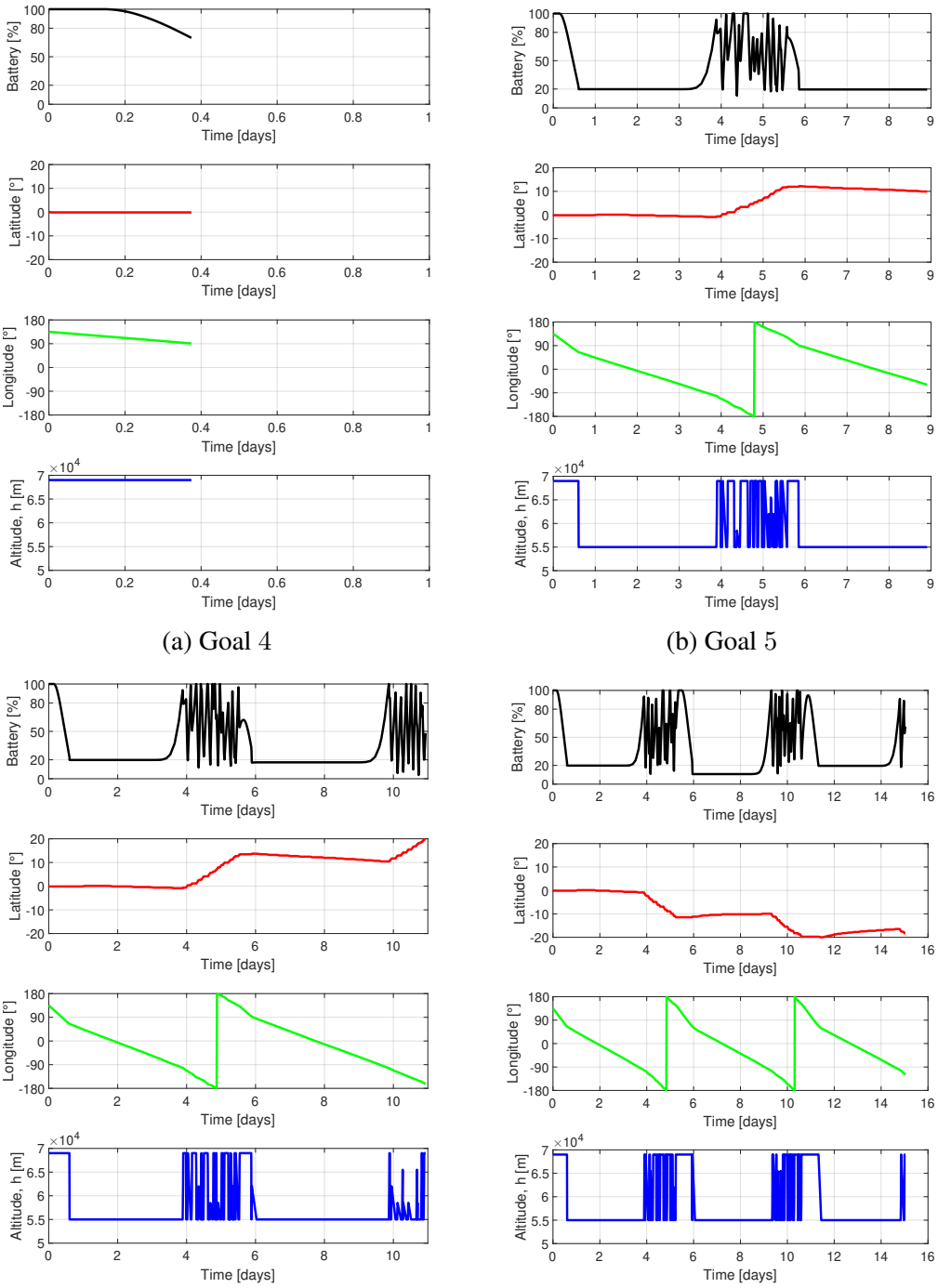


Figure 4.21: Time-optimal solution for an aerobot navigating the atmosphere of Venus found by the FlowFMT*. In this map projection, the solar intensity at the altitude of $70\,\mathrm{km}$ as a background, and solutions for the nine goals listed in Table 4.4 are plotted in red. We observe that the motion in latitude is possible when the vehicle is active, in the lighter regions, where it can recharge.

aerobot is carried by the wind for around three days, with minimum changes in latitude due to the meridional winds. Around day 4, it has already circumnavigated the planet and reaches the day side again. At this point, it alternates between moving fast at higher altitudes, descending to lower altitudes as it runs out of battery, and drifting for some time while it charges. During the day, the planning tree expands to the target latitudes. Goals 8 and 9 roughly show the same behavior, with the addition that to reach goal 8, the aerobot undergoes two nights, and to reach goal 9, the aerobot undergoes three nights.

4.8 Summary

In this chapter, we proposed strategies to solve the navigation problem for an aerobot in the atmosphere of Venus, which is a challenging task due to the strong winds, uncommon topology, and large environment. Two specific aspects make the problem of exploring Venus' atmosphere difficult. Firstly, the strong winds in the atmosphere of Venus create a



(a) Goal 4

(b) Goal 5

(c) Goal 8

(d) Goal 9

Figure 4.22: Time-Optimal FlowFMT* solution for an aerobot navigating the atmosphere of Venus. In (a) with goal 4, the aircraft did not enter the dark region; in (b) with goal 5, the aircraft entered the dark region once; in (c) with goal 8, the aircraft entered the dark region twice; and in (d) with the goal 9, the aircraft entered the dark region three times.

reachability problem and, as shown in this chapter, depending on the combination of start and goal locations, the vehicle needs to loop around the planet. Secondly, the aerobot is solar-powered, and when it moves onto the night side, it cannot recharge, which becomes a significant issue because, during this time, the aerobot runs out of battery and cannot actuate.

Our first attempt to solve the problem was embodied with the EWRRT planner, which considered that the aerobot was nonholonomic but (i) could not find optimal paths, (ii) was restricted to small portions of planning space, and (iii) did not account for the night side of the planet. To overcome these challenges, we extended the method proposed in the previous chapter, the FlowFMT*, for non-actuating regions. First, the method was validated with a simpler and more tractable example, the jet crossing flow. Finally, it was employed with the Venus problem at the planet scale, when we were able to observe (1) trajectories that looped the planet multiple times to reach the goal destination, (2) the vehicle moving up in the atmosphere to recharge and down as it runs out of battery, (3) the trajectories mainly being aligned with the equator line on the night side of the planet, due to the predominance of zonal winds over meridional and vertical winds. Other than the Venus problem, applications of the FlowFMT* for non-actuating regions could include surveillance vehicles that move through quiet zones where the vehicle needs to shut down its actuator to avoid disturbing the environment.

As described in Chapter 3, we could potentially convert the plans into vector field policies to allow trajectories from any position on the planet to a specified goal coordinate. In Chapter 5, a method is proposed to facilitate using these policies in real-world scenarios when new information about obstacles or restricted regions is obtained during the vehicle's navigation. We propose a local planner that seeks to optimize the vehicle's motion by following a guiding vector field (policy) and avoiding obstacles detected within the aerobot's sensing range.

Chapter 5

Vector Field Navigation in the Presence of Unknown Obstacles

This chapter proposes a novel methodology for real-time motion planning in cluttered environments. The methodology proposed here allows an autonomous vehicle to follow artificial vectors, such as the ones generated by the policies presented in chapters 3 and 4, and avoid obstacles that were unaccounted for during the field computation. Part of this chapter was published first at the 2021 IEEE International Conference on Robotics and Automation (ICRA) [52] and subsequently, as a final version, in the MDPI Sensors journal [53].

5.1 Introduction

As autonomous vehicles and other mobile robots are integrated into our world, they need to be able to navigate in dynamic, tight, and cluttered scenarios. Example applications, where both ground and aerial robots are exposed to such environments, include logistics [102, 103] and forest monitoring [104, 105]. This indicates that the robots must use their sensors to perceive the environment and its changes in real time [8] and then react with powerful and efficient motion plans. A significant challenge for these robots is planning up to the bound of the control feedback loop frequency (e.g., ≈100Hz) [106]. A current limiting

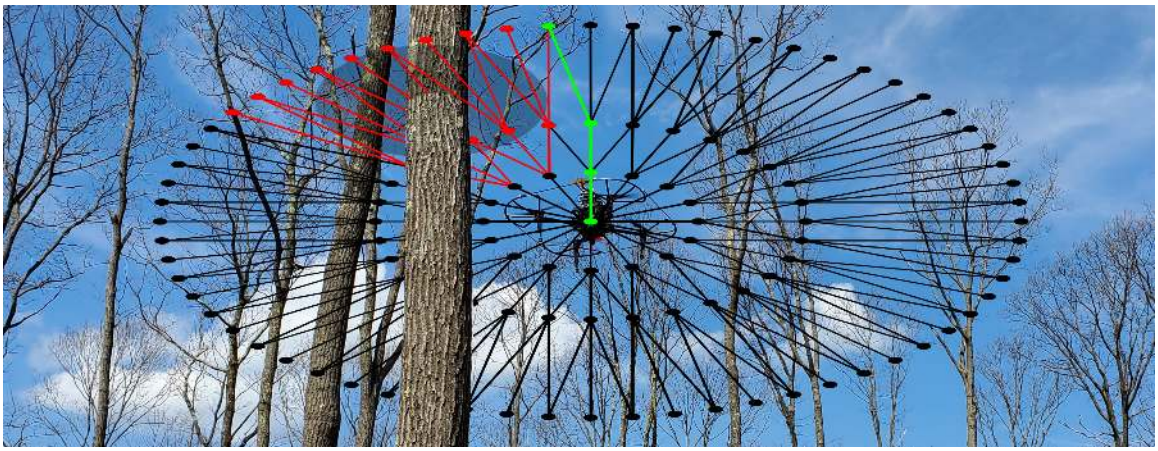


Figure 5.1: Illustration of the proposed methodology. A lattice in the sensor space encodes, in a search tree, the set of paths that the robot (i.e., drone) can take locally. The optimal path (green edges) is computed in real-time by pruning the edges of the tree that lead to collision (red edges) and minimizing a path cost functional on the remaining tree.

factor for this high performance is that motion planning algorithms spend significant time on collision checking, especially in environments with multiple obstacles. Approximately 90% of the total planning time on probabilistic roadmaps (PRM) algorithms, for example, is spent on collision checking [107]. One of the main reasons is that these algorithms, which could be massively parallelized, execute each of its steps in sequence [108, 107]. Therefore, our research aims to develop efficient parallel motion planners that allow robots to move at high speeds in cluttered and dynamic environments.

5.1.1 Problem Definition

Given a robot working in an environment populated with unknown static and moving obstacles, plan the local motion of the robot such that it can safely and quickly move among the obstacles without compromising the execution of its global task. To solve this problem, the main challenge addressed in this chapter is the real-time computation of motion plans. Solving this challenge will allow the robot to traverse the environment at speeds only limited by its hardware and low-level path-following controller.

This chapter presents the Sensor-Space Lattice (SSLAT) motion planner, an embarrass-

ingly parallel motion planner that computes paths in the sensor space of a robot in real-time using unprocessed sensor data. SSLAT is a hybrid planner that consists of two hierarchical layers. The top layer consists of a vector field, which is used to encode the global task of the robot (e.g., following a road, traversing a forest, tracking a boundary). The vector field ignores small and dynamic obstacles. In the lower layer, a lattice in the sensor space encodes a search tree representing a set of precomputed paths. An illustration of such a lattice is shown in Figure 5.1. The lattice is quickly pruned using sensor measurements to allow for obstacle avoidance. A cost-functional that indicates how aligned a path is from the vector field is used to select the best pre-computed path. Therefore, SSLAT can be considered part of a global navigation solution, provided that a global vector field is available or as a local method for obstacle avoidance when simple local fields are used. In the experiments presented in this chapter, we mainly evaluate the ability of SSLAT to perform obstacle avoidance but also illustrate, through a real-world experiment, that it can be used to control a drone for persistent monitoring of a forest using a global vector field. The definition of global vector fields is not the focus of this chapter, but it is considered in several recent publications [109, 110, 111, 112]. SSLAT was designed to work with planar Light Detection and Ranging (LIDAR) sensors, which makes it useful for several robots, including self-driving cars and drones. In this chapter, our implementation focused on parallelizing the method's collision detection sub-routine (tree pruning) using an NVidia® CUDA®-enabled Graphics processing unit (GPU). Tree pruning is the most expensive component when computing plans with SSLAT. Due to the structure of the method, other sub-routines of SSLAT can also be parallelized in future implementations.

5.1.2 Contributions

The main contributions of this chapter are as follows: (1) a lattice-based method that optimizes a local, vector-field-dependent functional for obstacle avoidance and field tracking; (2) a method for generating a lattice in the sensor space of the robot that simultaneously tes-

sellates the space and stores in a tree graph the possible paths that a robot can take locally; (3) an embarrassingly parallel strategy based on a precomputed mapping between sensor measurements and lattice edges for collision detection in parallel architectures, including CUDA. Compared to previous graph-based methods, the main advantage of using a lattice to find paths that avoid obstacles and follow a vector field (Contribution 1) is the high efficiency of obstacle detection in the lattice. While traditional graph-based approaches require several collision checks for each new node being inserted on the graph, the fixed and regular structure of the lattice allows for a constant and small number of tests. To simplify and speed up even more, in these tests, our lattice design (Contribution 2) enables a direct correspondence between the edges of the lattice and geometric shapes (triangles) in a way that a single collision detection invalidates a pair of edges and nodes of the graph.

5.1.3 Organization

The rest of this chapter is structured as follows. The motion planner methodology is explained in Section 5.3. The parallelization of the algorithm is detailed in Section 5.3.2. Section 5.4 presents a series of simulations and real robot experiments that illustrate and evaluate the proposed approach, followed by a discussion. Finally, Section 5.6 summarizes this chapter's contents.

5.2 Related Work

Vector fields represent a simple way of providing the robot with a preferred direction of motion for each point in its workspace. They can be easily computed with the use of artificial potential functions (APF) [31] or navigation functions [33]. However, using these techniques, which consider obstacles, the vector field would change every time the map of the environment changes (e.g., people or other vehicles enter the environment). An alternative strategy is to construct the vector field ignoring the obstacles and solve obstacle

avoidance in a lower-level planner, as suggested in [51, 113, 44]. Thus, the vector field is used to encode the high-level specification of a task, which may be, for example, the periodic survey of a given curve [113]. Vector fields for curve circulation were proposed in [110, 109, 114, 115].

When the vector field ignores the obstacles, local motion planning is needed to avoid obstacles and follow the vector field. The idea of modifying preexisting plans with local information is not new and was used by the authors of the Elastic Bands [116], which deforms the path computed offline by a global planner in real time. Two legacy obstacle avoidance approaches that use planar sensors to measure the surroundings of the robot are the Virtual Force Field (VFF) [117] and Vector Field Histogram (VFH) [118]. The former builds an occupancy grid centered on the vehicle. Repulsive forces are calculated for each cell using the same idea of APF but corrected with a weighting function. VFH is an improved version of VFF that maps obstacles to a polar graph with peaks where the obstacles are likely to be and valleys in directions that are likely to be free of them. The second step uses heuristics to choose which valley to follow. New variations of VFH have been recently proposed [119], including some that explicitly include moving obstacles [120]. An alternative local obstacle avoidance method is the Dynamic Window Approach (DWA) [121], which constructs a window of allowed velocities based on sensor data. Although considered a legacy approach, DWA is still a prevalent method. Recent works using DWA include [122], which proposes a new method for avoidance of dynamic obstacles that look for free gaps in between the obstacles to compute a trajectory and applies the DWA to safely follow such a trajectory assuming that the obstacles can move. More related to our work, assuming that a global vector field gives the desired velocity of the vehicle, the dynamic window was used to detect if the field can be followed safely [43]. If the velocity vector given by the field leads to a collision, this vector is modified before being sent to the vehicle.

Another important aspect of our approach resembles the methodologies introduced in [123] and [124]. In [123], precomputed relations between sensor space and planning,

embedded in collision detection circuits (CDCs), are used to prune collision nodes in a probabilistic roadmap. In [124], precomputed alternative paths generated offline are pruned using real-time data from the robot's onboard sensor. Similarly, our method uses precomputed relations to eliminate paths in collision with obstacles. However, we chose to build a regular discretization of the sensor's field of view using trees (graphs without cycles) that eliminates paths that are in collision after pruning and speeds up searching for the optimal path. To create the tree on the sensor space, we decided to use a lattice inspired by Bethe Lattices and Cayley trees [125]. Our approach is related to ego-graphs for path planning, proposed in [126], state lattices that precompute a set of actions while accounting for dynamic constraints [127], and motion primitives [128] methods. The main difference of our proposed discretization is that the spatial representation of our lattice is carefully designed to obtain the mappings between sensor measurements and the traversable paths, thus allowing for fast pruning. Further, we use our lattice to track the global behavior (vector field) by minimizing a cost-functional that measures how much the paths are aligned with the field, as is done in [51] using the Rapidly Exploring Random Tree Star (RRT*) algorithm [129].

Using precomputed trees using lattices, besides facilitating obstacle detection, also makes the planner embarrassingly parallel, i.e., it can be easily divided into a finite number of parallel tasks [130]. There is only limited literature about the parallel implementation of motion planners. Some work with parallel architectures was done for sampling-based algorithms such as Rapidly Exploring Random Tree (RRT) [131], PRM [132], and RRT* [107] and their variations [133, 134]. Parallel grid-based path planners, such as the A*, were also investigated [135]. High-dimensional planners, such as the R* search, were parallelized and shown to preserve all theoretical properties, including probabilistic bounds on sub-optimality [132]. The Group Marching Tree (GMT*) was parallelized, allowing planning in real-time with solutions for complex planning problems under differential constraints found in the order of $\approx$30 ms on an embedded GPU [106]. A common way of parallelizing planning algorithms is to make some of their functions, such as neighbor search-

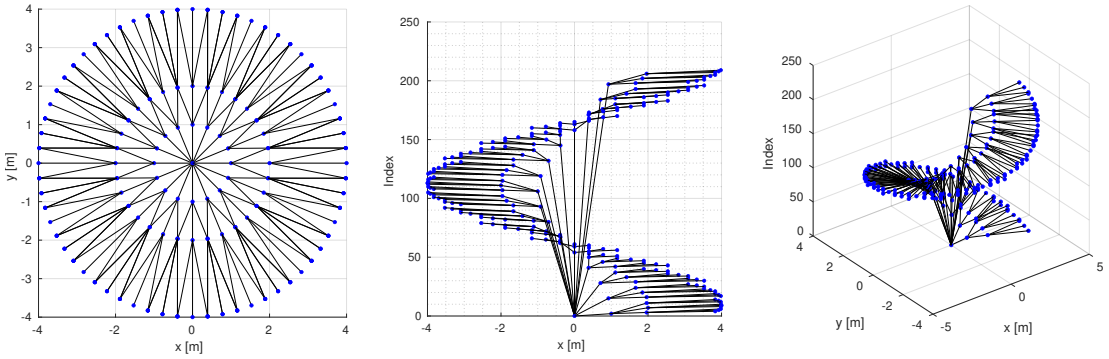
ing [136, 137] and collision checking [107, 138], parallel. For example, the authors of [138] proposed a parallel approach for collision detection that could perform half a million collision checks per second using benchmark problems, which is 10 times faster than prior parallel techniques. They were also able to compute collision-free paths for a great range of models (single body or multi-bodies) in less than $100\,\mathrm{ms}$ for many benchmarks, which is almost 50–100 times faster than current sequential PRM planners. In the motion planning proposed in this chapter, we perform parallel collision checking and obtain speedups in the order of 80 times compared to our collision-check approach's sequential implementation. Moreover, we do not see any degradation in the planning efficiency when the number of obstacles and beams in the LIDAR sensor increases. The following section will present the details of our approach.

5.3 Methodology

This section describes the methodology developed to solve the above-mentioned problem, the Sensor-Space Lattice motion planner (SSLAT). At the end of the section, we describe a parallel implementation of the algorithm that allows for faster planning times.

5.3.1 Sensor-Space Lattice Motion Planner

SSLAT is a technique that relies on (1) a pre-computed tree graph that embeds a set of paths that a robot could take in its current field of view and (2) sensor-to-graph mappings that can be used to quickly prune the edges of the graph that would lead to collisions between robot and obstacle. Given this, the planner is divided into two steps: an offline step that computes the graph, the sensor-to-graph mappings, and the global task (represented by a vector field), and an online step that prunes the tree and finds the free path that closely follows the vector field. These two steps are detailed next.



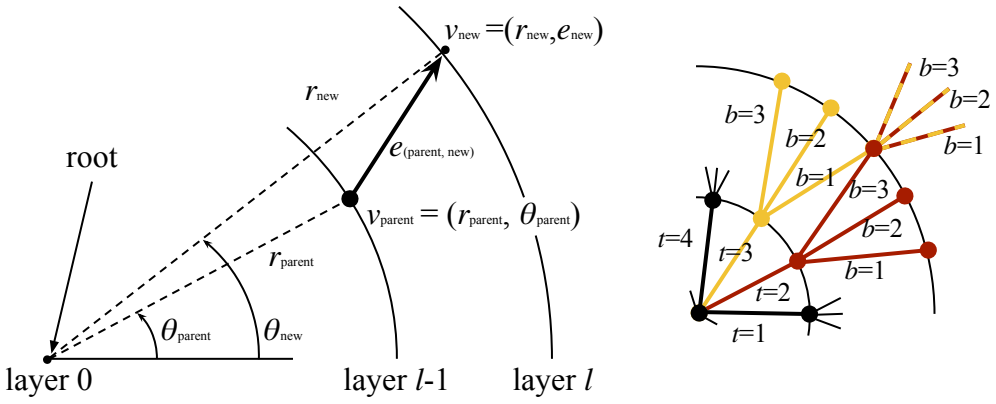
(a) (b) (c)

Figure 5.2: A lattice with $(K, N_T, N_B, N_L, r_0) = (2, 16, 3, 3, 1)$. In black, we see the edges that belong to the graph. In (b) and (c), it is possible to view an exploded view of the lattice plotted in 3D to emphasize that two non-neighbor nodes may have the same position in the plane. The graph comprises a single tree with only one path from each vertex to the root.

Offline Step

The first step of SSLAT, executed when the robot is initialized, is subdivided into three, as described in the following subsections:

Computing the lattice To encode robot local paths, we create a lattice represented by graph $\mathcal{G}(V, E)$, which is a planar-directed tree spatially contained in the sensor space, $\mathcal{S} \subset \mathbb{R} \times \mathbb{S}$. Figure 5.2, at the left, shows an example of a lattice created with our methodology. The tree is constructed recursively from the root node, which coincides with the sensor space's origin. New vertices are created in circular layers around the root. In the first layer, the vertices are equally spaced in a circle of radius r_0 and connected by edges from the root node. The number of vertices in this layer, N_T, defines the number of trunks of the tree, equivalent to the number of sub-trees connected to the root. Each vertex of the first and subsequent layers branches out to N_B children vertices in the subsequent layers. The creation process of vertices is repeated until the number of layers, N_L, is reached. The growth ratio, K, defines the relationship of the layers' radii. With these parameters (namely: r_0, N_T, N_B, N_L, and K), the polar coordinates $(r, \theta) \in \mathcal{S}$ of each vertex $\mathrm{v} \in V$



(a) (b)

Figure 5.3: (a) Construction of the lattice adds vertices in layers around the sensor origin. Each new vertex (v_{new}) is added according to its parent vertex ($\mathrm{v}_{\text{parent}}$). This process is executed recursively starting from the root node, where trunks $\mathrm{t} = 1, 2, \ldots, N_T$ are added, connecting it to the first layer of the lattice. (b) The leaf nodes are subdivided for each new layer in branches $\mathrm{b} = 1, 2, \ldots, N_B$. Notice that, during this process, sub-trees are created with coincident spatial representation for some vertices, as exemplified in red and yellow.

of the graph can be found from the polar coordinates of its parent vertex as:

$$r_{\text{new}} = K^{(\mathrm{t}-1)} r_0 \,,$$

$$\theta_{\text{new}} = \begin{cases} \left(\frac{2\pi}{N_T}\right)(\mathrm{t}-1) & \text{if the parent is the root node} \\ \theta_{\text{parent}} + \left(\frac{2\pi}{N_T}\right)\left(\frac{\mathrm{b}-(N_B+1)/2}{(N_B-1)^{(\mathrm{t}-1)}}\right) & \text{otherwise} \,, \end{cases}$$

where $\theta_{parent} = \tan^{-1}(y_{\text{parent}}/x_{\text{parent}})$, l is the index of the l-th layer, t is the index of the t-th trunk, and b is the index of the b-th branch of each trunk (see Figure 5.3). Notice that there will be more than one vertex in the same (r, θ) position, but they are not connected since they come from different trunks. This can be observed by the yellow and red sub-trees in Figure 5.3 and is better visualized when the lattice is plotted in 3D with the node index as the third dimension, as shown in right plots of Figure 5.2. Although this generates extra data in the graph representation, keeping the nodes disconnected also keeps the graph acyclic, as a tree, which is important for the online optimization we perform using the graph.

Obtaining sensor-to-graph mappings After generating the lattice, it is possible to create a tessellation of the region around the origin as a mesh of non-overlapping triangles that have exactly two of their sides coincident with the edges of the graph. Figure 5.6(a) shows an example of such triangulation. Let $\cdot$ be the set of triangles obtained by the triangulation. By comparing the coordinates of the vertices that compose each edge and the coordinates of the triangles, it is possible to compute a map $\mathcal{R}^E_\cdot : \cdot \rightarrow E$ for which each edge $\mathrm{e} \in E$ will have two triangles $\tau \in \cdot$ assigned to it. As the sub-trees of $\mathcal{G}$ may overlap spatially, this will give a one-to-many relation where each triangle will be linked to many edges.

Let $\mathcal{L}$ be the set of range measurements from the sensor. For planar LIDARs, it is well known that these measurements have a particular angular distribution. For example, the sensor can have a beam at each $0.5°$ around the $360°$ circumference. It is then possible to compute a map $\mathcal{R}^\cdot_\mathcal{L} : \mathcal{L} \rightarrow \cdot$ by checking which triangles can be intersected by each range measurement. Thus, each sensor range will have a list of triangles that must be tested for collisions.

By composing $\mathcal{R}^\cdot_\mathcal{L}$ and $\mathcal{R}^E_\cdot$ we can quickly prune the edges of the tree that are related to the obstacles detected by the sensor. Notice that, for each range measurement, only a few well-defined edges can be pruned.

Defining a global task The lattice-shaped tree presented in subsection 5.3.1 is limited to being inside the sensor's field of view. Therefore, it cannot account for the entire robot task. In the proposed methodology, we frame the global plan as an artificial vector field designed to follow curves in the workspace. Computation of such artificial vector fields was extensively explored in [114] and [139], where it is proposed that the superposition of a normal component can obtain a vector field in a two-dimensional environment, $\mathbf{N}$, which makes the robot converge to the curve to be followed, and a tangential component, $\mathbf{T}$, which drives the robot along the curve. The vector field is then defined for the coordinates

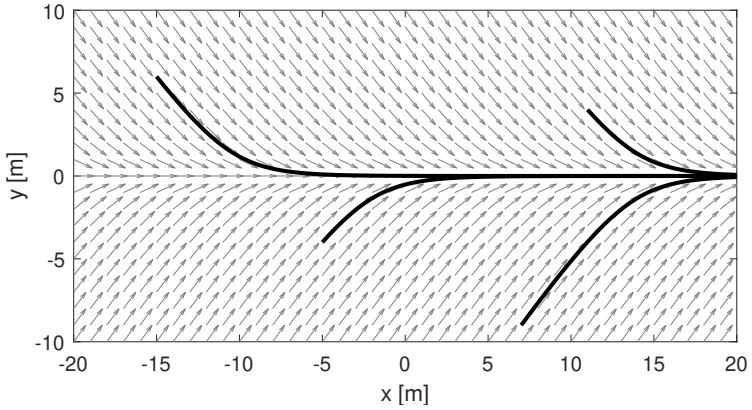


Figure 5.4: Example vector field that guides a robot to follow the x-axis. The gray arrows represent the velocity given as input to the vehicle. The black curves are examples of paths the vehicle would take by following the field.

$(x, y)_g$ in the global frame g as:

$$\mathbf{F}_v\big((x,y)_g\big) = \mathbf{N}\big((x,y)_g\big) + \mathbf{T}\big((x,y)_g\big)\,. \tag{5.1}$$

With the correct choice of functions, obtaining a normalized vector field for a given curve is possible. For example, Figure 5.4 presents a vector field that can be used to follow a straight line aligned with the x-axis of the world. As proposed in [114], this field can be obtained by making $\mathbf{N}\big((x,y)_g\big) = 1/\sqrt{1+f(y_g)^2}$ and $\mathbf{T}\big((x,y)_g\big) = f(y_g)/\sqrt{1+f(y_g)^2}$ where $f(y_g) = -\arctan c\, y_g$, and c is a convergence parameter. This parameter was set to 0.5 in Figure 5.4.

Online Step

By avoiding the necessity of generating the paths themselves, the motion planning problem to be executed online is reduced to perceiving the obstacles, pruning the tree accordingly, assigning costs to the different paths embedded in the tree, and choosing the best path to be followed. These steps are summarized in Figure 5.5 and explained in detail in the following subsections.

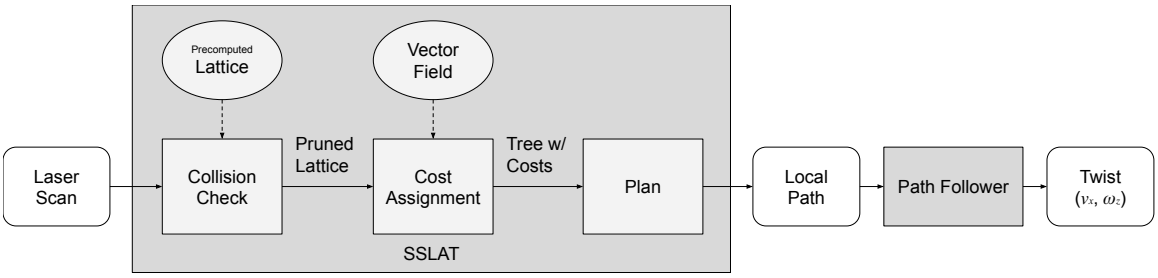
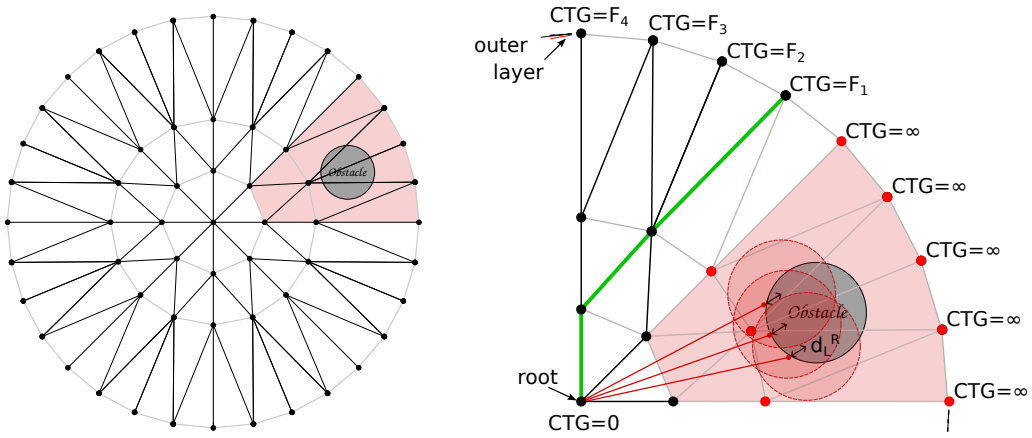


Figure 5.5: Flowchart of the method. The sequence of functions (bottom row of the chart) is run every time a new measurement (laser scan) is available.

Tree pruning Let $l_m \in \mathcal{L}$ be the m-th range measurement coming from the LIDAR sensor, $(x, y)_m$ its corresponding Cartesian values in the sensor frame, and C_m be a circular region with robot radius r_R centered at $c_R = (x, y)_m + d_L^R$, with d_L^R being an offset representing the translation from the sensor frame to the robot frame. Without loss of generality, we assume that both frames have the same orientation since the lattice can be constructed accordingly. The laser measurements are filtered, and to be considered valid, a range measurement needs to be less than the lattice outer layer radius plus the robot radius. For each triangle $\tau \in \cdot$ obtained from $\mathcal{R}_{\mathcal{L}}^{\cdot}(l_m)$, a collision between the m-th valid LIDAR beam and the triangle τ is detected if $\tau \cap C_m \neq \emptyset$. Computationally, this check can be done by a function that returns true if: $(c_R \cap \tau \neq \emptyset) \wedge ({}^{\tau}\mathrm{e}_i \cap C_m \neq \emptyset)$ for $i = 1, 2, 3$ and false otherwise, where ${}^{\tau}\mathrm{e}_i$ are edges of the triangle ${}^{\tau}$. An illustration of this approach is shown in Figure 5.6. When a collision is detected, all the edges and corresponding vertices defined by the triangle are marked as untraversable, thus pruning the tree. The remaining tree only contains edges that the robot can safely follow.

Assigning costs After the tree is pruned, a breadth-first search algorithm assigns the cost-to-go (CTG) for each vertex, assuming that the CTG of the root node is zero. The CTG_j of the vertex v_j is obtained by:

$$CTG_j = CTG_i + \mathcal{F}[\xi_i^j, \mathbf{F}_v]\,, \tag{5.2}$$



(a) (b)

Figure 5.6: Collision detection approach. (a) The lattice is triangulated to facilitate collision detection. The edges of the lattice are shown in black and the edges completing the triangles are shown in gray. (b) is a detailed view of (a). In red, the range sensor (e.g., LIDAR) measurements are transformed into circles in the sensor space that account for the robot dimensions. The triangles painted in light red overlap with these circles and have their edges pruned from the graph. The motion planning algorithm chooses the path, shown in green, from the remaining edges that minimizes the cost-to-go (CTG) from root to the outer layer.

where CTG_i is the CTG of the parent vertex v_i. The cost $\mathcal{F}[\xi_i^j, \mathbf{F}_v]$ measures the codirectionality of the vector field $\mathbf{F}_v$ and the edge ξ_i^j. For this, we use the upstream criterion, proposed in [140] and [51] as the following functional:

$$\mathcal{F}[\xi_i^j, \mathbf{F}_v] = \int_0^1 \left(1 - \frac{\xi_i^{'j}(s)}{\|\xi_i^{'j}(s)\|} \cdot \frac{\mathbf{F}_v(\xi_i^j(s))}{\|\mathbf{F}_v(\xi_i^j(s))\|}\right) \|\xi_i^{'j}(s)\| ds\,, \tag{5.3}$$

where $\xi_i^{'j}(s) = \partial \xi_i^j / \partial s$ is the first derivative of the edge with respect to the spatial parameterization variable s. Notice that, in our case, the derivative of each edge is constant. Using this functional, the cost for the case in which the path is parallel to the field is zero, and the cost for the anti-parallel case is equal to twice the length of the path. Hence, the CTG of each node in the tree will be as low as the path from the root to that node is "close" to the vector field.

Searching for the optimal path After costs are assigned for each reachable vertex of the tree (some vertices are not reachable after the tree is pruned), the vertices that belong to the outer layer are sorted by their CTG. By selecting the node with minimum cost and following the back-pointers of the tree to the root node, we extract the best path of the tree. If all the vertices in the outer layer are not reachable, the precedent layer is searched for the minimum value. The process is repeated if no vertices are reachable in this layer. A minimum cost can always be found because this procedure can be repeated until the root vertex (robot's current position), which has zero CTG. In this case, the robot will stop and return failure.

5.3.2 Parallelization

Due to its structure, the parallelization of the SSLAT algorithm is very natural. Its serial implementation has two bottlenecks: tree pruning and cost assignment. In this chapter, we focus on the parallelization of the pruning function, more specifically, in its implementation using the parallel computing platform CUDA [141] for computation using a graphics processing unit (GPU). By using CUDA, it is possible to define *kernels*, an expanded version of C++ functions that can be called in parallel by a determined number of threads. Threads are grouped into *blocks*, and a group of blocks forms a *grid* [142]. In the algorithm proposed in the last section, each valid LIDAR beam is tested for collision with each triangle in the lattice. Each of the laser measurements can be tested independently. So, to make the algorithm parallel, we create a kernel that takes as inputs the coordinates of the range measurements, the coordinates of the triangles, and the pointer to a list of lengths equal to the number of triangles that contains binary data representing the collision state. The list is kept in global memory and initialized with 0 (false), i.e., no collision. The kernel runs in parallel, and each thread will either do nothing if no collision is detected or change the data of the respective triangle in the list to 1 (true) if it detects a collision.

The number of blocks per grid and threads per block executed in parallel must be passed

to the kernel. On the GPUs that are available at this time, a thread block is limited to a maximum of 1024 threads. The number of triangles is not limited since it varies according to the parameters chosen to generate the lattice. The number of laser beams is tied to the sensor hardware (e.g., it is usually limited to less than 1024 for conventional planar LIDARs). Hence, in this work, we chose each block to execute $N_{threads}$ threads, where $N_{threads}$ corresponds to the number of valid laser beams, and execute tests on triangles on a grid with N_{blocks} blocks, where N_{blocks} is the number of triangles. This architecture can be seen in Figure 5.7. Both grid and blocks are chosen to be uni-dimensional.

The kernel operates three checks: 1) if the vertices of a given triangle are within the circle designated by the range measurement, 2) if the circle center is within the triangle, and 3) if the circle intersects any of the edges of the triangle, as described in the subsection 5.3.1. If any of these checks is true, it changes the status of the triangle to 1, denoting the existence of collision.

Disregarding the memory copy time of the LIDAR scans from the host (CPU) to the device (GPU) and the output back to the host, all collision checks are expected to be performed at roughly the same time as one single collision check. Nevertheless, the memory copy can be considered a significant part of the time spent in the parallel implementation. Our implementation, evaluated in the next section, uses Allocate Unified Memory, accessible from CPU or GPU. This further simplifies the parallel implementation.

5.4 Results

This section presents simulations and real-world experiments that illustrate and evaluate the proposed strategy. The experiments shown in this section include simulations in very dense forests and a real-world demonstration with an iRobot Create robot localizing itself using Simultaneous Localization and Mapping (SLAM).

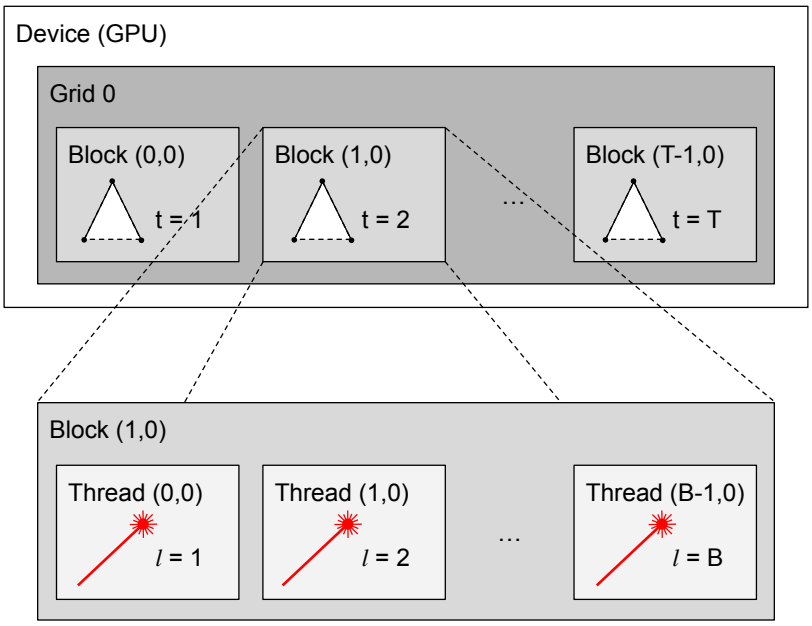


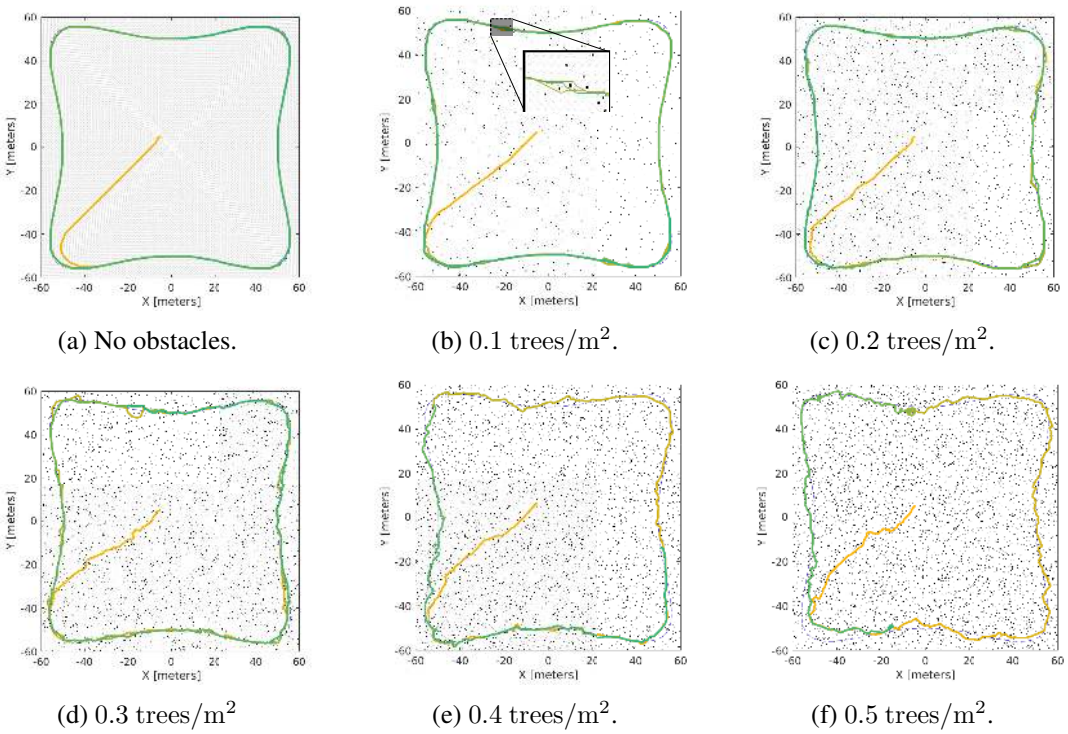
Figure 5.7: Parallel collision check approach. Triangles are organized in blocks, and laser scans are organized in threads of these blocks. All collision checks are run simultaneously.

5.4.1 Simulation with a Holonomic Robot in a Static Environment

Our methodology was first implemented in MATLAB®. We have used the Navigation Toolbox implementation of a range sensor to simulate a planar robot moving in a squared forest with $120\,\mathrm{m}$ side. Trees of diameter $r_{\text{tree}} = 0.1\,\mathrm{m}$ were distributed using a Poisson distribution at different densities, as proposed in [143]. The number of LIDAR beams was 1024; the sensor range was $10\,\mathrm{m}$, and the measurement error was $0.01\,\mathrm{m}$.

Once a path is computed as a sequence of nodes in the lattice, the robot follows only part of this path, which we call a committed path. In our simulations, we chose the committed path to include the first node of the planned path only. The travel distance was obtained by using a reference vehicle speed of $10\,\mathrm{m/s}$ and time equivalent to the processing time needed for the path generation. We ran our simulations in an Intel® Core™ i7-4700MQ $2.40\mathrm{GHz} \times 8$ CPU. Figure 5.8 shows the computed collision-free paths for 2000 iterations of the simulation for different forest densities.

Table 5.1 shows metrics regarding the planner and the path found for each density in Figure 5.8. As expected, tree density did not greatly affect the processing time for com-



(a) No obstacles. (b) 0.1 trees/m^2. (c) 0.2 trees/m^2.

(d) 0.3 trees/m^2 (e) 0.4 trees/m^2. (f) 0.5 trees/m^2.

Figure 5.8: Computed paths for a simulated forest environment, with different obstacle densities. The artificial vector field used for all the simulations is shown in (a), and its construction is detailed in [114]. (b) shows a detailed view of the robot paths in the three times it crosses the same region.

puting the paths in each iteration (ideally, it should only depend on the lattice parameters and the number of laser beams). In fact, larger computation times for the first lower density values are due to vector field computation inside the functional since fewer edges are pruned when the environment is free. The table also shows that the cost of the actual path (computed using (5.3)) increased with the density, indicating an increasing difficulty in following the vector field as the environment gets more cluttered, as expected.

5.4.2 Simulation of a Ground Mobile Robot in Static Environments

This subsection shows the potential of our methodology to guide a ground robot in an environment with obstacles. We used part of the Benchmark for Autonomous Robot Navigation (BARN) dataset [144] to compare the method's performance with other strategies. Al-

Table 5.1: Simulation metrics: processing time and cost.

Tree Density [**trees**/m^2]	**Performance parameters**			
	Time per it. [ms]	***Total time*** [s]	***Cost per it.*** [mm]	***Total Cost*** [m]
0.0	82.1 ± 4.6	164.2	6.4 ± 5.7	12.9
0.1	76.2 ± 6.5	152.4	9.9 ± 20.9	19.8
0.2	70.8 ± 5.7	141.6	14.6 ± 29.5	29.3
0.3	66.6 ± 3.8	133.3	26.1 ± 53.1	52.2
0.4	66.0 ± 3.5	132.0	119.2 ± 118.5	238.5
0.5	65.6 ± 4.5	131.2	134.7 ± 139.3	266.4

though this dataset is meant to be used by complete navigation stacks that include mapping and global motion planning, we found that SSLAT can find solutions for most environments tested.

Experimental Setup

The BARN dataset is configured for a Clearpath's Jackal robot with dimensions $508 \times 430\,\mathrm{mm}$. This robot is simulated using Gazebo [145] and the Robot Operating System (ROS) version Noetic [146]. The robot is equipped with 270° field of view planar LIDAR. The proposed methodology was implemented in C++ and made publicly available at `https://bitbucket.org/wvufarolab/sslat_planning/`. All tests of this section were executed on a computer with an Intel® Core™ i9 at $3.6\,\mathrm{GHz}$, sixteen cores, and $32\,\mathrm{GB}$ of main memory, equipped with an NVIDIA® GeForce® RTX 2080 with compute capability 7.5, GPU Clock of 1.5GHz, and Global Memory Size of $8\,\mathrm{GB}$. The maximum grid dimensions are [2147483647, 65535, 65535], while the maximum block dimensions are [1024, 1024, 64] with a maximum number of threads per block of 1024. The dataset consists of 300 randomly generated environments of increasing difficulty according to 5 proposed metrics: distance to closest obstacle, average visibility, dispersion, characteristic dimension, and tortuosity. An example of a simulation environment provided by the dataset is shown in Figure 5.9.

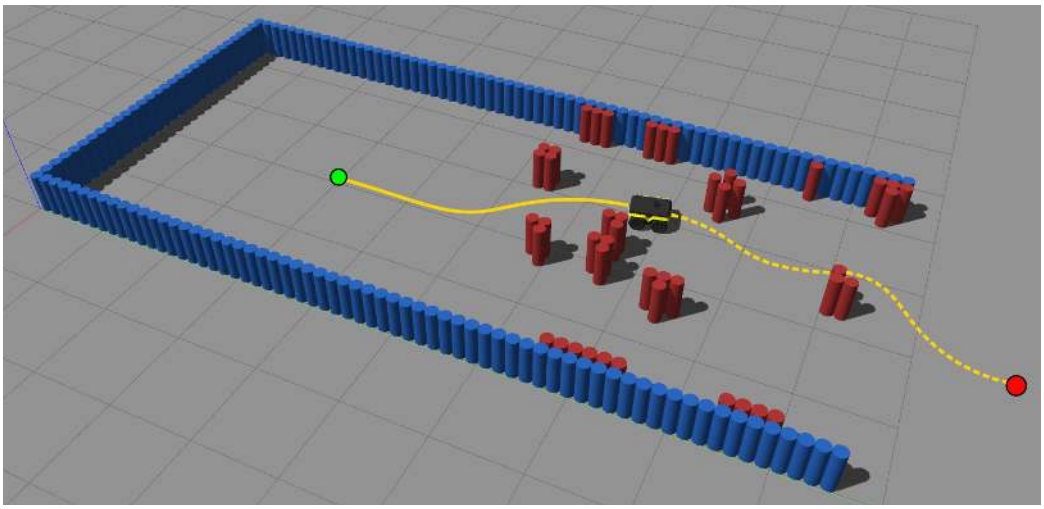


Figure 5.9: Environment #6 of the BARN dataset [144]. The objective is to exit the enclosed environment (blue cylinders), going from the start position (green dot) to the goal region (red dot) by traversing the cluttered area (red cylinders) as fast as possible.

To test the proposed methodology, from the start position $(x, y)_g = (0, 0)$, the robot was directed to the goal position $(x, y)_g = (0, 10)$ using a constant vector field that sends the robot to go straight in the y direction, $(u, v)_g = (0, 1)$, while crossing the field of obstacles, and another vector field pointing towards the goal afterward. The lattice was configured using $(K, N_T, N_B, N_L, r_0) = (2, 16, 3, 3, 0.4)$ and a collision radius of $0.35\,\mathrm{m}$. The robot was controlled using an in-house developed path-following node that generates forward linear velocity (v_x) and yaw angular rate (ω_z). Figure 5.10 shows four simulation snapshots showing the robot successfully traversing the obstacle field. This figure also shows the vector field, the sensor measurements, the instantaneous local plan provided by SSLAT, and the path executed.

We simulated five runs of each of the first 100 environments of the BARN dataset using two different robot speeds: 0.50 and $1.15\,\mathrm{m\,s^{-1}}$. For these speeds, the optimal time the robot would take to go from the initial position $(0, 0)$ to the goal area (a $1\,\mathrm{m}$ radius centered in $(0, 10)$) are 18.0 and $7.8\,\mathrm{s}$, respectively, if the robot moves in a straight line. To compare our results, we also tested two baseline algorithms provided by the authors of the dataset, Dynamic-Window Approach (DWA) [121] and Elastic Bands (EBand) [116], using ROS

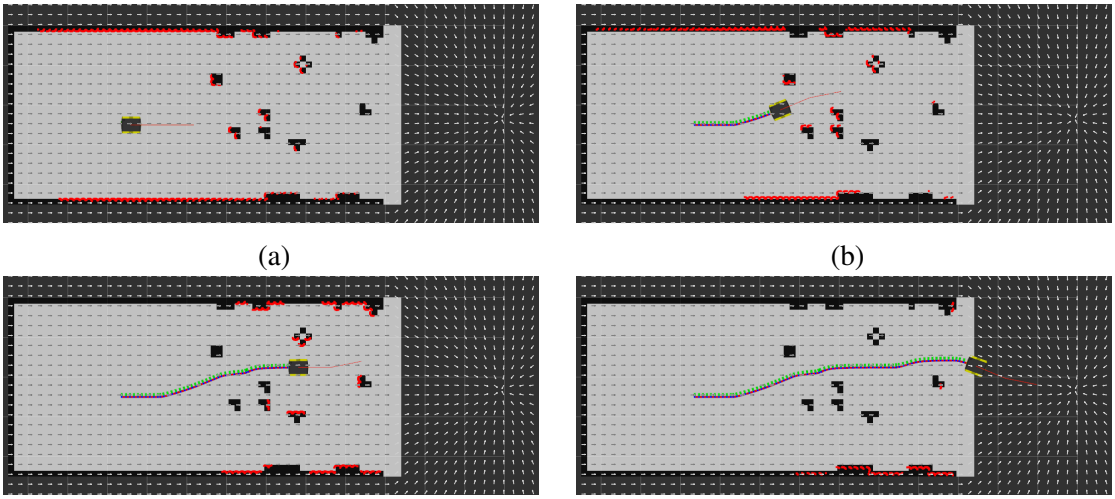


Figure 5.10: Snapshots of the solution provided by SSLAT in environment #6 of the BARN dataset (see Figure 5.9). From (a) to (d), four snapshots are provided. The planned path is shown in red. A sequence of reference coordinate frames shows the executed path. A close inspection shows that the robot's path follows the vector field whenever there are no interfering obstacles. Notice that the occupancy grid of the environment is shown to illustrate the scenario, and it is not used by SSLAT.

move_base as a navigation framework. In this framework, a higher-level planner (A* in our experiments) is responsible for planning a path on a map created online using Simultaneous Localization and Mapping (SLAM). DWA and EBand thus act as lower-level approaches that follow the path while avoiding previously undetected obstacles. The experiments were performed without tuning the parameters of each method for each environment.

Experimental Results

Figure 5.11 shows the traversal times for each of the methods using 0.50 (a) and $1.15\,\mathrm{m\,s^{-1}}$ (b). In this figure, runs for which collisions happened were given a total traverse time of $50\,\mathrm{s}$, as proposed by the authors of the dataset. Tests would also timeout at $50\,\mathrm{s}$ if the planners took too long to find a solution. Table 5.2 shows numerical details about the experiment.

Figure 5.11 and Table 5.2 show that DWA, EBand, and SSLAT consistently find solutions for the environments tested. DWA presents a great time dispersion since it frequently

Table 5.2: Comparison of three motion planning strategies using the first 100 environments of the BARN dataset [144]. Each method was run five (5) times for each environment. The mean and standard deviation values only consider successful (no collisions/no timeout) trials.

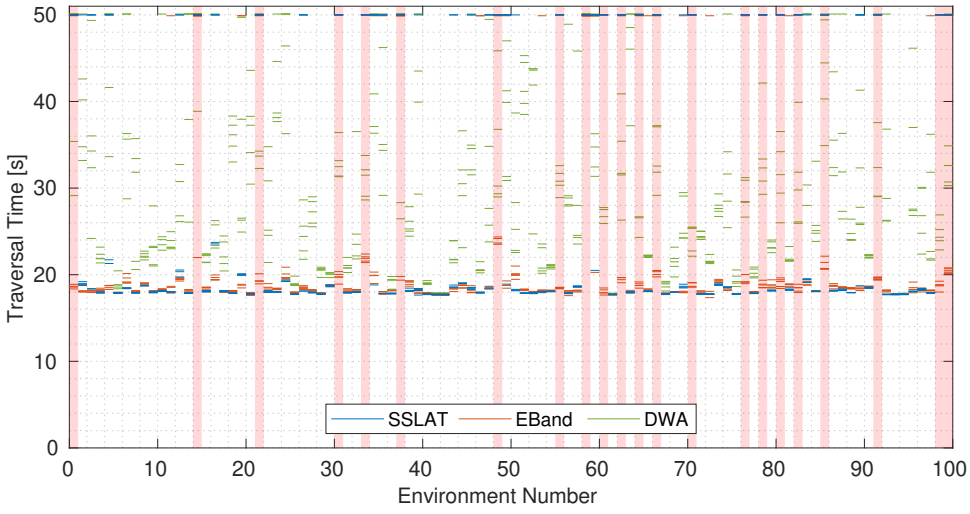
Method	**Robot Speed** [$\mathrm{m\,s^{-1}}$]	**Success Rate [%]**	**Envs. with Success in All Five Trials [%]**	**Envs. with Failure in All Five Trials [%]**	**Traversal Time** [s]	
					Mean	**Std. Dev.**
SSLAT	1.15	71.8	10.6	4.4	8.540	2.000
	0.50	69.4	12.0	28.2	18.722	2.495
EBand	1.15	93.4	16.4	0.4	8.540	0.931
	0.50	97.2	18.4	0.0	18.604	0.945
DWA	1.15	93.6	15.6	0.0	19.787	8.478
	0.50	97.4	18.2	0.0	27.604	7.493

slows down the robot to find solutions. EBand and SSLAT can obtain results close to the optimum (18.0 and $7.8\,\mathrm{s}$) when they find a solution. However, as shown in Table 5.2, none of the methods had 100% success in traversing all the environments. SSLAT cannot solve every environment mostly because it does not have enough information to deal with dead ends (the vector field, which acts as a higher-level planner, does not use a map and ignores all the obstacles). The other two methods rely on a global path planner, so they were expected to perform better in the dataset.

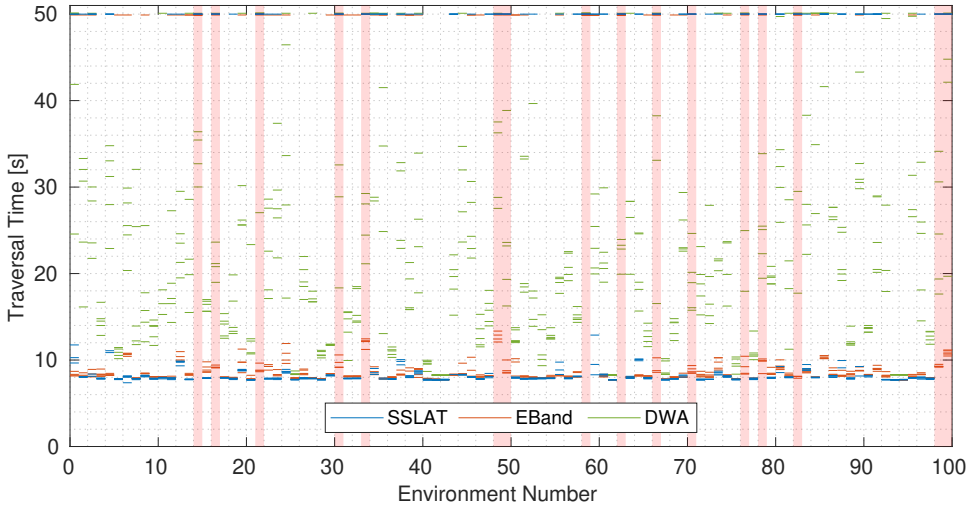
Besides task completion time, we also recorded the planning time for SSLAT and EBand. SSLAT has a maximum planning time of $3.28\,\mathrm{ms}$ while EBand has a maximum planning time of $4.19\,\mathrm{ms}$.

5.4.3 Simulations of a Ground Mobile Robot in a Dynamic Environment

This subsection shows the performance of SSLAT in a modified version of the BARN environment, which was adapted to contain moving obstacles.



(a)



(b)

Figure 5.11: Comparison of three motion planning strategies using the first 100 environments of the BARN dataset [144]: SSLAT (blue), EBand (red), and DWA (green). In (a) the robot speed is limited to $0.50\,\mathrm{m\,s^{-1}}$ while in (b) it is limited to $1.15\,\mathrm{m\,s^{-1}}$. The results for the 5 runs of each environment are shown for each method. In case of collision or time-out, the traversal time was set to $50\,\mathrm{s}$. The highlighted environments are the ones for which SSLAT could not find a solution.

Experimental Setup

In the environment used in this section, four moving cylinders were placed between the start and goal positions, as shown in Figure 5.12. The cylinders move side-to-side, each with a constant speed sampled uniformly from a speed interval at the beginning of each

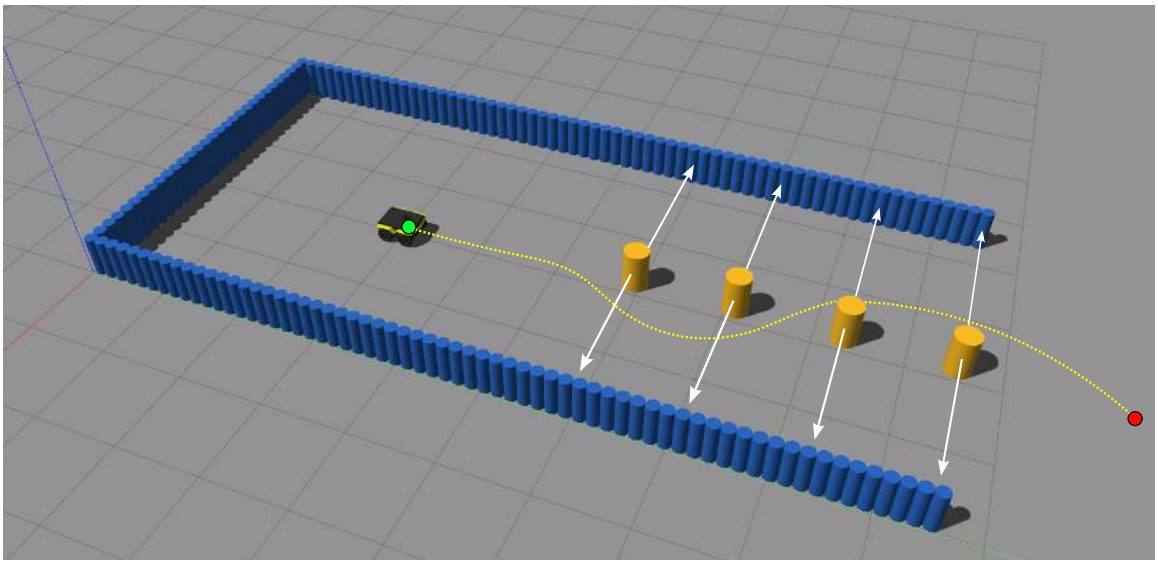


Figure 5.12: Environment used to test obstacle avoidance with dynamic obstacles. While the robot moves towards the goal, the four yellow obstacles move back and forth with random velocities.

trial. Once the obstacles reach the wall, they switch directions, going back and forth. Since the robot is non-holonomic and cannot move sideways, the obstacles stop moving once the robot's sensor passes by them, thus preventing unavoidable lateral collisions caused by the obstacles. Similarly to the previous subsection, three methods were tested: SSLAT, EBand, and DWA. Each method was tested for two robot maximum speeds (0.50 and $1.15\,\mathrm{m\,s^{-1}}$) and three intervals for the speed of the obstacles: $[0.00, 0.75]\,\mathrm{m\,s^{-1}}$, $[0.25, 1.00]\,\mathrm{m\,s^{-1}}$, and $[0.50, 1.25]\,\mathrm{m\,s^{-1}}$. For comparison, a person tends to walk at around $1.3\,\mathrm{m\,s^{-1}}$ [147]. The same vector field and lattice configurations defined for the static environment in the previous subsection were used in the experiments of this subsection.

Experimental Results

The success rate of the three planners for all situations tested is shown in Table 5.3. This table shows successful situations, i.e., when the robot reached the goal without collision in less than $100\,\mathrm{s}$. We run each situation 50 times for each method. Table 5.3 also shows the average traversal time for all methods and its standard deviation. SSLAT could handle

Table 5.3: Performance comparison of three motion planning strategies in the presence of moving obstacles, as shown in Figure 5.12. The speed of the obstacles is randomly sampled from the following intervals: Low $[0.00, 0.75]\,\mathrm{m\,s^{-1}}$; Mid $[0.25, 1.00]\,\mathrm{m\,s^{-1}}$; Fast $[0.50, 1.25]\ \mathrm{m\,s^{-1}}$

Method	**Robot Speed** [$\mathrm{m\,s^{-1}}$]	**Obstacle Speed** [$\mathrm{m\,s^{-1}}$]	**Success Rate [%]**	**Traversal Time** [s]	
				Mean	**Std. Dev.**
SSLAT	0.50	Slow	72	19.428	1.146
		Mid	38	19.661	0.899
		Fast	44	19.951	0.797
	1.15	Slow	72	8.960	1.140
		Mid	60	9.090	1.060
		Fast	58	8.970	0.890
EBand	0.50	Slow	58	19.041	0.918
		Mid	32	19.860	1.128
		Fast	24	19.931	0.910
	1.15	Slow	58	8.800	0.710
		Mid	66	8.910	0.640
		Fast	46	9.230	0.870
DWA	0.50	Slow	82	21.625	3.785
		Mid	62	20.833	2.230
		Fast	58	20.845	1.724
	1.15	Slow	92	11.730	4.330
		Mid	82	10.310	1.870
		Fast	80	10.660	1.450

moving obstacles better than EBand but was overcome by DWA, as seen by the success rate. Notice that this happens at the expense of an increased traversal time by DWA, which reduces the robot's speed to avoid the obstacles, while SSLAT and EBand try to keep the robot at maximum speed. One reason is that SSLAT does not have memory, runs very fast, and replaces the entire navigation stack used by EBand. Therefore, once an obstacle is detected, it is instantly considered in the planning. The other two methods are dependent on an occupancy grid and global plan generated by the navigation stack they use (*move_base*). Therefore, as costmaps are not updated fast enough, they are more prone to fail due to cells that are marked as occupied because obstacles have been there in the past. Further, it is noticeable that all three methods are susceptible to failure due to the slow dynamics of the robot, i.e. even if the plan is updated fast, the robot does not necessarily follow that change.

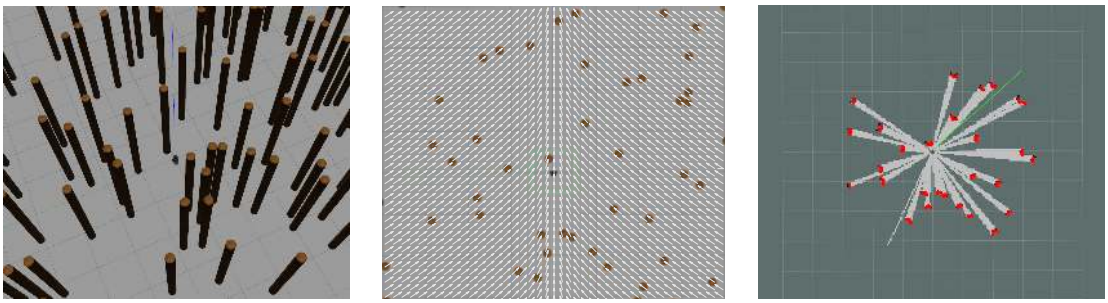
5.4.4 Validation and Evaluation of the Parallel Implementation

This subsection presents the results of a ground robot navigating in a forest-like environment using the proposed methodology. Our goal in this section is to evaluate the parallelization of the tree-pruning step of our algorithm by analyzing the effects of changing the density of obstacles in the environment and the number of triangles on the lattice.

Experimental Setup

In this subsection, SSLAT was tested on a laptop computer with an Intel® Core™ i7 at $2.4\,\mathrm{GHz}$, eight cores, and with $16\,\mathrm{GB}$ of main memory and an NVIDIA® GeForce® GT 740M with compute capability 3.5, two CUDA Cores, GPU Clock of 1GHz, and Global Memory Size of $2\,\mathrm{GB}$. The maximum grid dimensions are [2147483647, 65535, 65535], while the maximum block dimensions are [1024, 1024, 64] with a maximum number of threads per block of 1024. The total amount of shared memory per block is $48\,\mathrm{kB}$, and per thread is $16\,\mathrm{kB}$. Notice that, differently from the powerful desktop computer used in the previous section, the computer used in this section could be carried by a ground robot in real life.

Our experiments were executed using Gazebo simulations and ROS Melodic. A simulated ground robot was equipped with a planar LIDAR in this test. The LIDAR was set to have $3.5\,\mathrm{m}$ of range and 360 laser beams ($1\,\mathrm{deg}$ resolution). Figure 5.13 shows a typical environment, the vector field used to make the robot move across the forest, and an example laser scan measurement obtained while the robot moves amidst the obstacles. The obstacles consist of cylinders randomly placed on the map to simulate a Poisson forest of different densities, as proposed by [143]. In our experiments, we varied the density of the forest from 0.1 to 3.2. To vary the number of triangles in our method, the number of trunks, N_T, was changed from 8 to 64 in powers of 2 and the number of layers, N_L, varied between 4 and 5. All the other parameters were kept constant as $(K, N_B, r_0) = (2, 3, 0.4)$. The collision detection radius was set to $0.20\,\mathrm{m}$.



(a) Gazebo environment (b) Vector field (c) Laser scan

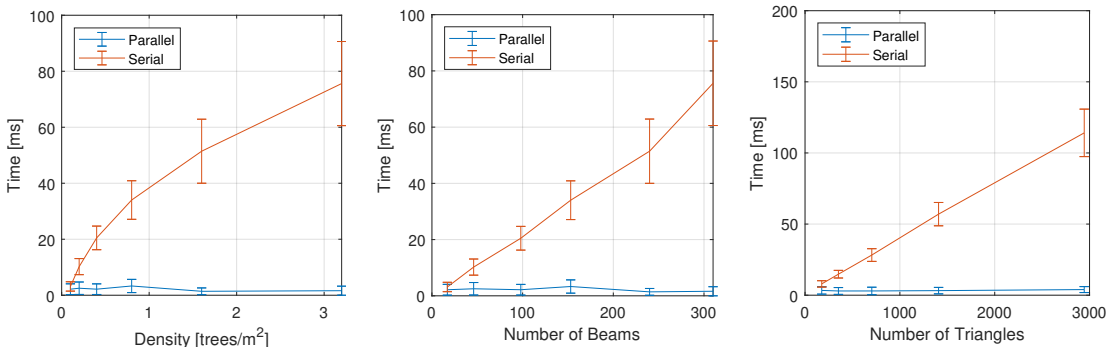
Figure 5.13: Experimental setup used to compare the performance of the parallel and serial versions of the SSLAT motion planner. In (a), the environment were constructed using cylindrical objects in Gazebo, to simulate a forest. The global task is encoded by the vector field in (b), which follows a straight line in the vertical direction. In (c) it is possible to see, in a specific robot position, the valid LIDAR scans (i.e., within the sensor range) used to prune the search tree.

Experimental Results

Figure 5.14 shows the effect of the forest density and the number of triangles in the lattice on the collision-check time used for pruning the search tree. In Figure 5.15 we have similar graphs for the complete algorithm.

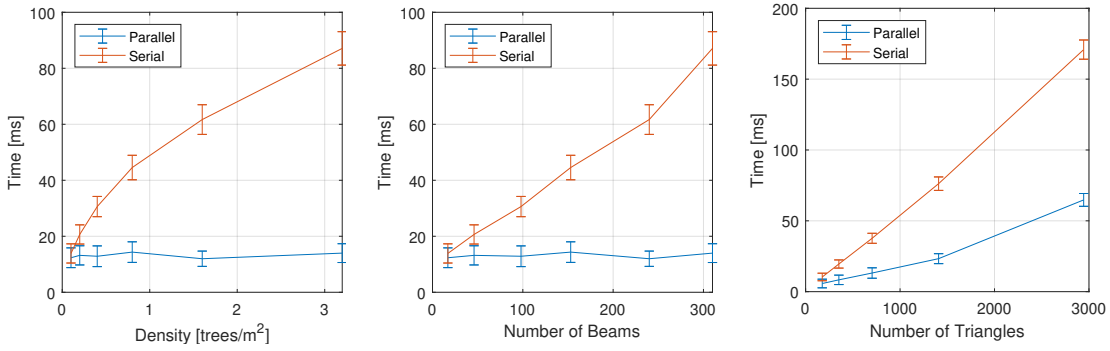
As the density increases more laser beams are considered valid causing the collision checking time in the serial implementation to grow monotonically. Notice in Figure 5.14(b) that the time grows almost linearly with the number of valid LIDAR beams. On the other hand, the parallel version maintains an almost constant time, independently of the density, number of beans, and number of triangles. As a result, for large obstacle densities, the speed-up of the parallel implementation is greater than 25.

Notice that, for the complete algorithm (Figure 5.15) the effect of the forest density leads to conclusions similar to the observed for the collision-checking time (Figure 5.14). However, an important difference in the graph measures the effect of the number of triangles, where the total time also increases in the parallel version. This is because cost assignment, which was not parallelized in our implementation, grows linearly with the number of edges. This suggests that future implementations should also consider the par-



(a) (b) (c)

Figure 5.14: Effect of the lattice parameters and environment density on the tree pruning time for both the serial and parallel implementation of the method. (a) Effect of the density of the forest; (b) effect of the number of laser beams; (c) effect of the number o triangles.



(a) (b) (c)

Figure 5.15: Effect of the lattice parameters and environment density on total planning time for both the serial and parallel implementation of the method. (a) Effect of the density of the forest; (b) effect of the number of laser beams; (c) effect of the number o triangles.

allel implementation of this step, which can be easily accomplished using the algorithm presented in [148].

5.4.5 Experiments with a Real-World Ground Mobile Robot

Experimental Setup

We implemented and tested our motion planner in iRobot's Create 2. This differential-drive robot has $0.34\,\mathrm{m}$ of diameter, a maximum speed of $0.5\,\mathrm{m/s}$ and a maximum turning radius of $2\,\mathrm{m}$. The robot was equipped with the YDLIDAR X4 360-degree planar LI-

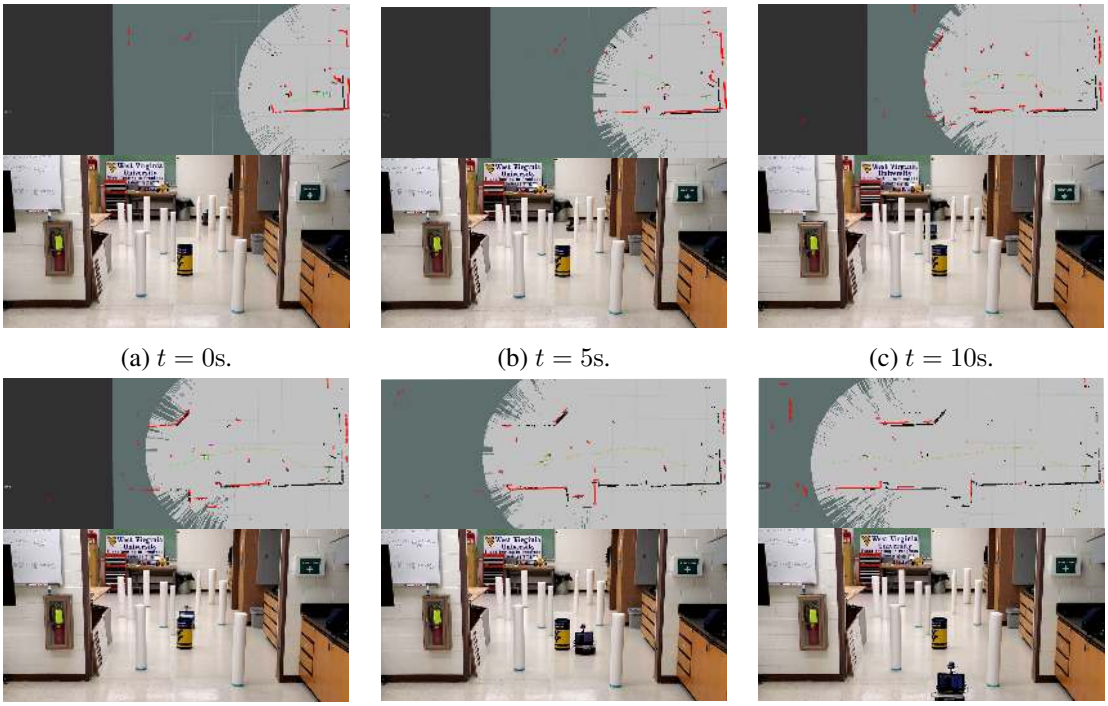
DAR. This sensor has a range of $10\,\mathrm{m}$, with scan frequency up to $12\,\mathrm{Hz}$. The angular resolution is $0.50 \pm 0.02°$. The robot is controlled by a laptop with an Intel® Core™ i3 $1.8\mathrm{GHz}$ CPU running Linux Ubuntu 18.04 and ROS Melodic. We used the SLAM package `gmapping` [149] for localization, necessary to compute the vector field. We controlled the robot using a simple ROS node developed in-house. Our planner was implemented in C++ and wrapped in a ROS node.

Our tests were performed indoors using cylindrical obstacles randomly placed in our laboratory. These tree-like obstacles are supposed to simulate a cluttered forest environment. The obstacles diameters ranged between $100\,\mathrm{mm}$, and $350\,\mathrm{mm}$. In our tests, the lattice was generated using the following parameters: $(K, N_T, N_B, N_L, r_0) = (2, 16, 3, 3, 0.4)$, which was shown to be adequate to the room and obstacle sizes. The planning range (lattice outer-layer radius) is $1.6\,\mathrm{m}$, smaller than the LIDAR range. Therefore, any measurement greater than the lattice range was disregarded in our implementation to avoid unnecessary computation of ranges far from the region of interest (as our laboratory walls, for example). The vector field used was the one used to follow a straight line, as exemplified in the methodology section, with convergence parameter $c = 2$.

Experimental Results

The robot was able to navigate at its maximum speed through the forest-like environment and snapshots of the robot navigation for one of our trials are shown in Figure 5.16. The path executed by the robot during the same trial and the distribution of the obstacles in the map is depicted in Figure 5.17. Notice that the robot tries to return to the preferred path (straight line) every time it can, but the cylinders repel it. When there are no more obstacles (Figure 5.16(f)), the robot successfully returns to the desired route.

The time necessary to generate the paths compared to the sensor frequency indicates the performance of our planner. While the data coming from the LIDAR is streamed at a period of $200\,\mathrm{ms}$, the paths were generated at $32.85 \pm 7.45\,\mathrm{ms}$ (collision: $30.20 \pm 7.64\,\mathrm{ms}$,



(a) $t = 0$s. (b) $t = 5$s. (c) $t = 10$s.

(d) $t = 15$s. (e) $t = 20$s. (f) $t = 27$s.

Figure 5.16: Snapshots of a real mobile robot experiment using cylinder obstacles. On the top, we show the constructed map, the real-time path planned (in green), and the laser scans (in red). On the bottom, the robot moves toward the reader, avoiding obstacles (white cylinders and the WVU bucket).

cost assignment: 2.52 ± 0.42 ms, search: 0.12 ± 0.01 ms)[1].

To compare the efficiency of our method with RRT*, used in [51] to optimize the field-based functional, we run both approaches in a fixed obstacle configuration. We used the RRT* implementation provided by OMPL [150]. Since RRT* is an anytime probabilistic planner, we ran it 100 times for 30 ms (the time spent by our method in the real-robot experiment) and one time for 3 s, which would represent a path very close to the optimal. Our method spent 21.91 ms in the same conditions to compute a path. Figure 5.18 shows the result of this experiment. Notice that the path computed by our approach (blue) is similar to the optimal one (dashed red), while the paths computed by RRT* with a strict deadline (shown in gray) present a large variation. This indicates that our method would be a better choice when a short time is available for path planning.

[1] A video with simulations and experiments can be found at: https://youtu.be/Axn7XRimgFU.

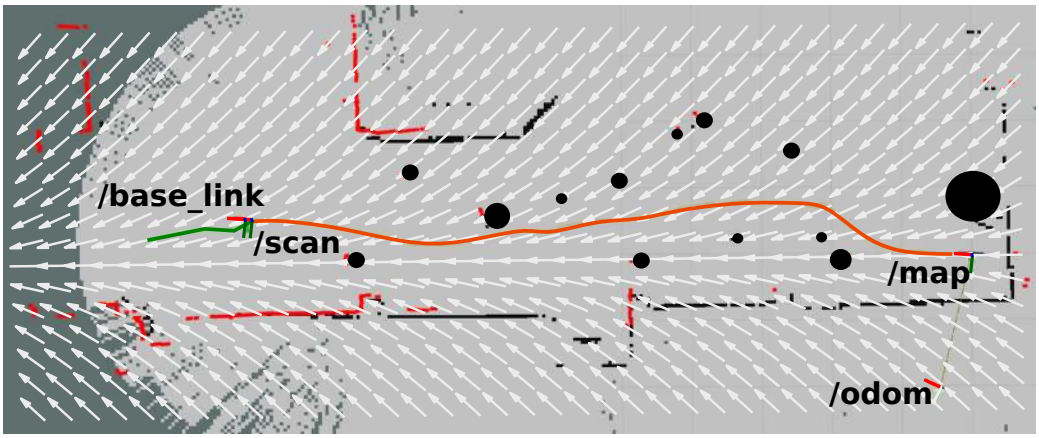


Figure 5.17: Path followed by the mobile robot in a real-world experiment. The vector field is given as white arrows; the path executed by the robot as an orange line; the next planned path as a green line; instantaneous lase scans as red dots; and the obstacle, as black dots. The $/map$ frame coincides with the robot's starting position. The $/odom$ frame is initially aligned with the $/map$ but drifts during execution. SLAM computes the transformation between them. The $/scan$ frame is at the origin of the sensor space, and the $/base\_link$ is at the center of the robot.

5.4.6 Experiments with a Real-World UAV

This subsection presents a real-world experiment that shows SSLAT flying a drone under the canopy of a sparse forest.

Experimental Setup

The robot used in this section is a DJI Matrice 100 quadrotor. This drone has a built-in navigation solution based on the Global Positioning System (GPS) and Inertial Measurement Unit (IMU) that provides vehicle localization at $50\,\mathrm{Hz}$. The drone is actuated by its linear velocities and angular rate. For our experiment, the drone was equipped with a UDOO X86 (CPU Intel Pentium N3710 2.56 GHz, 8G RAM) single-board computer running Linux, which communicates with the drone using a ROS driver provided by the maker. Since the drone's onboard computer is not compatible with CUDA, our experiments run the sequential, non-parallel version of SSLAT. Besides the computer, our drone was equipped with an Intel® RealSense™ D435i camera, which provides depth images at $30\,\mathrm{Hz}$ since

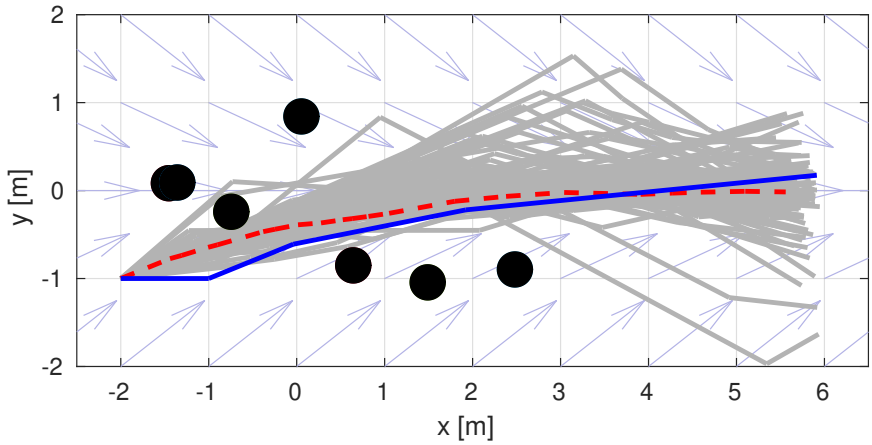


Figure 5.18: Comparison with RRT*. In light blue, the vector field. In blue, the planned path using our motion planner. In red, the optimal path generated by RRT* in 3 s. In gray, paths generated by RRT* in 30 ms. Obstacles are shown in black.

our algorithm expects 2D LIDAR data, a ROS node (*depthimage_to_laserscan*) was used to transform the camera output into the required input for our software.

In our experiment, we designed a vector field to circulate a closed curve shaped as a square with rounded corners and parameterized by the potential field $\phi(x, y) = (x/10)^4 + 0.5(x/10)^2(y/10)^2 + (y/10)^4 - 1$ (see Figure 5.19(g)). The computation of such an artificial vector field is shown in [139]. The lattice was configured with $(K, N_T, N_B, N_L, r_0) = (2, 16, 3, 3, 0.5)$ and a collision radius of $0.60\,\mathrm{m}$.

Instead of using a path-following controller, in this experiment, the drone was controlled using linear horizontal velocities (v_x and v_y) and yaw angular rate (ω_z) obtained from the first waypoint of the path planned by SSLAT, represented by (x_{sp}, y_{sp}) in the robot reference frame. The height was kept constant. The velocities commanded to the drone were computed as:

$$v_x = V\frac{x_{sp}}{\sqrt{x_{sp}^2 + y_{sp}^2}}\,, \quad v_y = V\frac{y_{sp}}{\sqrt{x_{sp}^2 + y_{sp}^2}}\,, \quad v_z = 0\,, \quad \omega_z = K_\psi \psi_{sp}\,, \tag{5.4}$$

where V is the constant linear speed, set to be $1.5\,\mathrm{m\,s^{-1}}$. The angular speed was set to be proportional to the error in yaw angle given by $\psi_{sp} = \arctan 2(y_{sp}, x_{sp})$. This maintains the robot pointing to the direction of the planned path, which is important due to the limited

field of view of the sensor used ($87°$).

Experimental Results

Figure 5.19 shows six snapshots of our experiment along with the drone's trajectory obtained by its localization system. Since the forest was sparse and had almost no leaves, GPS information was available during the entire flight. The results of Figure 5.19 show that the drone successfully tracked the target curve guided by the vector field, avoiding the trees when necessary. This suggests the proposed approach can provide safe motion at reasonably high speeds ($1.5\,\mathrm{m\,s^{-1}}$), even in challenging outdoor environments such as a forest.

5.5 Discussion

Our experimental results show that SSLAT has the potential to be used to guide fast robots to perform a variety of tasks. We benchmarked SSLAT against two well-known and optimized strategies available in ROS. Our numerical results showed that SSLAT, using a very simple vector field as a global planner, is slightly inferior then EBand when dealing with static environments but much faster than DWA in the same situations. At this point, notice that the success rate of our planner could be improved if we had opted to create a vector field using knowledge about the environment, which would be equivalent to the high-level planner used by EBand and DWA. On the other hand, in the presence of moving obstacles, the success rate of SSLAT is greater than the one observed for EBand and is less than the one observed for DWA, which tends to reduce the robot's speed to avoid obstacles.

Our results also show that a robot running SSLAT (and also EBand and DWA) may not be able to avoid every possible collision, indicating that it would need to be complemented by other strategies. In the simulations presented in this chapter, the robot is not equipped, for example, with an emergency-stop behavior that would stop the robot in the eminence of

a collision. Conversely, SSLAT tries to keep the robot's velocity at its maximum to reduce the time it traverses the environment. This behavior is risky and would not be acceptable in most real-world applications. Additionally, we notice that although SSLAT computes a free-of-collision path at every iteration, the controller cannot always follow the path. This suggests that better path-following controllers need to be used.

The main goal of the experiments presented in this chapter was to show the potential of SSLAT for fast path computation. Although the maximum planning times for both SSLAT and the baseline, EBand, are very small, indicating that both strategies can be used in hard real-time, notice that EBand was running on an Intel i9 CPU in our experiments, which may not be available for most robots. On the other hand, since the most expensive routine of SSLAT runs on the GPU, we expect that its planning time will not increase too much when it runs on embedded GPUs, such as, for example, the NVIDIA® Jetson®.

Our last experiment shows SSLAT controlling a drone in a challenging forest environment with environmental disturbances (i.e., wind, shades, direct sunlight). Although we have not explicitly considered uncertainties, disturbances, and failures in our method's design, SSLAT is inherently robust to such input issues once it is a closed-loop approach at all its levels, and most importantly, it runs fast. First, notice that the vector field is closed-loop since we have a velocity vector for each position in the space. Moreover, most vector fields are stable in the sense of Lyapunov [109, 112]. In the avoidance level, any disturbances, such as new obstacles, moving obstacles, and changes in the vehicle's position due to wind, are rapidly detected in the lattice, and a new path is created. This process happens in a few milliseconds, enough to reject the most common disturbances. In addition to external disturbances, SSLAT presents some robustness to sensor uncertainties. It tests the collision between a circle, initially defined by the robot's radius and the triangles of the lattice, and the measurement from each sensor beam. By increasing the circle's radius, we can account for the uncertainty of the sensor, consequently being more conservative and moving farther from the obstacles. We can also observe some robustness to sensor failures that could lead

to missing obstacle detections. Because we may have several LIDAR beams associated with each triangle, a failure in a single beam would not be a problem, especially for obstacles close to the robot, since another beam could prune the edges of the graph, which would lead to collisions. Naturally, the method would fail if any beam does not detect an obstacle, which may happen when the obstacle is small and is far from the robot. This, however, is a limitation of the sensor and not of the method.

5.6 Summary

Motion planning in real-time is an essential requirement for the navigation of fast robots in cluttered environments. This chapter presented the SSLAT motion planner, an embarrassingly parallel strategy that computes local paths within the sensor space of a robot. Our current CUDA hardware could compute paths using SSLAT as fast as 2 ms (500 Hz). This is just a fraction of the sampling time of a typical range sensor, which operates in frequencies that range from 10 to 100 Hz. So, SSLAT can compute paths in hard real-time, making it a good candidate strategy for use in static and dynamic environments, especially in applications where a vector field can define the main direction of movement. Although this seems to be limiting, there are several common tasks, such as road or corridor following [43, 44], forest traversing [113], or perimeter surveillance [42], that lie in this category. Other clear examples were given in chapters 3 and 4 of this dissertation, where a policy vector field was created to navigate strong natural flows. In these cases, the method proposed in this chapter could be used to track the field while avoiding unmodeled obstacles and no-fly zones.

This chapter showed the application of SSLAT for ground and aerial vehicles operating in cluttered environments. By benchmarking SSLAT against baseline approaches using the BARN dataset, we showed that it is approximately 2.3 times faster than DWA, but, overall, it leads to approximately 21% more collisions than EBand while moving at similar speeds.

This occurs because EBand uses a global optimum planner, while the vector field used by SSLAT was set to be a simple one that only moves the robot forward. By comparing SSLAT with the same baseline approaches in a dynamic environment, we show that its success rate is greater than that of EBand (58% versus 46% for high robot and obstacle speeds), but it is less than that of DWA (58% versus 80%). However, we noticed that DWA is more conservative, leading to completion times approximately 19% higher than those achieved with SSLAT. Overall, we conclude that SSLAT would be a competitive choice if both static and moving obstacles must be avoided at fast speeds.

The next chapter concludes the dissertation by summarizing our results, discussing their strengths and limitations, and proposing opportunities for future research.

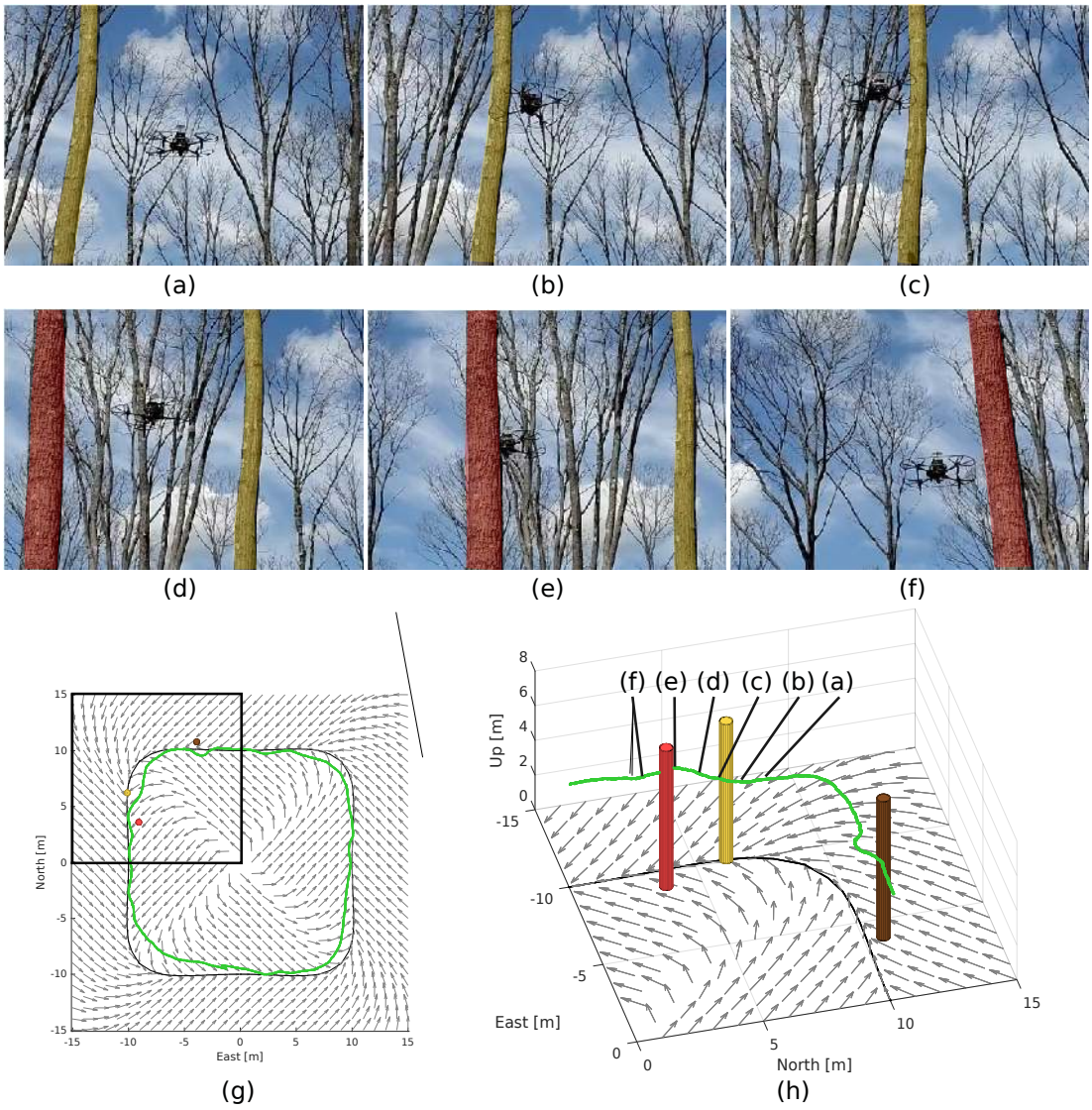


Figure 5.19: Results of an experiment performed with a drone in a forest. (a) to (f) shows snapshots of the part of the experiment highlighted in the black box of (g). In (g) and (h), the vector field is shown by the gray arrows, the black curve represents the target curve, and the cylinders represent trees. The cylinders were painted in yellow and red to facilitate the correspondence with the actual trees in snapshots (a) to (f), which were also painted with the same colors. The executed trajectory, obtained from the localization system of the drone, is shown in green. In this trajectory, the letters (a) to (f) correspond to the snapshots shown.

Chapter 6

Conclusions

This chapter summarizes the results and conclusions presented in this dissertation and presents a list of suggestions for future work.

6.1 Overview

This dissertation presents innovative approaches to solving complex robot motion planning problems using vector fields. In Chapter 3, a flow-aware sampling-based motion planner (FlowFMT*), based on the Fast Marching Trees algorithm, was proposed to find optimal paths in spatially-variant and time-invariant environmental flows, encoded as vector fields. The planning strategy accounts for limitations in reachability given by the ratio between the flow velocity and the maximum vehicle velocity relative to the flow and uses it to reduce the number of possible graph connections tested. The motion planning method is validated against the solution obtained from state-of-the-art optimal control solvers (OCS) for two- and three-dimensional environments and considers the possibility of restricted regions. Our results show that, in obstacle-free environments, OCS solutions tend to be faster after being adequately tuned, but they do not handle changes in homotopy properly. Although not explicitly tested in this work, it is expected that the gap in computational performance can be reduced or reversed with an increased number of obstacles. Finally, the method was

extended to generate a vector-field policy based on the local flow and direction of the tree. This policy can implicitly represent the motion plan to guide the vehicle to a desired goal. This vector field can be seen as a feedback law, introducing robustness to the vehicle's navigation. Our experiments show cases where the vehicle can recover from actuation failures and still follow optimal paths.

The method, however, still has a few limitations. We might encounter a case where we want to converge to an optimal solution faster and in a limited time. Rerunning the method with an increased number of particles is not a satisfactory solution, as the running time scales up rapidly with the number of samples (the FMT* algorithm is $\mathcal{O}(n \log n)$). A possibility for handling this issue is presented in Appendix A, where we show an any-time version of FlowFMT*. Another limitation of the method presented in Chapter 3 is that we considered a holonomic model for the vehicle. While this can be a reasonable assumption for slow-moving vehicles that traverse long distances, it is a limitation for more complex vehicle models. One option to extend the method for nonholonomic vehicles is to solve the system of equations that models the vehicle motion in terms of its linear and angular speeds. Appendix B presents a promising direction to solve this issue by expanding the geometric functions using Taylor series and solving an approximated system.

In Chapter 4, two solutions to the motion planning problem for an autonomous airship under super-rotation winds of the Venusian atmosphere were proposed. The model of the airship considers buoyancy and aerodynamic lift and drag to find the energy expenditure of the propulsive system. In addition, solar panels distributed over the aircraft provide energy to the propellers and allow for battery recharging. In our first approach, we used a sampling-based planner that we named EWRRT, which relies on Dubins' Airplane paths deformed under the influence of the winds to create a tree of kinematically feasible trajectories. We use the battery state to prune energetically unfeasible trajectories and propose a cost function that accounts for the energy expenditure of the propulsive system and considers battery charging by using the notion of cost of opportunity from the field of Economics.

The method was illustrated through a series of simulations that show how the vehicle takes longer and higher-altitude paths to minimize the use of energy and favor battery recharge. Even though the EWRRT incorporated a nonholonomic vehicle model, because the vehicle covered large distances, the importance of the nonholonomic behavior was reduced. With other limitations, such as lack of optimality, the method was put aside in favor of a second solution based on adapting the method described in Chapter 3 to solve the Venus navigation problem. This second method offered three main advantages.

The first advantage of using FlowFMT* is its capability to find optimal paths, which was not possible using EWRRT. The goal is to obtain paths that optimize a cost function that depends on the energy budget of the aircraft. Still, the EWRRT was only expanding the tree towards the direction of smaller costs without optimization due to a lack of tree rewiring. The second advantage is considering regions where the vehicle does not actuate. These regions are in place because the place is too dark for the robot's solar panels to generate enough power or because the robot is not allowed to operate in that region for some other reason (e.g., stealthiness). An adaptation was made to advance the vehicle state when the vehicle is not actuated (e.g., the battery ends). In these regions, the vehicle behaves as a pseudo-Lagrangian drifter, i.e., the flow rules its movement. This adaptation was validated using a more straightforward scenario where the vehicle must cross a jet flow. The third advantage is considering the entire environment for planning, not only a local chart. In general, since the dynamic programming step of the FlowFMT* is performed locally, we considered the geocentric representation of the world to expand the planning tree. However, local tangent frames and geodesic approximations were used during the non-actuating periods to make the vehicle follow the flow. Resulting time-optimal and energy-optimal paths of planet-scale planning are provided, confirming the preference to travel in higher altitudes and the possibility of reaching goal locations that were not possible without circumventing the planet.

In Chapter 5, a parallel motion planner for mobile robots and autonomous vehicles

based on lattices created in the sensor space of planar range finders was proposed to follow a vector field that encodes the global robot task. The planner can compute paths in a few milliseconds, thus allowing obstacle avoidance in real-time. The proposed sensor-space lattice (SSLAT) motion planner uses a lattice to tessellate the area covered by the sensor and rapidly compute collision-free paths in the robot surroundings by optimizing a cost function. The cost function guides the vehicle to follow a vector field that encodes the desired path. We evaluated our method in challenging, cluttered static environments, such as warehouses and forests, and in the presence of moving obstacles, both in simulations and real-world experiments. In these experiments, we show that our algorithm performs collision checking and path planning faster than baseline methods. Since the method can have sequential or parallel implementations, we also compare the two versions of SSLAT and show that the run time for its parallel implementation provides significant speedups (more than 25 times faster) and makes the method independent of the number and shape of the obstacles found in the environment.

6.2 Recommendations for Future Work

The algorithm presented in Chapter 3, the FlowFMT*, was designed to solve the motion planning problem for holonomic vehicles navigating known and time-invariant flows. While useful in specific scenarios, some steps could be considered to improve the methodology. The most valuable improvement would be extending the method for dynamic environments where the flow changes and obstacles move over time by including another planning dimension that includes changing wind and obstacle predictions as shown in [151]. Another possibility would be accounting for vehicle motion and flow model uncertainties. While this comes naturally with other formulation types, such as Markov Decision Problems, with sampling-based methods, this is not straightforward. Regarding the algorithm's performance, improvement could be achieved through parallelization of the functions that

return the neighborhood sets or the functions that check the costs to each of the neighbors. A formal algorithm analysis would help validate the assumption that optimality holds after including neighborhood sets and adaptive neighborhood radius.

As the aerial platform concepts and scientific missions for the Venus exploration become more well-defined, we would need to change the algorithm to handle the specificities of the problem proposed in Chapter 4. In this dissertation, we formulated the problem as a simple "go from point A to point B" problem. But, as seen [58] for volcanic activity monitoring, there is value in formulating the problem in other forms, like "plan movements to maximize the chance of passing over point A recurrently". Another limitation is the coarse estimation of the winds around the planet. For an actual aerobot, many steps would need to be taken to plan and execute paths in the atmosphere of Venus. Local, real-time measurements of the wind could be fused to the global coarse wind forecasts through Gaussian Processes, similar to what was done for X's Loon [55] for stratospheric balloons on Earth, to improve the accuracy of the plans. Multiple vehicles in coordinated missions could be deployed to probe the wind in different regions of the world and improve the planning quality for all vehicles. In this dissertation, we did not try to create an accurate aircraft model. Instead, we used a simplified model to illustrate the functioning of the methodology. An accurate aircraft model is paramount in a real-world scenario, especially in creating local connections between samples.

Chapter 5 showed the efficacy of the SSLAT method for obstacle avoidance in 2D, however, we have identified some forms to enhance the method for wider applications. First, we noticed a few situations in our experiments where the robot became undecided in front of an obstacle. Since the paths are computed much faster than the robot can follow them, in symmetric situations, SSLAT computes consecutive paths with different homotopies (e.g., one path avoids the obstacle by the right and the other by the left). We usually solved that by slowing down the path computation to allow the robot to follow the computed path. A better strategy would be slightly modifying our cost-functional to prioritize paths homo-

topic to the previous one. Second, we observed that the paths created by SSLAT are not smooth, which can be demanding for the robot controller. To solve this, a post-processing step could be applied to the final path, or, more interestingly, a smooth lattice [127] could be developed and applied. Finally, our method finds paths in 2D. Although we used SSLAT to control drones, future work could include expanding it to 3D. Giving the sensor space an extra dimension will increase the number of triangles (or perhaps tetrahedrons, in this case) to be tested. The parallelism of the collision check should be as simple as in the 2D case, but hardware limitations may appear, and consequently, multidimensional CUDA grids and blocks could be beneficial.

Bibliography

[1] M. M. Lee, "Robots are taming the wild," 2022.

[2] Y. Yokokohji, "The use of robots to respond to nuclear accidents: Applying the lessons of the past to the fukushima daiichi nuclear power station," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 681–710, 2021.

[3] L. D. Barker, M. V. Jakuba, A. D. Bowen, C. R. German, T. Maksym, L. Mayer, A. Boetius, P. Dutrieux, and L. L. Whitcomb, "Scientific challenges and present capabilities in underwater robotic vehicle design and navigation for oceanographic exploration under-ice," *Remote Sensing*, vol. 12, no. 16, p. 2588, 2020.

[4] E. B. Clark, A. Branch, R. Castano, I. Fenty, C. Gebara, A. Kourchians, D. Limonadi, G. Madhok, P. McGarey, F. Mechentel, *et al.*, "Icenode: a buoyant vehicle for acquiring well-distributed, long-duration melt rate measurements under ice shelves," in *OCEANS 2021: San Diego–Porto*, pp. 1–10, IEEE, 2021.

[5] S. Lebonnois, F. Hourdin, V. Eymet, A. Crespin, R. Fournier, and F. Forget, "Superrotation of venus' atmosphere analyzed with a full general circulation model," *Journal of Geophysical Research: Planets*, vol. 115, no. E6, 2010.

[6] S. Lebonnois, N. Sugimoto, and G. Gilli, "Wave analysis in the atmosphere of venus below 100-km altitude, simulated by the lmd venus gcm," *Icarus*, vol. 278, pp. 38–51, 2016.

[7] A. Martinez, S. Lebonnois, E. Millour, T. Pierron, E. Moisan, G. Gilli, and F. Lefèvre, "Exploring the variability of the venusian thermosphere with the ipsl venus gcm," *Icarus*, vol. 389, p. 115272, 2023.

[8] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[9] E. Zermelo, "Über das navigationsproblem bei ruhender oder veränderlicher windverteilung," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 11, no. 2, pp. 114–124, 1931.

[10] K. P. Carroll, S. R. McClaran, E. L. Nelson, D. M. Barnett, D. K. Friesen, and G. N. William, "Auv path planning: an a* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones," in *Proceedings of the 1992 symposium on autonomous underwater vehicle technology*, pp. 79–84, IEEE, 1992.

[11] J. Rubio and S. Kragelund, "The trans-pacific crossing: long range adaptive path planning for UAVs through variable wind fields," in *Digital Avionics Systems Conference, 2003. DASC '03. The 22nd*, vol. 2, pp. 8.B.4–81–12 vol.2, Oct. 2003.

[12] A. Alvarez, A. Caiti, and R. Onken, "Evolutionary path planning for autonomous underwater vehicles in a variable ocean," *IEEE Journal of Oceanic Engineering*, vol. 29, no. 2, pp. 418–429, 2004.

[13] B. Garau, A. Alvarez, and G. Oliver, "Path Planning of Autonomous Underwater Vehicles in Current Fields with Complex Spatial Variability: an A* Approach," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 194–198, Apr. 2005. ISSN: 1050-4729.

[14] M. Soulignac, "Feasible and Optimal Path Planning in Strong Current Fields," *IEEE Transactions on Robotics*, vol. 27, pp. 89–98, Feb. 2011. Conference Name: IEEE Transactions on Robotics.

[15] T.-B. Koay and M. Chitre, "Energy-efficient path planning for fully propelled AUVs in congested coastal waters," in *2013 MTS/IEEE OCEANS - Bergen*, pp. 1–9, June 2013.

[16] D. Kularatne, S. Bhattacharya, and M. A. Hsieh, "Going with the flow: a graph based approach to optimal path planning in general flows," *Autonomous Robots*, vol. 42, pp. 1369–1387, Oct. 2018.

[17] C. S. Kulkarni and P. F. Lermusiaux, "Three-dimensional time-optimal path planning in the ocean," *Ocean Modelling*, vol. 152, p. 101644, Aug. 2020.

[18] T. McGee, S. Spry, and K. Hedrick, "Optimal path planning in a constant wind with a bounded turning rate," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Guidance, Navigation, and Control and Co-located Conferences, American Institute of Aeronautics and Astronautics, Aug. 2005.

[19] N. Ceccarelli, J. J. Enright, E. Frazzoli, S. J. Rasmussen, and C. J. Schumacher, "Micro UAV Path Planning for Reconnaissance in Wind," in *2007 American Control Conference*, pp. 5310–5315, July 2007. ISSN: 2378-5861.

[20] P. Oettershagen, F. Achermann, B. Müller, D. Schneider, and R. Siegwart, "Towards Fully Environment-Aware UAVs: Real-Time Path Planning with Online 3D Wind Field Prediction in Complex Terrain," *arXiv:1712.03608 [cs]*, Dec. 2017. arXiv: 1712.03608.

[21] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *arXiv:1105.1186 [cs]*, May 2011. arXiv: 1105.1186.

[22] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a dubins airplane," in *2007 46th IEEE conference on decision and control*, pp. 2379–2384, IEEE, 2007.

[23] J. Ware and N. Roy, "An analysis of wind field estimation and exploitation for quadrotor flight in the urban canopy layer," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1507–1514, May 2016.

[24] J. W. Langelaan, "Tree-based trajectory planning to exploit atmospheric energy," in *2008 American Control Conference*, pp. 2328–2333, June 2008. ISSN: 2378-5861.

[25] A. Chakrabarty and J. Langelaan, "UAV flight path planning in time varying complex wind-fields," in *2013 American Control Conference*, pp. 2568–2574, June 2013. ISSN: 2378-5861.

[26] B. Moon, S. Sachdev, J. Yuan, and S. Scherer, "Time-optimal path planning in a constant wind for uncrewed aerial vehicles using dubins set classification," 2023.

[27] C. L. Hull, "The goal-gradient hypothesis and maze learning.," *Psychological review*, vol. 39, no. 1, p. 25, 1932.

[28] S. A. Masoud and A. A. Masoud, "Constrained motion control using vector potential fields," *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, vol. 30, no. 3, pp. 251–272, 2000.

[29] L. A. Loeff, "Algorithm for computer guidance of a manipulator in between obstacles," Master's thesis, Oklahoma State University, 1973.

[30] L. A. Loeff and A. Soni, "An algorithm for computer guidance of a manipulator in between obstacles," *Journal of Engineering for Industry*, vol. 97, no. 3, pp. 836–842, 1975.

[31] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, IEEE, 1985.

[32] L. E. Kavraki and S. M. LaValle, "Motion planning," in *Springer Handbook of Robotics*, pp. 139–162, Springer, 2016.

[33] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, 1992.

[34] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[35] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, vol. 22, no. 2, 1992.

[36] C. I. Connolly, J. B. Burns, and R. Weiss, "Path planning using laplace's equation," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 2102–2106, IEEE, 1990.

[37] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 3553–3559, IEEE, 2005.

[38] L. C. Pimenta, G. A. Pereira, and R. C. Mesquita, "Fully continuous vector fields for mobile robot navigation on sequences of discrete triangular regions," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1992–1997, IEEE, 2007.

[39] L. Yang and S. M. Lavalle, "The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 419–432, 2004.

[40] D. Lawrence, E. Frew, and W. Pisano, "Lyapunov vector fields for autonomous uav flight control," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, p. 6317, 2007.

[41] E. W. Frew, D. A. Lawrence, C. Dixon, J. Elston, and W. J. Pisano, "Lyapunov guidance vector fields for unmanned aircraft applications," in *2007 American control conference*, pp. 371–376, IEEE, 2007.

[42] L. C. Pimenta, G. A. Pereira, M. M. Gonçalves, N. Michael, M. Turpin, and V. Kumar, "Decentralized controllers for perimeter surveillance with teams of aerial robots," *Advanced Robotics*, vol. 27, no. 9, pp. 697–709, 2013.

[43] D. A. De Lima and G. A. S. Pereira, "Navigation of an autonomous car using vector fields and the dynamic window approach," *Journal of Control, Automation and Electrical Systems*, vol. 24, pp. 106–116, 2013.

[44] G. A. S. Pereira and E. J. Freitas, "Navigation of semi-autonomous service robots using local information and anytime motion planners," *Robotica*, vol. 38, pp. 2080–2098, 2020.

[45] W. Yao, "Guiding vector fields for multi-robot coordinated navigation," in *Guiding Vector Fields for Robot Motion Control*, pp. 191–232, Springer, 2023.

[46] J.-W. Kwon, J. H. Kim, and J. Seo, "Vector field guided auto-landing control of airship with wind disturbance," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1114–1119, 2014.

[47] V. M. Gonçalves, R. McLaughlin, and G. A. Pereira, "Precise landing of autonomous aerial vehicles using vector fields," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4337–4344, 2020.

[48] A. Guo, Z. Zhou, R. Wang, X. Zhao, and X. Zhu, "Vector field path following of a full-wing solar-powered unmanned aerial vehicle (UAV) landing based on dubins path: a lesson from multiple landing failures," *The Aeronautical Journal*, vol. 125, no. 1290, pp. 1283–1312, 2021.

[49] B. Martinez Rocamora Jr, R. R. Lima, K. Samarakoon, J. Rathjen, J. N. Gross, and G. A. Pereira, "Oxpecker: A tethered uav for inspection of stone-mine pillars," *Drones*, vol. 7, no. 2, p. 73, 2023.

[50] R. R. Lima, B. M. Rocamora, and G. A. Pereira, "Continuous vector fields for precise cable-guided landing of tethered uavs," *IEEE Robotics and Automation Letters*, 2023.

[51] G. A. Pereira, S. Choudhury, and S. Scherer, "A framework for optimal repairing of vector field-based motion plans," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 261–266, IEEE, 2016.

[52] B. M. R. Jr. and G. A. S. Pereira, "Fast path computation using lattices in the sensor-space for forest navigation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1117–1123, 2021.

[53] B. Martinez Rocamora Jr and G. A. Pereira, "Parallel sensor-space lattice planner for real-time obstacle avoidance," *Sensors*, vol. 22, no. 13, p. 4770, 2022.

[54] F. Rossi, A. Branch, M. P. Schodlok, T. Stanton, I. G. Fenty, J. V. Hook, and E. B. Clark, "Stochastic Guidance of Buoyancy Controlled Vehicles under Ice Shelves using Ocean Currents," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8657–8664, Sept. 2021. ISSN: 2153-0866.

[55] L. Nagpal and K. Samdani, "Project Loon: Innovating the connectivity worldwide," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pp. 1778–1784, May 2017.

[56] VEXAG, "Roadmap for Venus Exploration," tech. rep., Venus Exploration Analysis Group, 2019.

[57] J. S. Greaves, A. M. S. Richards, W. Bains, P. B. Rimmer, H. Sagawa, D. L. Clements, S. Seager, J. J. Petkowski, C. Sousa-Silva, S. Ranjan, E. Drabek-Maunder, H. J. Fraser, A. Cartwright, I. Mueller-Wodarg, Z. Zhan, P. Friberg, I. Coulson, E. Lee, and J. Hoge, "Phosphine gas in the cloud decks of Venus," *Nature Astronomy*, vol. 5, pp. 655–664, Sept. 2020.

[58] F. Rossi, M. Saboia, S. Krishnamoorthy, and J. V. Hook, "Proximal Exploration of Venus Volcanism with Teams of Autonomous Buoyancy-Controlled Balloons," Mar. 2023. arXiv:2303.02104 [cs].

[59] B. Martinez Rocamora Jr., A. P. I. Juan, and G. A. Pereira, "Towards finding energy efficient paths for hybrid airships in the atmosphere of venus," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 386–393, IEEE, 2022.

[60] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast Marching Tree: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions," Feb. 2015. arXiv:1306.3532 [cs].

[61] J. Lee, C. Yoo, R. Hall, S. Anstee, and R. Fitch, "Energy-optimal kinodynamic planning for underwater gliders in flow fields," in *Australasian Conference on Robotics and Automation, ACRA*, 2017.

[62] L. Bonin, A. Guitart, D. Delahaye, X. Prats, and E. Feron, "Computing optimal trajectories for light soaring aircraft using Fast Marching Tree Star," Jan. 2023.

[63] D. E. Kirk, *Optimal control theory: an introduction*. Courier Corporation, 2004.

[64] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, "Path planning for autonomous underwater vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 331–341, 2007.

[65] L. Blackmore, Y. Kuwata, M. T. Wolf, C. Assad, N. Fathpour, C. Newman, and A. Elfes, "Global reachability and path planning for planetary exploration with montgolfiere balloons," in *2010 IEEE International Conference on Robotics and Automation*, pp. 3581–3588, IEEE, 2010.

[66] Y. Nie, O. Faqir, and E. C. Kerrigan, "ICLOCS2: Try this Optimal Control Problem Solver Before you Try the Rest," in *2018 UKACC 12th International Conference on Control (CONTROL)*, pp. 336–336, Sept. 2018.

[67] A. T. Basilevsky and J. W. Head, "The surface of Venus," *Reports on Progress in Physics*, vol. 66, pp. 1699–1734, Sept. 2003. Publisher: IOP Publishing.

[68] G. Landis, A. Colozza, and C. LaMarre, "Atmospheric flight on Venus," in *40th AIAA Aerospace Sciences Meeting & Exhibit*, (Reno,NV,U.S.A.), American Institute of Aeronautics and Astronautics, Jan. 2002.

[69] A. Sánchez-Lavega, S. Lebonnois, T. Imamura, P. Read, and D. Luz, "The Atmospheric Dynamics of Venus," *Space Science Reviews*, vol. 212, pp. 1541–1616, Nov. 2017.

[70] G. Dorrington, "Venus Cloud Life in situ Sampling Platform Options," in *43rd COSPAR Scientific Assembly*, vol. 43, p. 434, Jan. 2021.

[71] S. S. Limaye, R. Mogul, D. J. Smith, A. H. Ansari, G. P. Słowik, and P. Vaishampayan, "Venus' Spectral Signatures and the Potential for Life in the Clouds," *Astrobiology*, vol. 18, pp. 1181–1198, Sept. 2018.

[72] S. Smrekar, S. Hensley, D. Dyar, and J. Helbert, "VERITAS (Venus Emissivity, Radio Science, InSAR, Topography And Spectroscopy): A Proposed Discovery Mission," in *EPSC-DPS Joint Meeting 2019*, vol. 13, (Geneva, Switzerland), p. 1124, 2019.

[73] V. Avduevsky, M. Y. Marov, and M. Rozhdestvensky, "Preliminary results of measurements by space probes venera 5 and venera 6 in the atmosphere of venus—summary," *Radio Science*, vol. 5, no. 2, pp. 333–337, 1970.

[74] R. Sagdeev, V. Linkin, J. Blamont, and R. Preston, "The vega venus balloon experiment," *Science*, vol. 231, no. 4744, pp. 1407–1408, 1986.

[75] B. C. Murray, M. J. Belton, G. E. Danielson, M. E. Davies, D. Gault, B. Hapke, B. O'Leary, R. G. Strom, V. Suomi, and N. Trask, "Venus: Atmospheric motion and structure from mariner 10 pictures," *Science*, vol. 183, no. 4131, pp. 1307–1315, 1974.

[76] L. Colin, "The pioneer venus program," *Journal of Geophysical Research: Space Physics*, vol. 85, no. A13, pp. 7575–7598, 1980.

[77] H. Masursky, E. Eliason, P. G. Ford, G. E. McGill, G. H. Pettengill, G. G. Schaber, and G. Schubert, "Pioneer venus radar results: Geology from images and altimetry," *Journal of Geophysical Research: Space Physics*, vol. 85, no. A13, pp. 8232–8260, 1980.

[78] R. Saunders, A. Spear, P. Allin, R. Austin, A. Berman, R. Chandlee, J. Clark, A. Decharon, E. De Jong, D. Griffith, *et al.*, "Magellan mission summary," *Journal of Geophysical Research: Planets*, vol. 97, no. E8, pp. 13067–13090, 1992.

[79] H. Svedhem, D. Titov, F. Taylor, and O. Witasse, "Venus express mission," *Journal of Geophysical Research: Planets*, vol. 114, no. E5, 2009.

[80] L. S. Glaze, J. B. Garvin, B. Robertson, N. M. Johnson, M. J. Amato, J. Thompson, C. Goodloe, and D. Everett, "Davinci: Deep atmosphere venus investigation of noble gases, chemistry, and imaging," in *2017 IEEE Aerospace Conference*, pp. 1–5, IEEE, 2017.

[81] S. Smrekar, S. Hensley, R. Nybakken, M. S. Wallace, D. Perkovic-Martin, T.-H. You, D. Nunes, J. Brophy, T. Ely, E. Burt, *et al.*, "Veritas (venus emissivity, radio science, insar, topography, and spectroscopy): a discovery mission," in *2022 IEEE Aerospace Conference (AERO)*, pp. 1–20, IEEE, 2022.

[82] James A. Cutts, Larry H. Matthies, and Thomas W. Thompson, "Aerial Platforms for the Scientific Exploration of Venus," tech. rep., NASA Jet Propulsion Laboratory, 2018.

[83] G. A. Landis, "Exploring Venus by solar airplane," in *AIP Conference Proceedings*, vol. 552, (Albuquerque, New Mexico), pp. 16–18, AIP, 2001. ISSN: 0094243X.

[84] E. Z. Noe Dobrea, J. Freeman, A. R. Gibson, D. Hall, L. Lemke, B. Pham, and B. T. Schiltgen, "Exploring aircraft and mission profile designs for long-duration flight in the Venusian atmosphere," in *AIAA Scitech 2020 Forum*, (Orlando, FL), American Institute of Aeronautics and Astronautics, Jan. 2020.

[85] Z. Rubin, A. Aboelezz, B. Herkenhoff, and M. Hassanalian, "Drones for Venus Exploration: Energy Harvesting Mechanisms and Thermal-based Flight Control," in *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics, 2021.

[86] J. Cutts, K. Baines, L. Dorsky, W. Frazier, J. Izraelevitz, S. Krishnamoorthy, M. Pauken, M. S. Wallace, P. Byrne, S. Seager, C. Wilson, and J. O'Rourke, "Exploring the Clouds of Venus: Science Driven Aerobot Missions to our Sister Planet," in *2022 IEEE Aerospace Conference (AERO)*, pp. 1–20, Mar. 2022. ISSN: 1095-323X.

[87] A. Arredondo, A. Hodges, J. N. H. Abrahams, C. C. Bedford, B. D. Boatwright, J. Buz, C. Cantrall, J. Clark, A. Erwin, S. Krishnamoorthy, L. Magaña, R. M. McCabe, E. C. McIntosh, J. L. Noviello, M. Pellegrino, C. Ray, M. J. Styczinski, and P. Weigel, "VALENTInE: A Concept for a New Frontiers–Class Long-duration In Situ Balloon-based Aerobot Mission to Venus," *The Planetary Science Journal*, vol. 3, p. 152, July 2022. Publisher: IOP Publishing.

[88] R. Polidan, G. Lee, D. Sokol, K. Griffin, and L. Bolisay, "Venus Atmospheric Maneuverable Platform (VAMP)," in *Workshop on Venus Exploration Targets*, vol. 1781, p. 6011, May 2014. Conference Name: ADS Bibcode: 2014LPICo1781.6011P.

[89] G. A. Landis and E. Haag, "Analysis of Solar Cell Efficiency for Venus Atmosphere and Surface Missions," in *11th International Energy Conversion Engineering Conference*, (San Jose, CA), American Institute of Aeronautics and Astronautics, July 2013.

[90] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, "Autonomous navigation of stratospheric balloons using reinforcement learning," *Nature*, vol. 588, no. 7836, pp. 77–82, 2020.

[91] J. L. Hall, J. Cameron, M. Pauken, J. Izraelevitz, M. W. Dominguez, and K. T. Wehage, "Altitude-Controlled Light Gas Balloons for Venus and Titan Exploration," in *AIAA Aviation 2019 Forum*, (Dallas, Texas), American Institute of Aeronautics and Astronautics, June 2019.

[92] M. T. Wolf, L. Blackmore, Y. Kuwata, N. Fathpour, A. Elfes, and C. Newman, "Probabilistic motion planning of balloons in strong, uncertain wind fields," in *2010 IEEE International Conference on Robotics and Automation*, pp. 1123–1129, May 2010. ISSN: 1050-4729.

[93] M. Marov, "Results of Venus Missions," *Annual Review of astronomy and astrophysics*, vol. 16, no. 1, pp. 141–169, 1978.

[94] J. M. Buchanan, "Opportunity Cost," in *The World of Economics* (J. Eatwell, M. Milgate, and P. Newman, eds.), The New Palgrave, pp. 520–525, London: Palgrave Macmillan UK, 1991.

[95] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT*," in *2011 IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 1478–1483, IEEE, May 2011.

[96] M. Owen, R. W. Beard, and T. W. McLain, "Implementing Dubins Airplane Paths on Fixed-Wing UAVs*," in *Handbook of Unmanned Aerial Vehicles* (K. P. Valavanis and G. J. Vachtsevanos, eds.), pp. 1677–1701, Dordrecht: Springer Netherlands, 2015.

[97] S. Swenson, B. Argrow, E. Frew, S. Borenstein, and J. Keeler, "Development and deployment of air-launched drifters from small uas," *Sensors*, vol. 19, no. 9, p. 2149, 2019.

[98] T. Lolla, M. P. Ueckermann, K. Yiğit, P. J. Haley, and P. F. J. Lermusiaux, "Path planning in time dependent flow fields using level set methods," in *2012 IEEE International Conference on Robotics and Automation*, pp. 166–173, May 2012. ISSN: 1050-4729.

[99] T. Lolla, P. J. Haley Jr., and P. F. J. Lermusiaux, "Path planning in multi-scale ocean flows: Coordination and dynamic obstacles," *Ocean Modelling*, vol. 94, pp. 46–66, Oct. 2015.

[100] D. N. Subramani, "Energy-optimal path planning by stochastic dynamically orthogonal level-set optimization," *Ocean Modelling*, p. 21, 2016.

[101] D. N. Subramani, Q. J. Wei, and P. F. Lermusiaux, "Stochastic time-optimal path-planning in uncertain, strong, and dynamic flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 333, pp. 218–237, May 2018.

[102] C. Lemardelé, M. Estrada, L. Pagès, and M. Bachofner, "Potentialities of drones and ground autonomous delivery devices for last-mile logistics," *Transportation Research Part E: Logistics and Transportation Review*, vol. 149, p. 102325, 2021.

[103] P. Baniasadi, M. Foumani, K. Smith-Miles, and V. Ejov, "A transformation technique for the clustered generalized traveling salesman problem with applications to logistics," *European Journal of Operational Research*, vol. 285, no. 2, pp. 444–457, 2020.

[104] L. F. Oliveira, A. P. Moreira, and M. F. Silva, "Advances in forest robotics: A state-of-the-art survey," *Robotics*, vol. 10, no. 2, p. 53, 2021.

[105] B. B. Kocer, B. Ho, X. Zhu, P. Zheng, A. Farinha, F. Xiao, B. Stephens, F. Wiesemüller, L. Orr, and M. Kovac, "Forest drones for environmental sensing and nature conservation," in *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*, pp. 1–8, IEEE, 2021.

[106] B. Ichter, E. Schmerling, and M. Pavone, "Group marching tree: Sampling-based approximately optimal motion planning on gpus," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, pp. 219–226, IEEE, 2017.

[107] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT*," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3513–3518, 2011.

[108] N. M. Amato and L. K. Dale, "Probabilistic roadmap methods are embarrassingly parallel," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 1, pp. 688–694, IEEE, 1999.

[109] A. M. C. Rezende, V. M. Goncalves, and L. C. A. Pimenta, "Constructive time-varying vector fields for robot navigation," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 852–867, 2022.

[110] W. Yao, H. G. de Marina, B. Lin, and M. Cao, "Singularity-free guiding vector field for robot navigation," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1206–1221, 2021.

[111] C. Wu, J. Chen, D. Jeltsema, and C. Dai, "Guidance vector field encoding based on contraction analysis," in *2018 European Control Conference (ECC)*, pp. 282–287, IEEE, 2018.

[112] E. W. Frew and D. Lawrence, "Tracking dynamic star curves using guidance vector fields," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 6, pp. 1488–1495, 2017.

[113] A. C. Chiella, H. N. Machado, B. O. Teixeira, and G. A. S. Pereira, "GNSS/LiDAR-Based navigation of an aerial robot in sparse forests," *Sensors*, vol. 19, no. 19, p. 4061, 2019.

[114] V. M. Gonçalves, L. C. Pimenta, C. A. Maia, B. C. Dutra, and G. A. S. Pereira, "Vector fields for robot navigation along time-varying curves in n-dimensions," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 647–659, 2010.

[115] G. S. Clem, *An Optimized Circulating Vector Field Obstacle Avoidance Guidance for Unmanned Aerial Vehicles*. PhD thesis, Ohio University, 2018.

[116] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 802–807, IEEE, 1993.

[117] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[118] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE transactions on robotics and automation*, vol. 7, no. 3, pp. 278–288, 1991.

[119] W. Chen, N. Wang, X. Liu, and C. Yang, "Vfh* based local path planning for mobile robot," in *2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*, pp. 18–23, IEEE, 2019.

[120] A. Babinec, F. Duchoň, M. Dekan, Z. Mikulová, and L. Jurišica, "Vector field histogram* with look-ahead tree extension dependent on time variable environment," *Transactions of the Institute of Measurement and Control*, vol. 40, no. 4, pp. 1250–1264, 2018.

[121] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[122] T. Hossain, H. Habibullah, R. Islam, and R. V. Padilla, "Local path planning for autonomous mobile robots by integrating modified dynamic-window approach and improved follow the gap method," *Journal of Field Robotics*, vol. 39, no. 4, pp. 371–386, 2022.

[123] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris, "Robot motion planning on a chip.," in *Robotics: Science and Systems*, 2016.

[124] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-CAP: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 8456–8463, 2018.

[125] M. Ostilli, "Cayley trees and bethe lattices: A concise analysis for mathematicians and physicists," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 12, pp. 3417–3423, 2012.

[126] A. Lacaze, Y. Moscovitz, N. DeClaris, and K. Murphy, "Path planning for autonomous vehicles driving over rough terrain," in *IEEE International Symposium on Intelligent Control*, pp. 50–55, 1998.

[127] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[128] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "Faster: Fast and safe trajectory planner for flights in unknown environments," *arXiv preprint arXiv:2001.04420*, 2020.

[129] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[130] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear, *The art of multiprocessor programming*. Newnes, 2020.

[131] S. Carpin and E. Pagello, "On parallel rrts for multi-robot systems," in *Proc. 8th Conf. Italian Association for Artificial Intelligence*, pp. 834–841, 2002.

[132] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.

[133] A. Hidalgo-Paniagua, J. P. Bandera, M. Ruiz-de Quintanilla, and A. Bandera, "Quad-rrt: A real-time gpu-based global path planner in large-scale real environments," *Expert Systems with Applications*, vol. 99, pp. 141–154, 2018.

[134] R. C. Lawson, L. Wills, and P. Tsiotras, "Gpu parallelization of policy iteration rrt," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4369–4374, IEEE, 2020.

[135] A. Bleiweiss, "Gpu accelerated pathfinding," in *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pp. 65–74, 2008.

[136] J. Pan, C. Lauterbach, and D. Manocha, "g-planner: Real-time motion planning and global navigation using gpus.," in *AAAI*, pp. 2243–2248, 2010.

[137] J. Pan, C. Lauterbach, and D. Manocha, "Efficient nearest-neighbor computation for gpu-based motion planning," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2243–2248, IEEE, 2010.

[138] J. Pan and D. Manocha, "Gpu-based parallel collision detection for fast motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.

[139] V. M. Gonçalves, C. A. Maia, G. A. S. Pereira, and L. C. A. Pimenta, "Navegação de robôs utilizando curvas implícitas," *Sba: Controle & Automação Sociedade Brasileira de Automatica*, vol. 21, no. 1, pp. 43–57, 2010.

[140] I. Ko, B. Kim, and F. C. Park, "Randomized path planning on vector fields," *The International Journal of Robotics Research*, vol. 33, no. 13, pp. 1664–1682, 2014.

[141] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020.

[142] N. Corporation, "Cuda c programming guide," 2017.

[143] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *IEEE International Conference on Robotics and Automation*, pp. 2899–2906, 2012.

[144] D. Perille, A. Truong, X. Xiao, and P. Stone, "Benchmarking metric ground navigation," in *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, 2020.

[145] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[146] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system," 2018.

[147] B. J. Mohler, W. B. Thompson, S. H. Creem-Regehr, H. L. Pick, and W. H. Warren, "Visual flow influences gait transition speed and preferred walking speed," *Experimental brain research*, vol. 181, no. 2, pp. 221–228, 2007.

[148] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," *ACM Sigplan Notices*, vol. 47, no. 8, pp. 117–128, 2012.

[149] G. Grisettiyz, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 2432–2437, 2005.

[150] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. https://ompl.kavrakilab.org.

[151] M. Otte, W. Silva, and E. Frew, "Any-time path-planning: Time-varying wind field + moving obstacles," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2575–2582, May 2016.

[152] J. Xu, K. Song, D. Zhang, H. Dong, Y. Yan, and Q. Meng, "Informed anytime fast marching tree for asymptotically optimal motion planning," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 6, pp. 5068–5077, 2021.

[153] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

Appendix A

Extending the FlowFMT*: How to Converge to Optimality Faster?

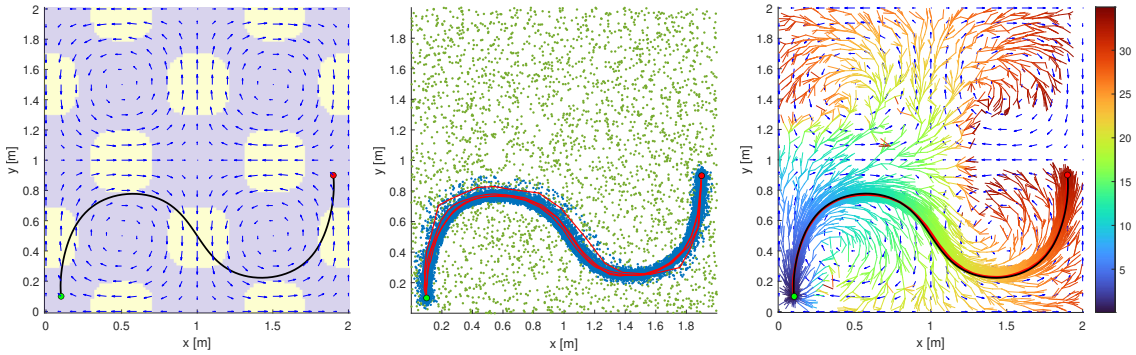
In Chapter 3, we might encounter a case where we want to converge to an optimal solution faster and in a limited time. However, rerunning the method with an increased number of particles is not a satisfactory solution, as the running time scales up rapidly with the number of samples (the FMT* algorithm is $\mathcal{O}(n \log n)$). Therefore, to achieve faster convergence, one option is to run the algorithm with a smaller number of samples just to find a rough approximation of the optimal path, with the correct homotopy. Then, we increase the density of samples locally, around the path that was found. Therefore, the FlowFMT* algorithm could be extended to be "informed" and "anytime", similar to the work proposed by [152]. By "informed", we mean that given a certain amount of maximum time t_{max}, it will look for a solution, if it has not found one yet, or, in the alternative case, a better solution. Between two subsequent searches, the number of samples gets doubled. By "informed", we mean that if the method has found a solution path, it will prioritize looking for a better path near the previously found solution. While the main preoccupation in [152] is to improve computational performance, our reasons are not only related to performance. But also the fact that we consider that the flow has constant speed along the edges of the graph. However, this assumption is not necessarily valid when the method uses low-density random

sampling and the cost of the path is underestimated. This problem is solved by increasing the sampling density. Therefore, to guarantee that the solution can change its topology, we provide a way to continue to look for better paths globally as well. We increase the density both inside and outside of the region defined by the best solution already found. A heuristic, α, can be included to control the balance between exploration and exploitation of the method. A possible solution is shown in Alg. 9 (lines 8 and 9).

Algorithm 9 Informed Anytime Flow-Aware FMT* Algorithm

1: **function** $\sigma =$ INFORMED ANYTIME FLOWFMT*(x_s, x_g, n, α)
2: $\quad V_{pool} \leftarrow RandomSample(n)$
3: $\quad$ **while** $t \leq t_{max}$ **do**
4: $\quad\quad \sigma \leftarrow FAFMT*(x_s, x_g, V_{pool})$
5: $\quad\quad$ **if** $\sigma = \varnothing$ **then**
6: $\quad\quad\quad V_{pool} \leftarrow V_{pool} \cup RandomSample(n)$
7: $\quad\quad$ **else**
8: $\quad\quad\quad V_{pool} \leftarrow V_{pool} \cup RandomSample(n\alpha)$ ▷ Exploration
9: $\quad\quad\quad V_{pool} \leftarrow V_{pool} \cup SamplePath(n(1-\alpha))$ ▷ Exploitation
10: $\quad\quad$ **end if**
11: $\quad\quad n \leftarrow 2n$
12: $\quad$ **end while**
13: $\quad$ **if** $\sigma = \varnothing$ **then**
14: $\quad\quad$ **return** FAILURE
15: $\quad$ **end if**
16: $\quad$ **return** σ
17: **end function**

In the 2D environment described in Chapter 3.3.3, the Flow-Aware FMT* (FlowFMT*) is used to find paths from the start position $\vec{x}_{start} = [0.1, 0.1]^\mathsf{T}$ to a goal position $x_{goal} = [1.9, 0.9]^\mathsf{T}$, as shown in Figure A.1 given $60\,\mathrm{s}$. This figure shows the optimal-time paths for minimum-time cost functions and the resultant trees used to obtain these paths. The edges of the tree were colored by the cost of the vertices. We observed a cost of $C_t(\sigma) = 32.86\,\mathrm{s}$ using OCS (set up with ICLOCS2), and $C_t(\sigma) = 32.52\,\mathrm{s}$ using FlowFMT*. At the last iteration, our method ran with only 12,800 samples.



(a) Environment (b) Sampling Strategy (c) Search Tree

Figure A.1: Results from the anytime FlowFMT*. In (a), we show the environment and the OCS solution; in (b), we show the samples generated all over the environment (green dots) and the samples generated around the previously found paths (red lines) in blue; in (c), we show the search tree and comparison of FlowFMT* and OCS solutions.

Appendix B

How to Find Time- and Energy-Optimal Paths for Nonholonomic Vehicles?

Taylor-Series Approximation of Trochoid Curves

Another restriction of the method presented in Chapter 3 is that we considered a holonomic model for the vehicle. While this can be a reasonable assumption for slow vehicles that traverse long distances, it is a limitation for more complex vehicle models. A possible extension of the method would consider a nonholonomic model, which can only change its heading over time by controlling its rotational speed. An example of such a model considers that the orientation of the vehicle is fully defined by its heading and decouples

the vertical component of the equation:

$$\begin{aligned} x_g(t) &= x_g(0) + \int_0^t (v_r \cos(\phi)\cos(\psi) + c_x)\, dt \\ y_g(t) &= y_g(0) + \int_0^t (v_r \cos(\phi)\sin(\psi) + c_y)\, dt \\ z_g(t) &= z_g(0) + \int_0^t (v_r \sin(\phi) + c_z)\, dt \\ \psi_g(t) &= \psi_g(0) + \int_0^t \dot{\psi}\, dt \\ \phi_g(t) &= \phi_g(0) + \int_0^t \dot{\phi}\, dt\,. \end{aligned} \tag{B.1}$$

For a two-dimensional environment ($\phi = 0$ and $\dot{\phi} = 0$), and considering the time interval $\Delta t = t - t_0$, the model can be reduced to

$$\begin{cases} x_g(t) = x_0 + \frac{v_r}{\dot{\psi}}(\sin(\psi(t)) - \sin\psi_0) + c_x \Delta t \\ y_g(t) = y_0 + \frac{v_r}{\dot{\psi}}(\cos(\psi(t)) - \cos\psi_0) + c_y \Delta t \\ \psi_g(t) = \psi_0 + \dot{\psi}\Delta t \end{cases} \tag{B.2}$$

This is known as a *trochoid* curve. The ratio between v_r and c defines if the trochoid is curtate ($v_r < c$), common ($v_r = c$), or prolate ($v_r > c$). The problem we need to solve is the inverse trochoid problem, i.e., given the initial position $\vec{x}(t_0) = [x(t_0), y(t_0)]^T = [x_0, y_0]^T$ and heading $\psi(t_0) = \psi_0$, goal position $\vec{x}_g(t) = [x_g(t), y_g(t)]^T$, the local wind vector $\vec{c}$ and the vehicle airspeed v_r, we wish to find t and $\dot{\psi}$, satisfying the nonlinear system of equations. The nonlinear system of equations can be solved numerically or through sampling and assessment of candidate trochoidal paths. However, these alternatives can be computationally expensive. An idea is trying to find a closed solution to a simplified problem using the Taylor expansions of the geometric functions at t_0:

$$\sin(\psi(t)) = \sin(\psi_0 + \dot{\psi}\Delta t) = \sin\psi_0 + \sum_{k=1}^{\infty} \frac{(-1)^k(\psi_0 + \dot{\psi}\Delta t)^{1+2k}}{(1+2k)!} \tag{B.3}$$

and

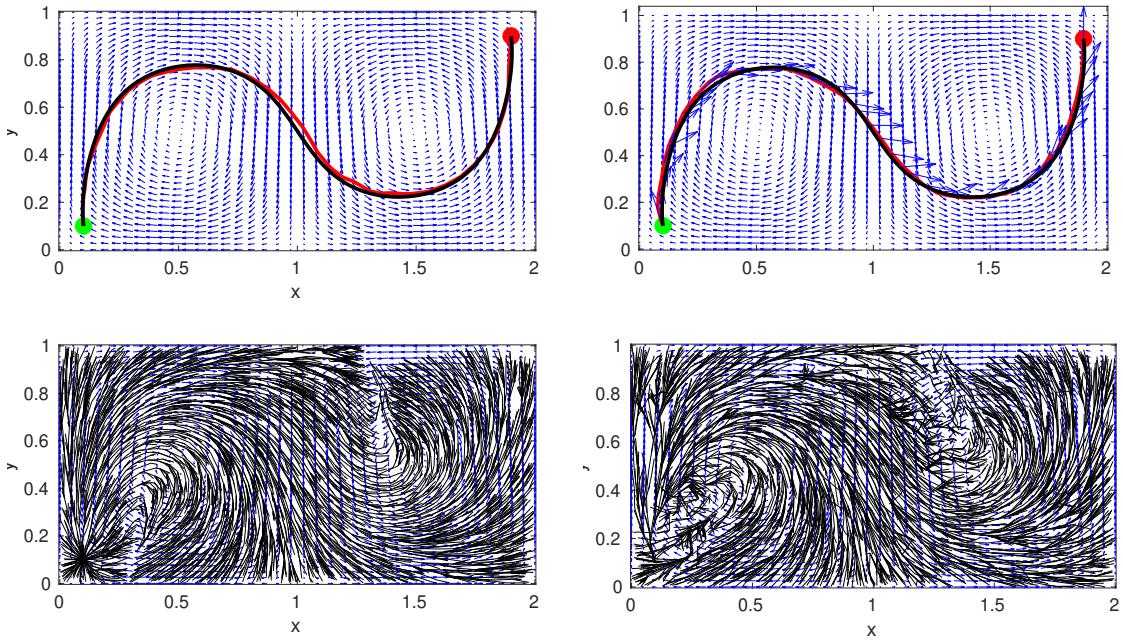
$$\cos\left(\psi(t)\right) = \cos\left(\psi_0 + \dot{\psi}\Delta t\right) = \cos\psi_0 + \sum_{k=1}^{\infty} \frac{(-1)^k(\psi_0 + \dot{\psi}\Delta t)^{2k}}{(2k)!}\,, \tag{B.4}$$

and rewriting Eq. B.2 as:

$$\begin{cases} x_g(t) = x_0 + v_r\Big(\Delta t\cos\psi_0 - \frac{1}{2}\dot{\psi}\Delta t^2\sin\psi_0 + \mathcal{O}(\Delta t^3)\Big) + c_x\Delta t \\ y_g(t) = y_0 + v_r\Big(-\Delta t\sin\psi_0 - \frac{1}{2}\dot{\psi}\Delta t^2\cos\psi_0 + \mathcal{O}(\Delta t^3)\Big) + c_y\Delta t \\ \psi_g(t) = \psi_0 + \dot{\psi}\Delta t \end{cases} \tag{B.5}$$

Then, we solve the system for the two variables Δt and $\dot{\psi}$. The reachable region for such a vehicle is given by an expansion in time of the maximum and minimum rotational speed (assuming it can change rotational speed instantaneously).

FlowFMT* was also tested considering nonholonomic constraints. For that, we assume that the vehicle's configuration includes its heading, ψ. We also assume that the vehicle can move with a constant and bounded yaw rate, $\|\dot{\psi}\| \leq \dot{\psi}_{max}$, between two vertices of the graph. Figure B.1 compares the optimal time paths with increasing constraints in the yaw rate, from unbounded to $0.25\,\mathrm{rad\,s^{-1}}$. Again, the start position is $\vec{x}_{start} = (0.1, 0.1)$ and goal position, $\vec{x}_{goal} = (1.9, 0.9)$. The initial heading is $\psi_0 = \pi/2$. It is possible to see how the tree is affected by observing the region around its root. With an unbounded yaw rate, the vehicle is able to move to the surrounding area, with the heading constraint, the tree gets narrower. The trajectories for the equivalent optimal control obtained using ICLOCS are also included for reference.



(a) Holonomic, $\|\dot{\psi}\| \leq \infty$ (b) $\|\dot{\psi}\| \leq 1\,\mathrm{rad\,s^{-1}}$

Figure B.1: The FlowFMT* was tested using the 2D double gyre flow and nonholonomic constraints. This experiment used 6400 samples, and again, the method replicated the optimal control solution.

Appendix C

Gradient-Descent Method to Find Wind-Deformed Dubins Paths Between Two Poses

Chapter 4 showed the EWRRT algorithm, which calculated connections between vertices using Dubins paths. We started by connecting start and goal positions in a wind-free environment. Later, it uses the $DeformTrajectory$ function to deform the path considering the wind field. The endpoints of these two paths become different after accounting for the drift caused by the wind; the vehicle does not reach the desired goal. The idea, inspired by [96], a method for calculating Dubins Airplane paths under the effect of wind is shown. Then, they use the difference between these two endpoints to create a new virtual goal (by subtracting this distance from the previous wind-free goal) for the wind-free step, hoping that once it is deformed, it will reach the goal. The method iterates until the solution converges. However, the method cannot solve the problem when strong currents ($|w| > |v_a|$) are present. Here, we present a new method to address and solve this issue.

The idea is to connect the start and goal in a wind-free environment, deform the path with the wind, and calculate the distance between the endpoint and the desired goal. But now we also calculate the distance gradient to the desired goal with respect to the wind-

Algorithm 10 Get Gradient

function $\nabla d = \text{GETGRADIENT}(v_{start}, v_{goal})$
  for j = 1, 2, 3 **do**
    $v_{new,j}^{virtual,+} \leftarrow v_{new,j}^{virtual} + \delta x_j$
    $\sigma \leftarrow DubinsPath(v_{near}, v_{new})$
    $S_{wind}^{+} \leftarrow DeformTrajectory(\sigma, N, v_a)$
    $d^{+} = |S_{wind}^{+}(end) - v_{near}|$
    $v_{new,j}^{virtual,-} \leftarrow v_{new,j}^{virtual} - \delta x_j$
    $\sigma \leftarrow DubinsPath(v_{near}, v_{new})$
    $S_{wind}^{-} \leftarrow DeformTrajectory(\sigma, N, v_a)$
    $d^{-} = |S_{wind}^{-}(end) - v_{near}|$
    $\partial d/\partial x_j \leftarrow (d^{+} - d^{-})/(2\delta x_j)$
  end for
  $\nabla d \leftarrow [\partial d/\partial x, \partial d/\partial y, \partial d/\partial z]$
end function

free goal using the $GetGradient$ function. We assume that if the distance function is well-behaved, we can minimize the distance between the desired goal and the endpoint of the trajectory that considers the wind by following the gradient. Following the gradient scaled by a step parameter, α, we can update the virtual goal and iterate until specific distance criteria are met ($d < d_{threshold}$). Choosing a fixed value for α can create convergence problems like any gradient-based method (see [153] for a better explanation and possible solutions).

The local planner $DubinsSteer$, shown in Alg. 11, calculates a path connecting a given vertex v_{near} to v_{new} using an extended 3D version of the Dubins' Airplane paths. As described in [96] we consider that the airspeed, v_a, is constant and that the roll angle ϕ can only assume three values $\{\phi_{min}, 0, \phi_{max}\}$ (in other words, the aircraft is capable of turning in maximum rate or follow a straight heading angle), and the flight path angle, γ, can assume any value within its limits $[\gamma_{min}, \gamma_{max}]$. Therefore, $DubinsPath$ solves the sequence of inputs $u = [v_a, \gamma, \phi]^T$ that takes the system from v_{near} to v_{new} in the wind reference frame, i.e., without considering the effect of the wind.

An example of this procedure is given in Fig. C.1. Alternatively, an approach that also tries to solve the time-optimal path planning problem for nonholonomic vehicles based on

Algorithm 11 Steering using Dubins' Airplane and wind

function $S_{wind}, c_{min} = \text{DUBINSSTEER}(v_{near}, v_{new}, \alpha)$
  $v_{new}^{virtual} \leftarrow v_{new}$
  while $(d > threshold)$ **do**
    $\nabla d \leftarrow GetGradient(\Delta, p_{new}^{virtual})$
    $v_{new,j}^{virtual} \leftarrow v_{new,j}^{virtual} + \alpha \nabla d$
    $\sigma \leftarrow DubinsPath(v_{near}, v_{new})$
    $S_{wind} \leftarrow DeformTrajectory(\sigma, N, v_a)$
    $d \leftarrow |S_{wind}(end) - v_{near}|$
  end while
  $T \leftarrow RequiredThrust(S_{wind})$
  $c_{min} \leftarrow Cost(v_{near}, \sigma_{wind}, T)$
end function

the classification of Dubin's paths was proposed recently in the preprint [26].

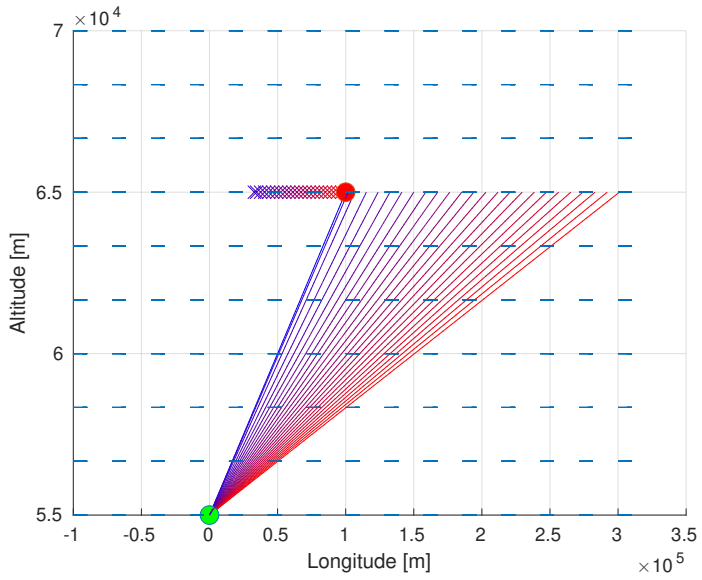


Figure C.1: Gradient-based iteration process to find a virtual goal leading to a ground path reaching the desired goal. The vehicle must go from the start position (green dot) to the end position (red dot). By deforming with the wind the path that was calculated using Dubins Airplane, and with these two points, we get the red line on the right side. By calculating a distance gradient, we obtain a direction to move the virtual goal (to the left) that will make the endpoint of the wind-deformed path closer to the desired goal (red dot). The final virtual goal is shown as a blue cross.