Silesian University of Technology
Faculty of Mechanical Engineering
Department of Fundamentals of Machinery Design

Doctoral dissertation

Model-Based Adaptive Path Planning Algorithm for Unmanned Aerial Vehicles

MSc, Eng. Mateusz Kosior

Primary Supervisor:
PhD, DSc, Eng. Piotr Przystałka, Associate Professor

Secondary Supervisor:
PhD, Eng. Wawrzyniec Panfil, Assistant Professor

Gliwice, 2022

Contents

List of used acronyms

Algorithms

$ACO_{\mathbb{R}}$	Ant Colony Optimization for Continuous Domains
APP	Adaptive Path Planner (GPP + LPP)
GA	Genetic Algorithm
GPP	Global Path Planner
I-GWO	Improved Grey Wolf Optimization
LPP	Local Path Planner
PSO	Particle Swarm Optimization
RRT	Rapidly-exploring Random Tree
VG	Visibility Graph

Unmanned vehicles and airspace

AUP	Airspace Use Plan
HALE	High-Altitude Long Endurance
HAPS	High-Altitude Pseudo Satellite
RPA(S)	Remotely Piloted Aircraft (System)
NFZ	No-Fly Zone
RTH	Return-To-Home (safety maneuver)
TS	Twin Stratos (a HALE UAV)
UA	Unmanned Aircraft
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UMS	Unmanned System
UUP	Updated Airspace Use Plan
UxV	Unmanned Vehicle (general term)

Metrics

CT	Computation time
CFE	Cost function evaluations
COST	Cost returned by the cost function
EEE	Estimated energy expenditure
FT	Time of flight
LEN	Path length
NCOL	Number of colliding waypoints
NTREE	Number of RRT vertices in the whole tree
NPTS	Number of waypoints along a path
NRRT	Number of RRT vertices on the shortest path
SMOO	Path smoothness

Institutions, regions, authorities

CEDD	Central European Drone Demonstrator (Poland)
EASA	European Union Aviation Safety Agency
ICAO	International Civil Aviation Organization
LEADER	Long-endurance UAV for collecting air quality data with high spatial and temporal resolutions (project name)
NIST	National Institute of Standards and Technology
NORCE	Norwegian Research Centre
PANSA	Polish Air Navigation Services Agency
SAE	Society of Automotive Engineers
SUT	Silesian University of Technology (Poland)
UW	University of Warsaw (Poland)

Measurements and meteorology

BC	Black Carbon
ECMWF	European Centre for Medium-Range Weather Forecasts (weather model)
GFS	Global Forecast System
PBL	Planetary Boundary Layer
SLCF	Short Lived Climate Forcer

Other acronyms

AGL	Above Ground Level
AMSL	Above Mean Sea Level
ENU	East-North-Up coordinate system
GCS	Ground Control Station
HIL	Hardware-in-the-Loop
MBD	Model-Based Design
MIL	Model-in-the-Loop
NED	North-East-Down coordinate system
PIL	Processor-in-the-Loop
ROI	Region of Interest
SIL	Software-in-the-Loop
SRTM	Shuttle Radar Topography Mission

List of used symbols

Aircraft modeling

$\mathcal{D}_{2D}$	admissible Dubins car path
$\mathcal{D}_{3D}$ or $\mathcal{D}$	admissible Dubins airplane path
$\mathcal{F}^i$	inertial frame
$\mathcal{F}^v, \mathcal{F}^{v1}, \mathcal{F}^{v2}$	vehicle frames
$\mathcal{F}^b$	body frame
$\mathbf{i}, \mathbf{j}, \mathbf{k}$	unit vectors along x, y and z axes
$\vec{v}_a$	velocity relative to air (airspeed)
$\vec{v}_g$	velocity relative to ground
$\vec{v}_w$	velocity relative to wind
ψ	heading or yaw angle
θ	pitch angle
ϕ	roll angle
γ	inertial-referenced flight-path angle
γ_a	air-mass-referenced flight-path angle
χ	course angle
χ_w	horizontal wind angle
χ_c	crab angle
β	sideslip angle
α	angle of attack
p_n, p_e, h	inertial north, east and height (negative down) UAV positions in $\mathcal{F}^i$
b	controller constant
$(\cdot)^c$	(as superscript) controlled variable
g	standard acceleration due to gravity
F_{drag}, F_{lift}	drag and lift force, respectively
F_{thrust}	thrust force of the propulsion
ρ	air density
S	total wing area
C_D, C_L	drag and lift coefficient, respectively
m	total mass of the aircraft

Operators

$(\cdot)^T$	transposition
$\lVert\cdot\rVert$	length of a vector or a segment
$\texttt{card}(\cdot)$	cardinality
$\texttt{free}\,(\cdot)$	obstacle presence at a waypoint
$\texttt{wind}\,(\cdot)$	wind velocity at a waypoint
$\texttt{pollution}\,(\cdot)$	pollution intensity at a waypoint
$\angle(\cdot)$	angle between the arguments
$\mathcal{N}(\cdot)$	normal (Gaussian) distribution

Environment maps

$\mathcal{M}$	environment map (general term)
$\mathcal{M}_t$	terrain map (general term)
$\mathbf{M}_t^V$	voxel-based terrain map
$\mathbf{M}_t^D, \mathbf{M}_t^C$	elevation based terrain map: discrete and continuous, respectively
$\mathcal{M}_w$	wind map (general term)
$\mathbf{M}_w^D, \mathbf{M}_w^C$	discrete and continuous wind map, respectively
$\mathcal{M}_a$	airspace map
$\triangle^{2D}$	horizontally-aligned polygon
$\triangle^{3D}$	vertically-aligned prism (NFZ ROI)
$\mathcal{M}_m$	measurement map (general term)
$\mathbf{M}_m^D, \mathbf{M}_m^C$	discrete and continuous measurement map, respectively
$\mathcal{M}_o$	obstacle map (terrain map + airspace map)

Adaptive Path Planner

$\mathcal{P}^G, \mathcal{P}^L$	global and local path (general term)
$\mathbf{w}, \mathbf{W}$	waypoint and an array thereof
$\mathbf{C}, C$	multi- and single-objevitve cost function
c_n	cost function of the n-th criterion
$\mathcal{S}, \mathbf{p}$	scenario and its parameters
Ω	constraints and boundaries function
Ω_k	kinematic constraints of the UAV
$\mathbf{g}$	array of global mission-specific settings
s_{fn}	scaling factor of the n-th criterion
P_{ctrl}	power required by the UAV controller and peripherals
$\mathbf{E}$	array of reference energy data
$\triangle^m$	horizontally-aligned measurement polygon

General denotation concept

$\vec{x}$	vector in strict mathematical sense
$\mathbf{x}, \mathbf{X}$	(bold) 1D and nD arrays, $n > 1$
x, X	(regular italics) scalar
$\mathcal{X}$	(calligraphic) general (abstract) term
x_n	(numerical subscript) x indexed by n
x^n, x_n	(superscript or non-numerical subscript) context-dependent annotation

1. Introduction

In recent years, Unmanned Aerial Vehicles (UAVs) have been gaining more and more popularity [1–3], despite the fact the historical development of UAVs began in the mid-19th century [1]. According to Cifaldi et al., the first Remotely Piloted Aircraft Systems (RPASs) date back to the mid-1800s [4].

Zhao et al. noted a growth in the number of papers concerning path planning for UAVs. The rate of growth was increasing slightly for the first 7 years from 2008. Between 2014 and 2016 there was a slight fluctuation. They reported, however, that the years 2017 and 2018 showed a more significant increase [5].

According to Pehlivanoglu et al., UAV will be more effective and cost efficient when they achieve the ability to intelligently modify their internal plan based on the environment. Path planning, especially adaptive path planning, is one of the areas which could benefit from increased autonomy [3]. Thus, path planning and trajectory[1] generation are both fundamental areas for the development of Unmanned Aerial Systems (UASs). Both provide the ability to figure out a way to efficiently and safely travel through an environment under a set of constraints [6].

It is important to note obstacles are not only solid objects. Weather conditions also play an important role in aviation. Garcia et al. argue that complicated weather conditions caused many fatal aircraft accidents. Moreover, UAVs are more susceptible to unfavorable weather than manned aircrafts – mostly due to the usually small size of the former. Hence, avoiding dangerous weather conditions is a key issue in unmanned flights [7].

In their work from 2020, Dhulkefl et al. highlight great attention that UAVs received over the past 10 years – both military and civil applications [1]. UAVs are willingly used to perform tasks considered unnecessary, repetitive or too dangerous for manned operations [6].

Introducing combat UAVs is one of the trends clearly visible in modern aerial weapon equipment [8, 9]. As noted by Ling and Hao, more and more UAVs have been used to carry out military missions. These missions include target detection, target destruction, close-in reconnaissance, precision bombing, high definition recording, electronic warfare and damage assessment [3, 9]. Contemporary UAVs are often employed to track, protect, or provide surveillance of a ground-based vehicle [10]. Especially useful for this task are multi-UAV convoys [11]. The research on UAVs is considered fundamental to battle effectiveness of the air force and thus is strictly related to safeness of nations [8].

Civil UAV applications are especially popular in Europe. According to a research by Cifaldi et al., in April 2016 there were 2500 European operators compared to 2342 operators located in the rest of the world. Moreover, they convince it is estimated the UAV industry may be worth 10% of the aviation market. Estimated demand could reach 10 billions EUR per year by 2035, and over 15 billions EUR per year by 2050 [4]. Civil applications include goals, such as reconnaissance, search and rescue, weather observation, aerial mapping and cinematography [3, 6]. Also, UAVs in construction sites provide constant monitoring, access hardly-reachable and dangerous areas and support post-disaster reconstruction [12].

Nevertheless, many of these missions are restricted by the capacity of fuel tanks or batteries. Therefore, many UAVs are solar-powered. High-Altitude Long Endurance (HALE) UAVs are a subgroup of these aircrafts. HALE UAVs are highly-efficient unmanned aircrafts designed to

[1] Schøler et al. distinguish the term *trajectory* from *path* – a trajectory includes time, while a path does not [6]

fly above the atmospheric flight region and below the spacecraft flight region (approximately 20 to 100 km) [13]. Depending on the efficiency of the platform, as well as sun conditions, they have the potential to stay airborne indefinitely. Moreover, due to their maneuverability, they are a serious alternative to balloons and airships. HALE UAVs can take the role of measurement platforms and also satellites. Hence their alternative name, High-Altitude Pseudo Satellites (HAPSs) [13].

Atmospheric density at 20 km is less than 10% of such at sea level. Dynamic pressure available to lift an airplane is minimal and is compensated with high velocities [13]. Therefore, reaching such altitude is a challenge for a streamlined aircraft with relatively humble battery capacity. Nevertheless, finally reaching that altitude grants the nearly-infinite source of solar power. This promise encouraged several teams of scientists to develop such aircrafts since the first solar powered flight *Sunrise* in 1974 [13].

This thesis tries to address some of this issues found in fixed-wing meteorological measurement platforms, especially HALE UAVs. The thesis proposes a two-level Adaptive Path Planner (APP), which provides an optimized path for such aircraft. The path must assure safety and reliability of the mission. To be feasible, the path also must be subject to the constraints of the UAV. Thus, APP considers obstacle avoidance and optimization of the path according to minimal energy expenditure and beneficial wind conditions.

1.1 Scientific background and motivation

The thesis is directly related to the major task of international project *Long-endurance UAV for collecting air quality data with high spatial and temporal resolutions* (LEADER)[2]. The goal for the project is to develop a High Altitude Long Endurance (HALE) UAV as a mobile research platform for measuring the composition of the atmosphere via e.g. sampled aerosol measurement [14]. The EU review [15] on black carbon (BC) highlights the role observations from ships or aircraft play in understanding distinct episodes of long-range transport.

Forest fires as well as contrails induced by air traffic in the tropopause region are the major sources of BC transport [14]. Gilardoni et al. in their report [16] define BC as “a component of submicrometer aerosol particles”, which “impacts the regional radiation balance by absorbing incoming solar radiation (direct effect), altering cloud distribution and their radiative properties, and reducing snow and ice surface albedo after deposition” [16].

Nevertheless, the authors of [15] note the extreme costs of intensive field campaigns related to BC observations. Thus, providing an energy-efficient, mostly autonomous and thus relatively cheap scientific platform for investigating the sources, sinks and transport of BC becomes crucial. Moreover, it will help to improve the knowledge how the transport of both natural and anthropogenic aerosols impacts clouds and their properties [14].

The LEADER project addresses these concerns and recommendation. The project is a direct response to the POLNOR 2019 call according to the programme *Applied Research* financed by The National Centre for Research and Development (Poland)[3]. The project is held by the Silesian University of Technology (SUT, Poland) in collaboration with the Norwegian Research Centre (NORCE, Norway), the University of Warsaw (UW, Poland) and SkyTech ELAB Sp. z o.o. (Poland). The project is multidisciplinary and encompasses engineering and technology, mechanical engineering and aerospace engineering [14]. The missions conducted during the project are divided into two categories based on the geographical region.

[2] Homepage of the LEADER project: `https://polnor-leader.eu/?lang=en` [27.06.2022]

[3] Homepage of the Applied Research programme: `https://eeagrants.org/archive/2014-2021/programmes/PL-Applied%20Research` [27.06.2022]

1.1.1 Missions in Poland

The Polish missions considered by the LEADER project focus on evaluating the performance of the HALE UAV and its equipment. The missions will be carried out over selected regions in southern Poland to measure the spatial distribution of pollutants, that is:

- Smog distribution in the planetary boundary layer (PBL) during cold season.
- Biomass burning and mineral dust events in the middle and upper troposphere (up to 15 km) during spring and summer.
- Vertical transport of aerosol through PBL [14].

Smog distribution

The smog in Poland is a major concern, especially due to the increased emission from hard coal burning by heating systems in the winter months. Previous research provided information about smog concentration at lower altitudes. Nevertheless, the 3D aerosol distribution in megacities and rural areas is still unknown. For this purpose, low altitude flights, i.e., up to 1–2 km, are planned for vertical and horizontal gradient measurements. They will be carried out mostly over Central European Drone Demonstrator (CEDD)[4] due to the legal regulations in Poland [14].

Biomass burning and mineral dust events

The biomass burning and mineral dust transport usually occurs in the middle and upper troposphere. The HALE UAV is therefore expected to fly between 3 km and 12 to 15 km. Unlike smog distribution, the transport is usually observed on a large scale, so the measurements can be done over any region of Poland. The strategy of this measurement will focus on vertical profile than horizontal gradient flight [14].

Measurements of biomass burning and dust events can be done between spring and late summer. During spring the source of biomass burning is usually grass fires in Ukraine and Russia (Kaliningrad Oblast). In the summer, forest wildfires are the main sources, where the long-range transport from Ukraine, Siberia, Canada and the USA can be observed in the upper troposphere as well as in the lower stratosphere [14].

Vertical transport of aerosol

The vertical transport of aerosol will be measured between free atmosphere and PBL. The UAV will fly mostly in the lower troposphere, as the height of PBL usually varies between 0.5 and 2.5 km. Unlike previous measurement objectives, the measurements of vertical aerosol transport are unrestricted by season. Like biomass burning, the exact localization of the measurements is not important [14].

The research will be carried out as horizontal or vertical profiles. Horizontal profiling aims at measuring the horizontal gradient of pollution concentration in the lower troposphere (up to 2.5 km). The focus of the vertical profiles is to understand the relation between variability of pollution concentration and thermodynamic parameters. The measurements conducted by the UAV will be used in conjunction with ground-based observations performed in the mobile laboratory [14].

[4] Homepage of the CEDD project (in polish): `https://cedd.pl` [28.11.2021].

1.1.2 Missions in the Arctic

The Arctic missions in the LEADER project will focus on investigating the role of the Short Lived Climate Forcers (SLCFs) on the climate. The lack of SLCF measurements in the eastern part of the Arctic is noted by the AMAP assessment [17], for example.

The missions will be held over Svalbard (Norway). The major regions of interest include the surroundings of Ny-Ålesund, the glaciers Holtedahlfonna and Kongsvegen (Fig. 1.1), as well as eastern regions of Svalbard [14].

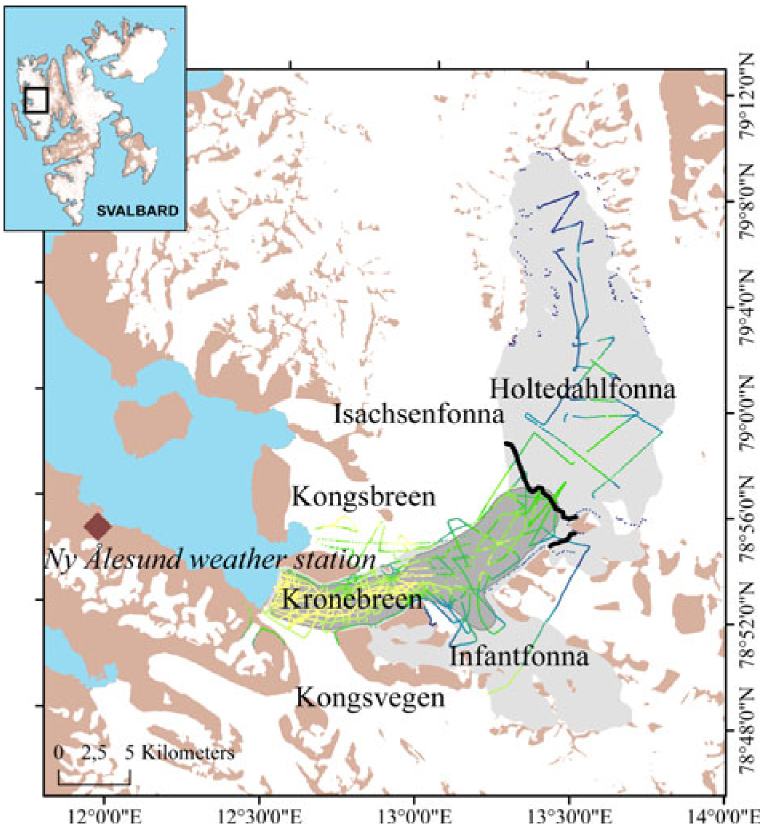

Fig. 1.1: Glaciers found in Svalbard [18]

The scope of the LEADER project considers several measurements in the Arctic, such as:

- The profiles and spatial variability of BC concentration.
- Aerosol composition for source validation, and then to compare forest fires with antropogenic sources.
- Cloud condensation nuclei.
- Meteorological parameters such as temperature and relative humidity.
- Atmospheric turbulence and radiation [14].

1.2 Aim

The aim of the dissertation is to design and verify the model-based adaptive path planning algorithm for pollution sampling with a HALE UAV which flies autonomously in a limited environment. The design principles of the algorithm must allow it to be feasible for deployment in the UAS designed for the LEADER project.

1.3 Scientific problem

To conduct pollution measurements, especially at higher altitudes, the HALE UAV developed for the LEADER project is required to stay airborne for prolonged amounts of time. Nevertheless, the energy resources of the UAV are very limited and the aircraft is subject to much more restrictive kinematic constraints than, for example, a multirotor. Therefore, to fulfill the requirements of the project, the flight path of the UAV must be optimized for minimal energy expenditure while subject to kinematic constraints of the HALE aircraft. Optimization must also take into account the environment of the UAV to minimize detrimental effects of wind and to avoid collisions with potential obstacles, which requires the environment model. Moreover, the environment can change during a mission, so the planner must be adaptive. Based on these requirements and limitations, the following scientific tasks were identified:

- Formulate the theoretical basis of a model-based adaptive path planning algorithm for pollution sampling.
- Formulate the theoretical basis of an environment model, which considers the components of the scene important for flight safety and energy conservation.
- Define the multi-criteria optimization problem considering obstacle avoidance, minimum energy expenditure, measurement strategy and kinematic constraints of the HALE aircraft and verify the results in model-based simulations.
- Identify the optimization criteria and formulate the objective function.
- Compare several different path planning and optimization algorithms and find the best one for energy-efficient adaptive path planning.
- Compare the energy expenditure measured for the path generated by the algorithm and for the reference path supplied by a human expert.
- Validate the adaptive path planning algorithm in simulation on use cases similar to these described in the LEADER project, that require dynamic re-planning of the path.

1.4 Scope

The thesis focuses on the model-based Adaptive Path Planner (APP) for a UAV, especially its application for pollution sampling with a HALE aircraft. The APP algorithm features dynamic recalculation of the flight path while the UAV is airborne to adapt to changing mission parameters, such weather conditions or the appearance of a new obstacle. Note that APP is not meant to handle the flight in dangerous weather conditions – its task is to adapt to environment to avoid such scenarios.

The dissertation addresses many issues in the field of mechanical engineering, including the formal description of a novel model-based adaptive path planning algorithm, optimization for minimum estimated energy expenditure, modeling the UAV and its environment, as well as a verification study, which employs Model-in-the-Loop (MIL) simulations.

Chapter 1 contains an introduction to the scientific problem raised in the thesis, as well as its motivation and scientific background. The chapter covers also some of the measurement scenarios considered in the LEADER project, and thus in the thesis.

Chapter 2 describes the fundamentals of modeling a fixed-wind UAV, such as coordination frames, wind triangle and Dubins airplane paths. Also, a few simplified guidance models (four kinematic and one dynamic) of the aircraft are presented. It should be noted that while the LEADER project also addresses developing the full non-linear dynamic model of a HALE aircraft, it is a task handled by other team. Hence, it is beyond the scope of the thesis.

Chapter 3 introduces the concept of the environment map, which is used to model the crucial components of the scene. For this purpose, four distinct models (maps) are proposed: a terrain map, a wind map, an airspace map and a measure map.

Chapter 4 contains the formal description of the model-based APP algorithm and its components. First, a general idea of adaptability in path planning is discussed. Then, the general form of APP is given specifying the role of model-based approach. Afterwards, the components of APP: Global Path Planner (GPP) and Local Path Planner (LPP) are described independently. GPP implements a global optimization algorithm to solve a multi-criteria problem of providing a feasible path optimized for minimum energy expenditure. LPP uses fast stochastic algorithms to provide a fallback path in case of emergency. The path must be collision-free and subject to the kinematic constraints of the UAV, but its optimality is sacrificed for computation performance.

Chapter 5 describes brief a verification study on environment and UAV models, as well as the extensive model-based validation tests of APP in simulation. The first part compares different implementations of the terrain map and validates the wind map. The second one focuses on GPP. The effects of the global optimization criteria are verified independently. Then, various flavors of single-objective global optimization algorithms are compared in MIL simulation and the optimal solution for GPP is chosen. The third part covers the extensive tests of LPP, including the comparison of different RRT-based algorithm, choosing the optimal one, and then calibrating it on a challenging environment map. Next, LPP is validated on the series of use cases, which require adaptive re-planning. Finally, APP is validated on two mission scenarios inspired by the LEADER project.

The final pages summarizes the research, while focusing on the author's contribution to modeling and model-based adaptive path planning. Detailed conclusions and considerations for future work presented afterwards conclude the thesis.

Appendix A contains definitions of selected terms related to path planning, optimization and autonomy, which are important for the thesis.

Appendix B is a brief state-of-the-art analysis of different path planning and global optimization algorithms. The first part focuses on path planning algorithms – both exact (Dijkstra's, A*, D*) and stochastic (RRT and RRT*). The second part describes selected general optimization algorithms for global optimization (GA, PSO, ACO and I-GWO).

1.5 Acknowledgments

The research described in the thesis was carried out and financed under the project *Long-endurance UAV for collecting air quality data with high spatial and temporal resolutions (LEADER)*, which is implemented under the programme *Applied Research* under Norwegian Financial Mechanisms 2014 – 2021 (NOR/POLNOR/LEPolUAV/0066/2019-00).

Wind forecast data was acquired via Meteomatics Weather API Connector for MATLAB [19]. The research used a license available thanks the courtesy of Meteomatics AG.

The major part of computations was done in MATLAB. License for MATLAB, as well as for its several toolboxes, was provided by the Department of Fundamentals of Machinery Design of Silesian University of Technology.

The thesis was written as the part of the project *SymIn – Simulations in Engineering*, implemented as part of module V – Doctoral studies, task 28 – Interdisciplinary training at the Doctoral School under the project *Silesian University of Technology as a Center for Modern Education based on research and innovation* (POWR.03.05.00-00.z098/17-00).

2. UAV modeling

The chapter focuses on the basic concepts of path description, as well as on kinematic and dynamic models of the UAV. Particular attention is paid to the UAV guidance models used by the model-based simulations in the verification study.

2.1 Coordination frames

A model may use many different coordinate systems or *frames*. Choosing the adequate frame depends on the context. For example, Beard and McLain in [20] define seven different coordination frames. The frames used throughout the thesis are described below. Unless otherwise stated, the thesis refers to the inertial frame by default. Other frames frequently used in aerospace engineering, as the stability frame and the wind frame, are described in-detail, e.g., in [20].

Inertial frame

The *inertial* (or *North-East-Down*, NED [20])[1] frame $\mathcal{F}^i$ is the coordinate system fixed to earth. Its origin is defined at an arbitrary home location [20]. Hence, $\mathcal{F}^i$ (Fig. 2.1) is completely static, i.e., its position and orientation never changes. It can be understood as a global coordinate system, to which other frames refer. Unit vectors $\mathbf{i}^i$, $\mathbf{j}^i$, $\mathbf{k}^i$ reflect axes x, y, z, respectively.

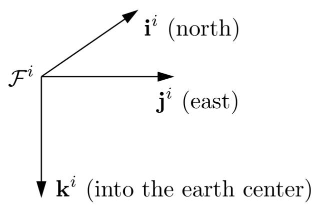

Fig. 2.1: The inertial coordinate frame $\mathcal{F}^i$ [20]

Vehicle frames

The *vehicle* frame $\mathcal{F}^v$ (Fig. 2.2) is bound to the UAV's center of mass, and thus follows the position of the aircraft. Axes, however, are always aligned to the inertial frame.
The *vehicle-1* frame $\mathcal{F}^{v1}$ is the same as $\mathcal{F}^v$ with added positive right-handed rotation ψ (*heading* or *yaw angle*) around $\mathbf{k}^v$ (Fig. 2.3, on the left) [20]. In other words $\mathcal{F}^{v1}$ considers the heading of the UAV, but not the two other angles.

The *vehicle-2* frame $\mathcal{F}^{v2}$ bases on $\mathcal{F}^{v1}$, but adds another positive right-handed rotation θ (*pitch angle*) – this time around j^{v1} (Fig. 2.3, on the right) [20]. So $\mathcal{F}^{v2}$ is aligned with the UAV, excluding its roll angle.

[1] Named according to the convention to align x and y axes to the north and the east, respectively. The resulting z axis points down to complete the right-handed CS.

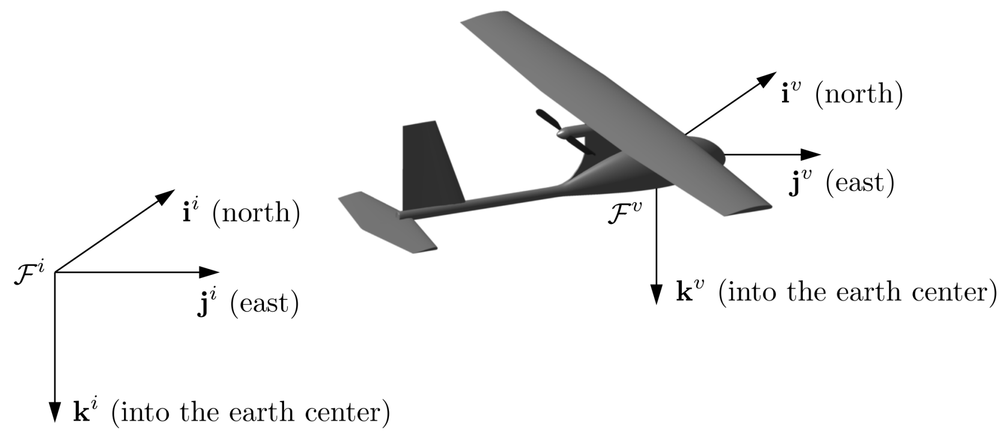

Fig. 2.2: The vehicle frame $\mathcal{F}^v$ [20]

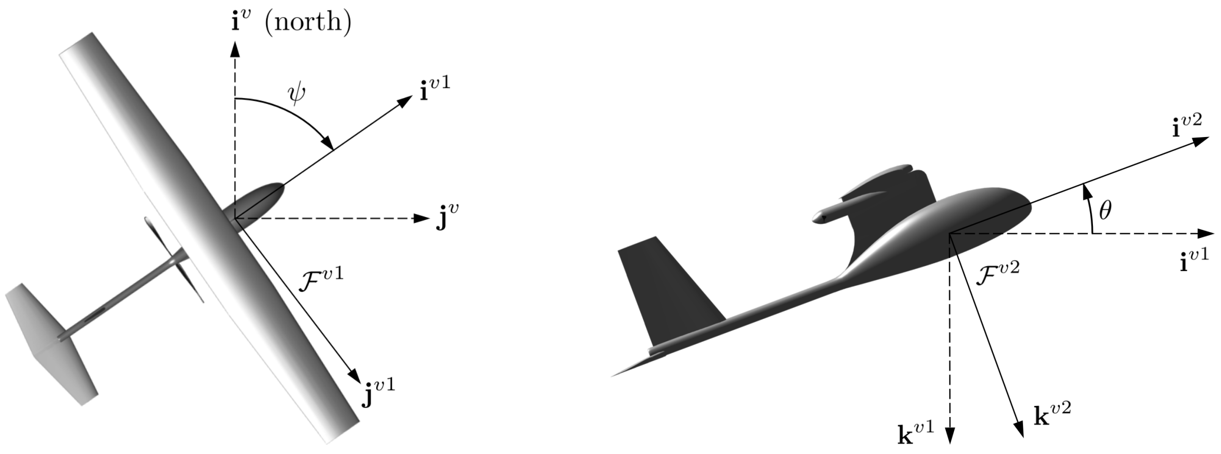

Fig. 2.3: The vehicle-1 frame $\mathcal{F}^{v1}$ (left) and vehicle-2 frame $\mathcal{F}^{v2}$ (right) [20]

Body frame

The UAV's *body* frame $\mathcal{F}^b$ adds up on the $\mathcal{F}^{v2}$ by adding the final, third rotation. It rotates the $\mathcal{F}^{v2}$ around $\mathbf{i}^{v2}$ by the *roll angle* ϕ, as seen in Fig. 2.4. $\mathcal{F}^b$ is fully aligned with the aircraft, unlike $\mathcal{F}^v$, $\mathcal{F}^{v1}$ and $\mathcal{F}^{v2}$. The unit vector $\mathbf{i}^b$ points out of the nose of the aircraft and $\mathbf{j}^b$ is perpendicular to $\mathbf{i}^b$ and points out of the right wing. The last unit vector, $\mathbf{k}^b$, is perpendicular to the other two unit vectors and points out of the UAV's belly [20].

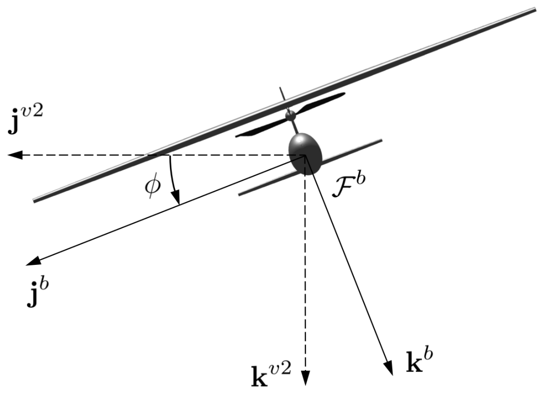

Fig. 2.4: The body frame $\mathcal{F}^b$ (front view)

2.2 Wind triangle

Wind influences heavily the performance of UAVs, especially the smaller ones. For example, wind often accounts for 20% to 50% of the total airspeed of miniature air vehicles[2]. Wind is also a major factor for HALE aircrafts, that try to conserve energy by using their propulsion sparingly [20].

Let $\vec{v}_a$ be *airspeed* of the UAV, that is, its velocity vector relative to the surrounding air mass flowing around it. Now, let $\vec{v}_g$ be the UAV's velocity relative to ground and $\vec{v}_w$ be the wind velocity. The relation between the three vectors is called the *wind triangle*. In vector notation it is given by

$$\vec{v}_a = \vec{v}_g - \vec{v}_w. \tag{2.1}$$

The vector $\vec{v}_g$ is defined relative to $\mathcal{F}^i$ using the *course angle* χ and the *inertial-referenced flight-path angle* γ. The *course angle* χ is measured in the horizontal plane from $\mathbf{i}^i$ (true north) to the $\vec{v}_g$ projection, as shown in Fig. 2.5. Similarly defined is the *horizontal wind angle* χ_w. The angle ψ is the yaw angle, as discussed in section 5. The angle χ_c between $\mathbf{i}^b$ and $\vec{v}_g$ is called the *crab angle*, while β represents the *sideslip angle*, which is zero in steady and level flight.

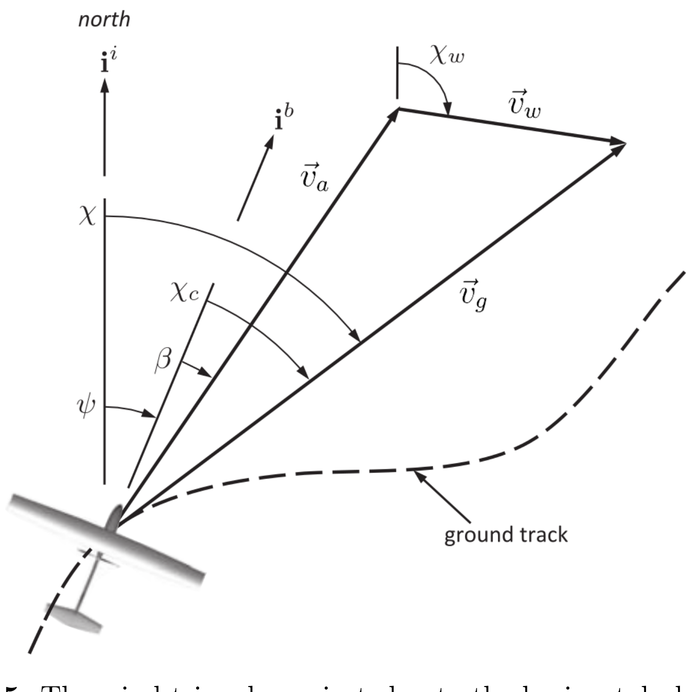

Fig. 2.5: The wind triangle projected onto the horizontal plane [20]

Fig. 2.6 illustrates the angle γ, which is measured in the vertical plane between the projections of $\vec{v}_g$ and the horizontal plane. The angle γ_a denotes the *air-mass-referenced flight-path angle*, i.e., the angle between the projection of $\vec{v}_a$ and the horizontal plane. The other angles are the pitch angle θ (see section 5) and the *angle of attack* α. If the vertical component of $\vec{v}_w$ is zero, so is α.
According to Beard and McLain, "in steady, level flight, $\vec{v}_a$ is commonly aligned with $\mathbf{i}^b$, meaning the sideslip angle β is zero" [20]. Hence, considering the angles given above and assuming β is

[2] That is, the aircrafts with wingspan of less than 1.52 m (or 5 feet [20]).

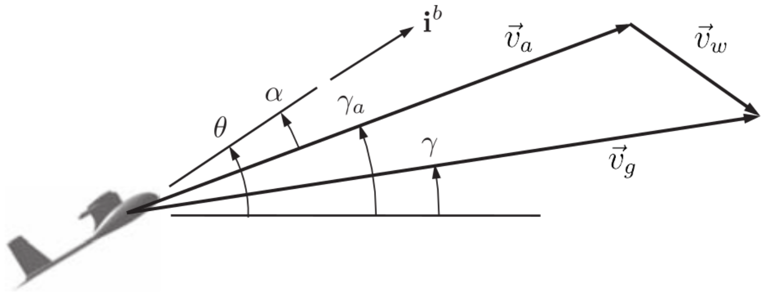

Fig. 2.6: The wind triangle projected onto the vertical plane [20]

zero, the wind triangle equation (2.1) expressed in $\mathcal{F}^i$ can be alternatively formulated as

$$v_g \begin{bmatrix} \cos\chi\cos\gamma \\ \sin\chi\cos\gamma \\ -\sin\gamma \end{bmatrix} - \begin{bmatrix} v_{wn} \\ v_{we} \\ v_{wd} \end{bmatrix} = v_a \begin{bmatrix} \cos\psi\cos\gamma_a \\ \sin\psi\cos\gamma_a \\ -\sin\gamma_a \end{bmatrix}. \tag{2.2}$$

where $v_g = \|\vec{v}_g\|$, $v_a = \|\vec{v}_a\|$ and v_{wn}, v_{we}, v_{wd} denote the NED components of $\vec{v}_w$,, i.e., along $\mathbf{i}^i$, $\mathbf{j}^i$, $\mathbf{k}^i$, respectively.

2.3 Kinematic guidance model

To verify the proposed algorithm, the fixed-wing UAV will be modeled with a reduced-order kinematic guidance model. The model is described in-detail by Beard and McLaine in [20]. The main simplification of the model is eliminating complex aerodynamic forces acting on the airframe. This implies force- and moment-balance equations of motion are deliberately neglected. The model, however, considers wind influence via vector summation and also covers the behavior of the aircraft's autopilot.

Kinematic guidance models of a fixed-wing UAV are available in different flavors. The simplest model assumes the autopilot controls airspeed, altitude and course angle. The corresponding equations of motion are given by

$$\begin{cases} \dot{p}_n = v_a\cos\psi + v_{wn} \\ \dot{p}_e = v_a\sin\psi + v_{we} \\ \ddot{\chi} = b_{\dot{\chi}}\left(\dot{\chi}^c - \dot{\chi}\right) + b_\chi\left(\chi^c - \chi\right) \\ \ddot{h} = b_{\dot{h}}\left(\dot{h}^c - \dot{h}\right) + b_h\left(h^c - h\right) \\ \dot{v_a} = b_{v_a}\left({v_a}^c - v_a\right) \end{cases}$$

where p_n and p_e are horizontal positions in $\mathcal{F}^i$ along north and east axes and h is height. The variable v_a denotes airspeed, ψ is the yaw angle in $\mathcal{F}^{v1}$ and χ represents course angle in $\mathcal{F}^i$. Wind velocity vector is described by components v_{wn}, v_{we}, v_{wd} in the inertia frame $\mathcal{F}^i$ (NED)[3]. Autopilot's constants are given as b with the corresponding subscript. The c superscript denotes the variable controlled by the autopilot [20].

Alternatively, course angle may not be controlled directly by the rudder. Instead, roll angle can be used according to the coordinated-turn condition, which is thoroughly explained in [20].

[3] This convention assumes wind is blowing to the positive direction of the axis. For example, positive w_n means wind blowing from south to north. Conversely, it would be called southerly wind using the meteorological convention.

As the result, the equations of motion are modified to

$$\begin{cases} \dot{p}_n = v_a \cos\psi + v_{wn} \\ \dot{p}_e = v_a \sin\psi + v_{we} \\ \dot{\psi} = \dfrac{g}{v_a}\tan\phi \\ \ddot{h} = b_{\dot{h}}\left(\dot{h}^c - \dot{h}\right) + b_h\left(h^c - h\right) \\ \dot{v_a} = b_{v_a}\left({v_a}^c - v_a\right) \\ \dot{\phi} = b_\phi\left(\phi^c - \phi\right) \end{cases}$$

where ϕ denotes the roll angle in $\mathcal{F}^b$ and g is the standard acceleration due to gravity. The third model further increases fidelity by controlling altitude indirectly through the flight-path angle. The model is described by the equations

$$\begin{cases} \dot{p}_n = v_a \cos\psi\cos\gamma_a + v_{wn} \\ \dot{p}_e = v_a \sin\psi\cos\gamma_a + v_{we} \\ \dot{h} = v_a\sin\gamma_a - v_{wd} \\ \dot{\chi} = \dfrac{g}{v_g}\tan\phi\cos\left(\chi - \psi\right) \\ \dot{\gamma} = b_\gamma\left(\gamma^c - \gamma\right) \\ \dot{v_a} = b_{v_a}\left({v_a}^c - v_a\right) \\ \dot{\phi} = b_\phi\left(\phi^c - \phi\right) \end{cases} \tag{2.3}$$

where γ is flight-path angle in $\mathcal{F}^i$ and v_g denotes the velocity of the aircraft relative to ground. These guidance models assume the fixed-wing UAV flies under a coordinated-turn condition, with zero side-slip (steady flight) [20]. The autopilot included in the model controls airspeed, flight-path angle and roll angle. Despite considering wind influence, the model is relatively simple when compared to dynamic models. Thus, it is possible to process it using the UAV's onboard computation module. Unless otherwise specified, the model (2.3) will be the default kinematic guidance model used in the thesis.

Nevertheless, the concept of kinematic guidance models can be further expanded, e.g., by implementing load factor. It will not be used in this research, however. Further information on kinematic guidance models can be found, e.g., in [20].

2.4 Dynamic guidance model

By neglecting aerodynamic forces entirely, the kinematic guidance model introduces an important limitation especially visible in the case of HALE UAVs. Lift force changes significantly between starting/landing conditions (near ground level) and the maximal achievable altitude of several kilometers.

As noted above, kinematic models are usable to be implemented in embedded controllers due to their relative simplicity and low computation effort. Nevertheless, if limited computation performance is not an issue, e.g., during simulation in the ground control station, the aerodynamic forces should not be neglected.

A dynamic guidance model includes lift, drag, and thrust forces aligned as in Fig. 2.7. Therefore, it can successfully model the behavior of the HALE UAV in changing air density. Nevertheless, the dynamic guidance model is still simpler than a full non-linear aircraft model presented, e.g., in [20].

Introducing dynamics to the model requires defining forces affecting the aircraft. Let F_{thrust} be the resultant thrusting force component generated by the propulsion, which acts along the

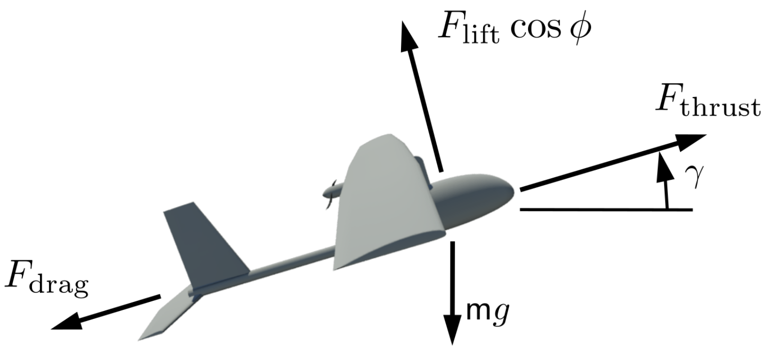

Fig. 2.7: External forces acting on the UAV along the i^b axis at roll angle of ϕ [20]

aircraft's $\mathbf{i}^{v2}$ axis. Now, assume F_{drag} represents drag force acting along the same axis, but having opposite direction

$$F_{drag} = \frac{1}{2}\rho v_a^2 S C_D$$

where ρ represents air density, S is total wing area and C_D denotes drag coefficient. The last force considered by the dynamic guidance model is lift F_{lift} given by Bernoulli's law

$$F_{lift} = \frac{1}{2}\rho v_a^2 S C_L$$

where C_L denotes the aircraft's lift coefficient. The corresponding equations of motion for the model are finally given by

$$\begin{cases} \dot{p}_n = v_g \cos\chi \cos\gamma \\ \dot{p}_e = v_g \sin\chi \cos\gamma \\ \dot{h} = v_g \sin\gamma \\ \dot{v_g} = \dfrac{F_{thrust} - F_{drag}}{m} - g\sin\gamma \\ \dot{\chi} = \dfrac{F_{lift}\sin\phi\cos(\chi - \psi)}{m v_g \cos\gamma} \\ \dot{\gamma} = \dfrac{F_{lift}}{m v_g}\cos\phi - \dfrac{g}{v_g}\cos\gamma \end{cases} \tag{2.4}$$

where v_g is the aircraft's velocity relative to ground and F_{thrust}, F_{drag}, F_{lift} denote thrust, drag and lift forces, respectively, in $\mathcal{F}^b$, as shown in Fig. 2.7. The mass of the UAV is represented by m, while g is the standard acceleration due to gravity [20].

The model described by equations (2.4) assumes the aircraft is controlled via thrust F_{thrust}, lift force F_{lift} and bank angle ϕ. These variables were choses as typical "inputs that a human pilot commonly controls: engine thrust, lift from the lifting surfaces, and bank angle" [20]. Using an approach similar to kinematic guidance models, the control variables are given by

$$\begin{cases} \dot{\phi} = b_\phi\left(\phi^c - \phi\right) \\ \dot{F}_{lift} = b_{F_{lift}}\left(F^c_{lift} - F_{lift}\right) \\ \dot{F}_{thrust} = b_{F_{thrust}}\left(F^c_{thrust} - F_{thrust}\right) \end{cases} \tag{2.5}$$

where b_ϕ, $b_{F_{lift}}$, $b_{F_{thrust}}$ are the autopilot's constants. The c superscript again denotes the controlled variables. The dynamic guidance model given by equations (2.4) and (2.5) can be used similarly to the kinematic guidance models in section 2.3. As it considers Bernoulli's law, the performance of the UAV depends on air density, which changes dramatically in the whole range of operational altitudes available to the HALE UAV.

Nevertheless, the dynamic guidance model also its has drawbacks. Although it considers axial wind conditions (i.e., headwind and tailwind), it lacks the influence of lateral forces due to crosswind. Moreover, a detailed model should also consider atmospheric disturbances such as wind gusts, which may be modeled using Dryden wind gust model as noted in MIL-F-8785C [21].

2.5 Dynamic vs kinematic

Dynamic guidance models apply force balance relations to point-mass models, while kinematic models utilize only kinematic relationships, without considering aerodynamics and forces directly [20].

Choosing between dynamic model and kinematic model depends heavily on the actual use case. If the computation time and resources are not an issue, the most precise, although the most complex, full non-linear dynamic model may be preferred. Therefore, the dynamic model is optimal for pre-flight path planning in a ground station. However, if a path fragment has to be validated in real-time using onboard processing unit, the simpler kinematic guidance model becomes optimal.

Nevertheless, for HALE-class UAV the flight altitude becomes a major concern. While feasible for other classes of aircrafts, kinematic models become gradually less usable with increasing altitude. Flying higher requires to consider, e.g., lower air density and thus changes to the lifting force, in accordance to Bernoulli's law. Wind influence also becomes a major factor and the inertial parameters of the UAV cannot be neglected anymore.

Therefore the optimal solution for HALE UAV's onboard computer may be the dynamic guidance model, which models the basic aerodynamics, while not completely sacrificing computation performance.

2.6 Dubins paths

The paths were introduced in 1957 by Dubins in [22]. The term *Dubins path* refers to a minimum-distance path between two vehicle *poses* (or *configurations* [23]) in a given *configuration space* [23, 24]. The vehicle is assumed to move at constant forward velocity from the *initial* pose to the *final* pose and is subject to kinematic constraints [24]. Each pose defines the position of the vehicle and its orientation.

Dubins paths are defined for both two- and three-dimensional spaces [11, 24–26]. In the thesis only three-dimensional Dubins paths will be considered. Therefore, for simplicity, any further use of the Dubins path $\mathcal{D}$ – without specifying its dimensions – will refer to a Dubins airplane path $\mathcal{D}_{3D}$.

2.6.1 Dubins car paths

Dubins himself in [22] shows that an optimal path[4] of a vehicle constrained to turning and driving forward is always an ordered set of three segments. It is either $\{CSC\}$ or $\{CCC\}$, where C is a turn with constant radius of ρ and S is a straight line. Assuming two possible turn directions the entire set of *admissible paths* expands to

$$\mathcal{D}_{2D} = \{LSL, RSR, RSL, LSR, RLR, LRL\} \tag{2.6}$$

[4] That is, the shortest path – or *geodesic* [22, 23] – in a windless 2D environment [24].

where: L is a left (counterclockwise) turn, R is a right (clockwise) turn and S is a straight line [23, 27, 28]. Further expanding the set by allowing reverse movement[5] leads ultimately to the set of 48 admissible paths [23].

In Dubins's original approach the optimal path is chosen by evaluating the entire set. Thus, as pointed out by Shkel and Lumelsky, this might be an issue for time-critical applications such as real-time path planning[6] [23].

Due to its resemblance to car-like movement the vehicle following a Dubins path is often referred to as a *Dubins car* [24, 30]. The model is widely used in algorithms for waypoint following, especially in nonholonomic UGVs [27]. For example Balluchi et al. simulate a unicycle [31]. Yong and Barth verify their real-time dynamic path planner by guiding their nonholonomic robot on Dubins paths [30]. Cowlagi and Tsiotras implicitly use Dubins paths to test their original 2D path planning algorithm [32]. Hanson et al. use the car model to move a virtual vehicle performing sequential target observation with ranged sensors [27]. Nevertheless, Dubins car model also has more abstract applications such as joining railways or planning pipe networks [23].

2.6.2 Dubins airplane paths

The model gained popularity in aerial applications as well [24]. Anderson and Milutinović use flat Dubins path for their UAV to track an unpredictably-moving (i.e., modeled with brownian motion) object [10]. Somewhat similar work by Rahmani et al. shows its usage for multi-UAV convoying of the group of UGVs [11]. However, as their UAVs flew on constant altitude the car model was sufficient.

Many applications, however, require incorporating altitude changes into the path planner. To rule out the planar restriction some modifications has been made. Chitsaz and LaValle added a climb-rate constraint to the turn-rate constraint, thus providing a *Dubins airplane* [25]. Hota and Ghose elaborate on their work by deriving an analytical solution and comparing it with purely numerical results [26].

Nonetheless, in [24] McLain et al. point Chitsaz and LaValle did not consider practical issues, which leads to implementation problems on a real UAV. As argued by McLain et al. "when the altitude component of the path falls within a specific range, there are an infinite number of paths that satisfy the minimum-distance objective" [24]. The authors expanded the basic concept by utilizing standard kinematic equations and including airspeed, flight-path angle and bank angle [24].

Moreover, they split the airplane paths into three altitude-based categories. Low-altitude paths differ only by the addition of flight-path angle γ. High-altitude paths which are not satisfied by Dubins car model require helix-based lifting or gliding maneuvers. Medium-altitude paths[7] are solved by performing an intermediate arc after the initial helix (for climbing) or before the terminal helix (for gliding) [24].

Finally, analogous to (2.6), the complete set of 28 possible movements for forward-moving UAV can be expressed by

$$\begin{aligned}\mathcal{D}_{3D} = \{&LSL, RSR, RSL, LSR, RLR, LRL, H_LLSL, H_LLSR,\\ &H_RRSL, H_RRSR, H_RRLR, H_LLRL, LSLH_L, LSRH_R,\\ &RSLH_L, RSRH_R, RLRH_R, LRLH_L, LRSL, LRSR, LRLR,\\ &RLSR, RLRL, RLSL, LSRL, RSRL, LSLR, RSLR\}\end{aligned} \tag{2.7}$$

[5] This effectively makes it a Reeds-Shepp model, as described in [29].

[6] Instead of considering the whole set, the authors provided the decision table for finding the shortest path without explicitly calculating all of the involved paths [23].

[7] That is, the paths too steep for satisfying flight-path angle constraints and too short for helix-based maneuvers – i.e., even a single-turn helix would overshot [24].

where: H_L is a left (counterclockwise) helix and H_R is a right (clockwise) helix. Note (2.7) contains either 3-segment or 4-segment paths.

2.6.3 Alternative approaches

A similar model to Dubins car was proposed by Reeds and Shepp in [29]. The Reeds-Shepp model adds the possibility to "switch gears" of the car, i.e., reverse the direction of its movement. As the model has no practical use for fixed-wing aircrafts it is not used in the thesis.

It is worth noting that Dubins problem neglects wind velocity entirely. For modeling those Zermelo approach may be preferred [28]. The problem described by Zermelo in [33] deals with a boat navigating in a body of water in the presence of wind and/or water currents. The goal is to derive the best possible control to reach the destination while minimizing time. Additionally, alternative approaches to Dubins or Reeds-Shepp paths may be employed. For example, κ-trajectory is a computationally efficient path smoothing method, which can provide a feasible path for an airplane to follow in real-time [8, 34].

Nevertheless, the thesis already addresses the wind problem in three ways. First, any small differences in wind speed and direction are assumed to be compensated by the controller of the UAV. Second, the Local Path Planner (LPP) runs repeatedly, thus automatically correcting eventual position errors caused by wind gusts to strong to be compensated by the controller. Following an exact path is not required as long as the waypoints provided by the Global Path Planner (GPP) are reached and the path is collision-free. Third, wind is already considered during waypoint optimization carried out by GPP. Thus, the set of waypoints given to LPP is already quasi-optimal with respect to the wind. However, further improving the APP algorithm by using Zermelo approach instead of Dubins will be addressed in the further research.

2.7 Summary

The chapter started with a short introduction to the basics of modeling a 3D path of an aircraft by introducing coordination frames and wind triangle. Next sections briefly described kinematic and dynamic guidance models of a generic fixed-wing UAV. This was followed by an introduction to Dubins paths and vehicle models in both planar and three-dimensional spaces. Beside basic concept, the terms of Dubins car and airplane were introduced and supplemented with examples. Finally, some considerations of other modeling approaches were given.

3. Environment modeling

This chapter describes the concept of an environment map. First, the general idea of the map is given. Then, every map layer is discussed in detail. The feasible and dynamic model of the environment is crucial for adaptability of APP.

3.1 Environment map

Apart from the UAV-specific parameters, the state space used by planning algorithms must include a well-defined scene as well. This includes any possible obstacles, such as terrain, areas restricted by law or dangerous weather conditions that should be avoided by the UAV. Moreover, to plan an energy-efficient path, data related to air currents in the flight area is necessary. Both issues will be addressed by building environment models (maps) using available topographical and meteorological data, as well as airspace segmentation data specified by air navigation services agencies, i.e., the Polish Air Navigation Services Agency (PANSA) in Poland.

The complete environment map $\mathcal{M}$ consists of 4 distinct layers:

- terrain map $\mathcal{M}_t$ – models the elevation of terrain and is used for collision avoidance;
- airspace map $\mathcal{M}_a$ – contains 3D regions of interest (ROIs) with values assigned;
- wind map $\mathcal{M}_w$ – which stores wind velocity forecast as a dynamic vector field;
- measurement maps $\mathcal{M}_m$ – optional and mission-specific map layers, that models, e.g., estimated pollutant distribution.

A concept of an automated map building methodology is presented in Fig. 3.1. The measurement maps vary the most between the missions, so they are built manually by a human expert. Hence they are not shown in the figure.

Building an environment map requires to specify its boundary coordinates, i.e., latitude, longitude and altitude. The map is a cuboid, which stores regularly sampled data. Thus, it requires two boundary values to be specified for each dimensions. Being dynamic, the wind map requires time as the fourth dimension as well.

The terrain map $\mathcal{M}_t$ is build from elevation data acquired, e.g., from a web server or by interpreting SRTM data [35]. First, horizontal coordinates (i.e., latitude and longitude) formulate a 2D grid. Then, elevation data is acquired for each point on the grid. Finally, the map model computes the elevation interpolant, which provides continuous elevation values for any pair of horizontal coordinates.

The airspace map $\mathcal{M}_a$ is build in parallel. $\mathcal{M}_a$ consists of a set of three-dimensional shapes representing airspace classes and possible obstacles. The map is built based mostly on the official airspace data provided by the local aviation authorities. Areas are filtered based on mission parameters and the class of UAV. The chosen areas are then converted to 3D shapes and each one gets a value assigned to it.

The wind map $\mathcal{M}_w$ is built by acquiring wind velocity data computed by a weather model. It is done by querying the time series for the set of specified geographical coordinates from a weather server. Alternatively, an offline pre-computed weather data may be used. To be computationally efficient, the wind map requires uniformly sampled gridded data, similarly to the terrain map. This leads to an issue, as the third dimension, altitude, might be available only as above ground

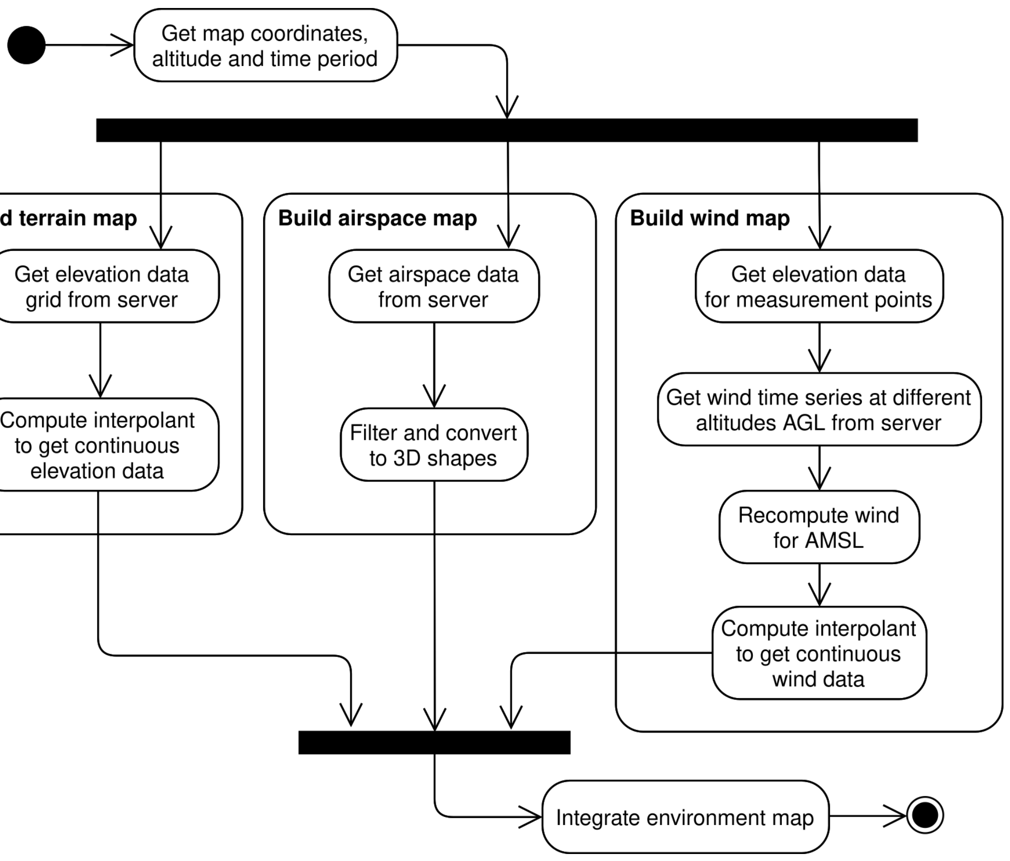
Get map coordinates, altitude and time period
Build terrain map
Get elevation data grid from server
Compute interpolant to get continuous elevation data
Build airspace map
Get airspace data from server
Filter and convert to 3D shapes
Build wind map
Get elevation data for measurement points
Get wind time series at different altitudes AGL from server
Recompute wind for AMSL
Compute interpolant to get continuous wind data
Integrate environment map

Fig. 3.1: Building environment maps

level (AGL). To be compatible with $\mathcal{M}_t$, the wind map altitudes have to be defined as absolute, that is, above mean sea level (AMSL).

To convert a wind measurement time series, the map builder also queries the elevation of the ground for a given coordinates, where the measurements are taken. Now, the builder acquires wind data for different altitude levels. Wind data is then extrapolated and extended into ground. While physically irrational, it is used to assure the map is uniformly gridded, so continuous wind measurements are possible. Uniformly placed data points are then used to approximate wind measurements continuously at any given point of the space-time by employing 4D interpolation[1]. The process is repeated for all the measurement points.

The maps are then integrated into a complete environment map $\mathcal{M}$ required by the planner described latter in the thesis. At this point, it is also possible to manually define additional obstacles to be included in the final obstacle map or define additional layers of the map, for example, a measurement layer related to pollutant distribution.

Alternatively, 3D space can be divided into a mesh, thus forming a three-dimensional network diagram, which connects the start and goal poses. Such mesh-based environment representation was used, e.g., by Duan et al. [8].

[1] The conversion and generation of artificial data is done to fulfill the continuity requirements of the gridded interpolation, which is computationally faster than interpolation of scattered data, see `https://www.mathworks.com/help/matlab/ref/griddata.html`.

3.2 Terrain map

To plan collision-free path, the knowledge of nearby obstacles is necessary. A terrain map $\mathcal{M}_t$ stores elevation data, thus defining terrain-based features treated as obstacles, e.g., mountains.

For the majority of time $\mathcal{M}_t$ is not required due to the mostly high-altitude flights of the UAV. Nevertheless, it becomes important for the crucial maneuvers such as gaining height after takeoff, preparing to land or conducting low-altitude measurements, such as smog profiling [14], especially in mountainous areas.

3.2.1 Voxel-based terrain map

A terrain map can be modeled as a voxel map. To reduce memory usage, it can be implemented using efficient data structures, e.g., binary trees or octrees. In this research, an OctoMap model presented by Hornung et al. in [36] was used (Fig. 3.2).

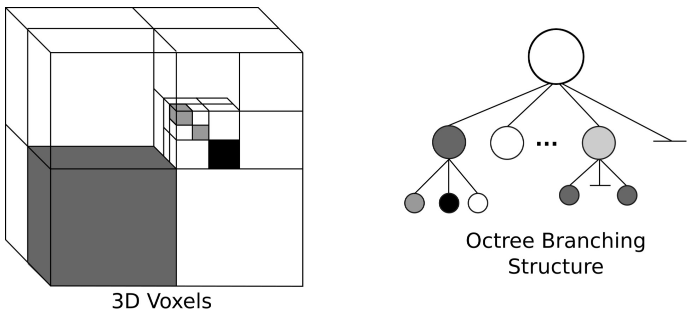

Fig. 3.2: Octree concept [37]

An octree is a hierarchical tree-like branching structure, which subdivides volume into voxels. The structure is recursive in that sense, each voxel is subdivided into another eight voxels until the map achieves desired resolution or voxel size. Thus, a large space without obstacles or large obstacle can fit into a single voxel, saving memory.

Contrary to the original probabilistic approach described in [36], only binary occupancy is considered in the thesis, as discussed above. Therefore, the map divides the scene into 2 binary states: free or occupied. The considered map is static and completely known *a priori*, so probabilistic approach serves no purpose and storing binary values instead of numerical leads to performance improvements.

Voxel-based terrain map can be based on elevation data as well, which is converted to an occupancy map. It is assumed the volume above the terrain is obstacle-free, while below ground is always occupied. The voxel-based binary terrain map is given by

$$\mathbf{M}_t^V = \{m_{tijk}^V\} : m_{tijk}^V \in \{0, 1\}, \qquad \begin{aligned} i &= 1, 2, ..., N_x, \\ j &= 1, 2, ..., N_y, \\ k &= 1, 2, ..., N_z \end{aligned} \tag{3.1}$$

where N_x, N_y and N_z are the number of cells along the map's x, y and z axes, respectively. A binary value m_{tijk}^V indicates that the cell at i-th, j-th and k-th index of respective axes x, y and z is either free (if 0) or occupied (if 1).

Limitations

Voxel-based terrain map $\mathbf{M}_t^V$ was used in the preliminary study, but later was superseded by a so-called 2.5D model for performance reasons. The 2.5D model is described in the next section. A detailed explanation is given in the comparative test described in Chapter 5.

3.2.2 Discrete terrain map

The discrete 2.5D terrain map is built by acquiring a finite set of gridded elevation data, for example from a web server. The measurement points are uniformly placed inside a bounding box defined by latitude and longitude boundaries to form a grid. The map is sampled using a constant step, which is equal in each direction. Sampling density can be adjusted to provide more or less reliable geometry, which influences the map's fidelity and computation time. Formally, a discrete terrain map can be expressed as a matrix $\mathbf{M}_t^D$ of elevation data points m_{tij}^D, such that

$$\mathbf{M}_t^D = \{z_{tij}\} : z_{tij} \in \langle z_{min}, z_{max} \rangle \,, \qquad \begin{aligned} i &= 1, 2, ..., N_x, \\ j &= 1, 2, ..., N_y, \end{aligned} \tag{3.2}$$

where N_x and N_y are the number of cells along the map's x and y axes, respectively. Each value z_{tij} holds an elevation of terrain above mean sea level (AMSL), limited by minimal terrain elevation z_{min} and maximal terrain elevation z_{max}.

Limitations

As 2.5D implies, the third dimension, altitude, is simplified. A 2.5D model is a two-dimensional static elevation map, which assumes altitude is modeled by a single threshold value. Values equal or below the threshold are considered colliding with ground. Consequently, this map cannot model tunnel-like features, i.e., the possibility to fly below ground. That features, however, are not considered as valid to fly in the thesis, thus this representation is preferred for better performance over the voxel-based map.

3.2.3 Continuous terrain map

Terrain's fidelity can be improved by implementing a continuous terrain map $\mathbf{M}_t^C$. The continuous map interpolates data from the nearby queried data points to provide elevation for continuous coordinates. $\mathbf{M}_t^C$ is formally defined as

$$\mathbf{M}_t^C = f\left(x, y, \mathbf{M}_t^D\right) \tag{3.3}$$

where $f(\cdot)$ is a generic interpolation function along horizontal dimensions x and y representing longitude and latitude, respectively. The function $f(\cdot)$ interpolates elevation value based on the points inside a discrete map $\mathbf{M}_t^D$ closest to the points specified by x and y. Exact form of $f(\cdot)$ depends on the used interpolation method.

By implementing interpolation methods of different complexity, the balance of smoothness vs computation performance of the final terrain model can be adjusted. In the simplest form, $\mathbf{M}_t^C$ uses nearest neighbors interpolation to provide discretized elevation. If not specified otherwise, the terrain maps in the thesis use bilinear interpolation, which provides C0-continuity and requires relatively low computation effort.

Fig. 3.3 illustrates an example[2] of C0-continuous terrain maps for two different sampling densities. The top figures (a and b) represent the coordinates of sampling points, i.e., the reference points with data queried from external source. Horizontal coordinates are given in meters relative to the origin of the map, where axes x and y are aligned with east and north, respectively. Crosses mark the local origin of each map, i.e., the point $(0, 0)$. The bottom figures (c and d) present resulting bilinearly interpolated terrain maps. Note smoothing out of the peaks shown in red boxes caused by reduced sampling density.

[2] The chosen area is Mt. Everest and its surrounding. Local origin $(0, 0)$ is set to the peak of Mt. Everest. This area was chosen as an example for its prominent terrain features.

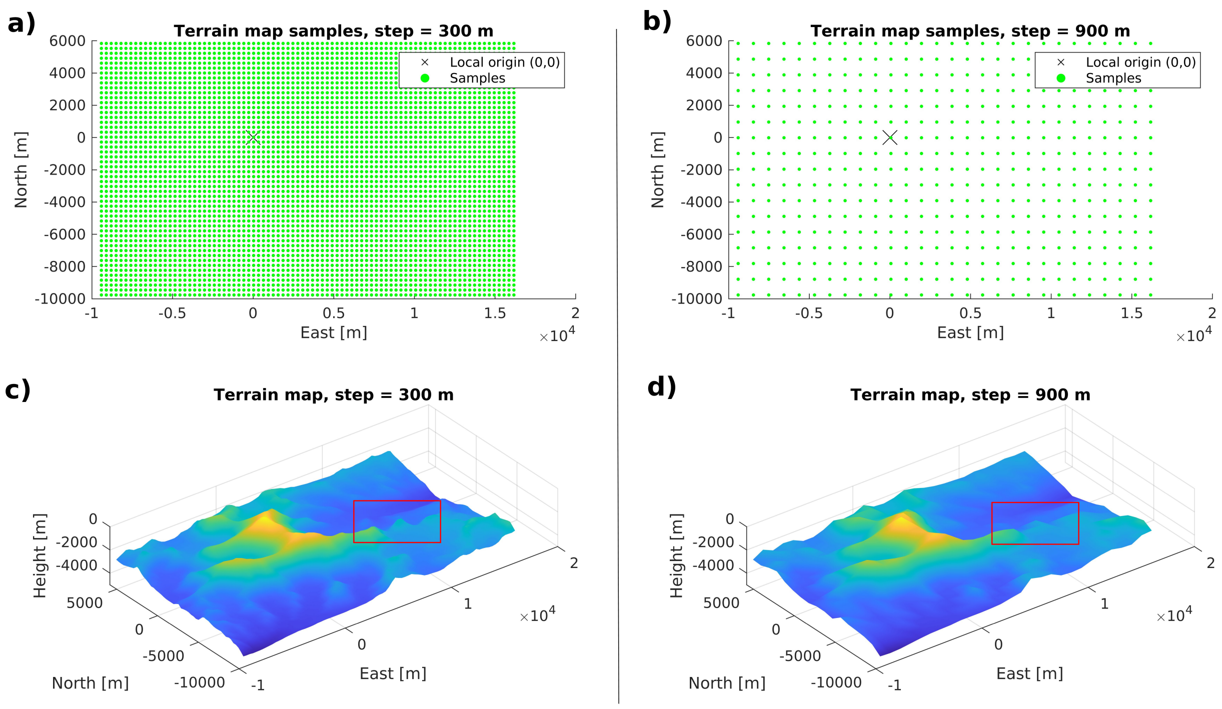

Fig. 3.3: Sample terrain maps

Limitations

The limitations of a discrete terrain map $\mathbf{M}_t^D$ applies also to its continuous counterpart.

3.2.4 Collision checking

To check for possible collisions with terrain, the $\texttt{free}\left(\cdot\right)$ operator is defined. It checks whether the point at given x, y and z coordinates is placed above the ground elevation at this point (classified as free), or not (classified as colliding with terrain). Assume $\mathbf{w}$ denotes a waypoint

$$\mathbf{w} = \begin{bmatrix} w_x & w_y & w_z \end{bmatrix} \tag{3.4}$$

where w_x, w_y and w_z are its coordinates along axes x, y and z, respectively. Now, in the most general form of the $\texttt{free}\left(\cdot\right)$ operator is given by

$$\texttt{free}\left(\mathbf{w}, \mathcal{M}_t\right) \tag{3.5}$$

where $\mathcal{M}_t$ denotes any kind of a terrain map as defined above. Depending on the flavor of terrain map, the exact implementation of $\texttt{free}\left(\cdot\right)$ differs slightly.

Voxel-based terrain map

The $\texttt{free}\left(\cdot\right)$ operator takes the simplest form in the case of the voxel-based terrain map $\mathbf{M}_t^V$. In that case, it is defined as

$$\texttt{free}\left(\mathbf{w}, \mathbf{M}_t^V\right) = 1 - m_{tijk}^V, \qquad \begin{aligned} i &= w_x \\ j &= w_y \\ k &= w_z \end{aligned} \tag{3.6}$$

where $\mathbf{w}$ is the queried waypoint, as in eq. (3.4). $\mathbf{M}_t^V$ and m_{tijk}^V are defined as in eq. (3.1), but i, j and k indices are given by $\mathbf{w}$. The operator returns either 1 if $\mathbf{w}$ is located inside a free voxel or 0 if that voxel is occupied.

Discrete terrain map

For a discrete terrain map $\mathbf{M}_t^D$, the collision check is given by

$$\texttt{free}\left(\mathbf{w}, \mathbf{M}_t^D\right) = \begin{cases} 1 \text{ if } w_z > z_{tij} \\ 0 \text{ otherwise} \end{cases}, \qquad \begin{aligned} i &= w_x \\ j &= w_y \end{aligned} \tag{3.7}$$

where $\mathbf{w}$ is the queried waypoint, as in eq. (3.4). $\mathbf{M}_t^D$ and z_{tij} are defined as in eq. (3.2), but i and j indices are given by $\mathbf{w}$.

Continuous terrain map

Collision checking for a continuous terrain map $\mathbf{M}_t^C$ is defined similarly as for $\mathbf{M}_t^D$, but also considers interpolation. The $\texttt{free}\left(\cdot\right)$ operator for $\mathbf{M}_t^C$ takes the form

$$\texttt{free}\left(\mathbf{w}, \mathbf{M}_t^C\right) = \begin{cases} 1 \text{ if } w_z > \mathbf{M}_t^C(w_x, w_y, \mathbf{M}_t^D) \\ 0 \text{ otherwise} \end{cases} \tag{3.8}$$

where $\mathbf{w}$ is the queried waypoint, as in eq. (3.4). $\mathbf{M}_t^C$ is defined as in eq. (3.3).

3.3 Wind map

Estimating wind conditions is crucial for energy-optimal flight, especially for HALE UAVs. High-altitude flights mean significant changes in height, so wind data must be available at different altitudes. Moreover, long endurance flights require a model valid for several hours after takeoff. Hence, the model must be dynamic. For example, it can be a function of coordinates, altitude and forecast time. This approach requires, however, calculating wind speed and direction each iteration [38]. Weather models such as the National Centers for Environmental Prediction Global Forecast System (GFS) or the European Centre for Medium-Range Weather Forecasts (ECMWF) [39] tend to be complex and require significant computation effort to be computed.

To improve computation performance, the thesis proposes a different method. Instead of computing wind estimates on-the-fly, the wind forecast map provides a simplified model – similar to the one used for terrain maps and discussed in detail in [38].

3.3.1 Discrete wind map

Precomputed wind velocity data is acquired from external source, e.g., a web server, and stored inside a wind map $\mathcal{M}_w$. It contains the horizontal component of the wind velocity vector, which is also sampled in time and altitude. The map $\mathcal{M}_w$ is dynamic in such way, that it contains wind forecasts for a given period of time. Thus, wind map $\mathcal{M}_w$ is effectively a dynamic 3D vector field defined in 4D space-time. A similar approach using a 2D wind forecast map was used by Andrade [40]. The idea of discrete wind map is presented in Fig. 3.4.

An example of the wind map for Poland is shown in Fig. 3.5. Wind at sample points is displayed as vectors. Wind blows from the tail of a vector to its head. Wind speed is reflected by the length of a vector, as well as by its color[3]. Thin vertical lines indicate measurement points located at the same horizontal coordinates (i.e., latitude and longitude), but at different altitude levels. The cross marks the origin of the map, i.e., $(0, 0, 0)$.

[3] Wind maps become difficult to understand if overlaid on top of terrain maps. Hence, information redundancy was introduced. Color is useful to compare and group differently aligned vectors, while length difference helps in quantitative comparison of wind velocity. Moreover, the colorbar allows reading the wind speed more precisely

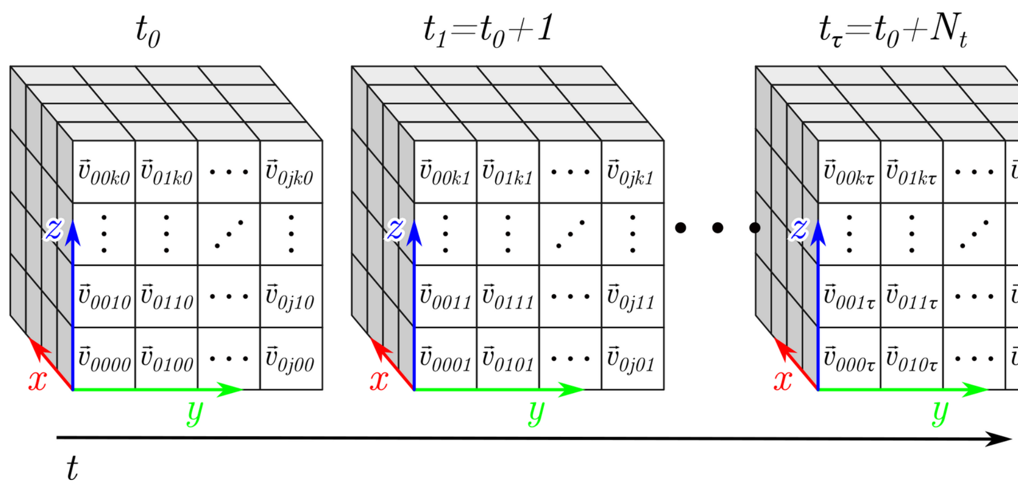
t_0
$t_1 = t_0 + 1$
$t_\tau = t_0 + N_t$
$\vec{v}_{00k0}$ $\vec{v}_{01k0}$ $\vec{v}_{0jk0}$
$\vec{v}_{0010}$ $\vec{v}_{0110}$ $\vec{v}_{0j10}$
$\vec{v}_{0000}$ $\vec{v}_{0100}$ $\vec{v}_{0j00}$
$\vec{v}_{00k1}$ $\vec{v}_{01k1}$ $\vec{v}_{0jk1}$
$\vec{v}_{0011}$ $\vec{v}_{0111}$ $\vec{v}_{0j11}$
$\vec{v}_{0001}$ $\vec{v}_{0101}$ $\vec{v}_{0j01}$
$\vec{v}_{00k\tau}$ $\vec{v}_{01k\tau}$ $\vec{v}_{0jk\tau}$
$\vec{v}_{001\tau}$ $\vec{v}_{011\tau}$ $\vec{v}_{0j1\tau}$
$\vec{v}_{000\tau}$ $\vec{v}_{010\tau}$ $\vec{v}_{0j0\tau}$
x y z
t

Fig. 3.4: Structure of data inside a discrete wind forecast map [38]

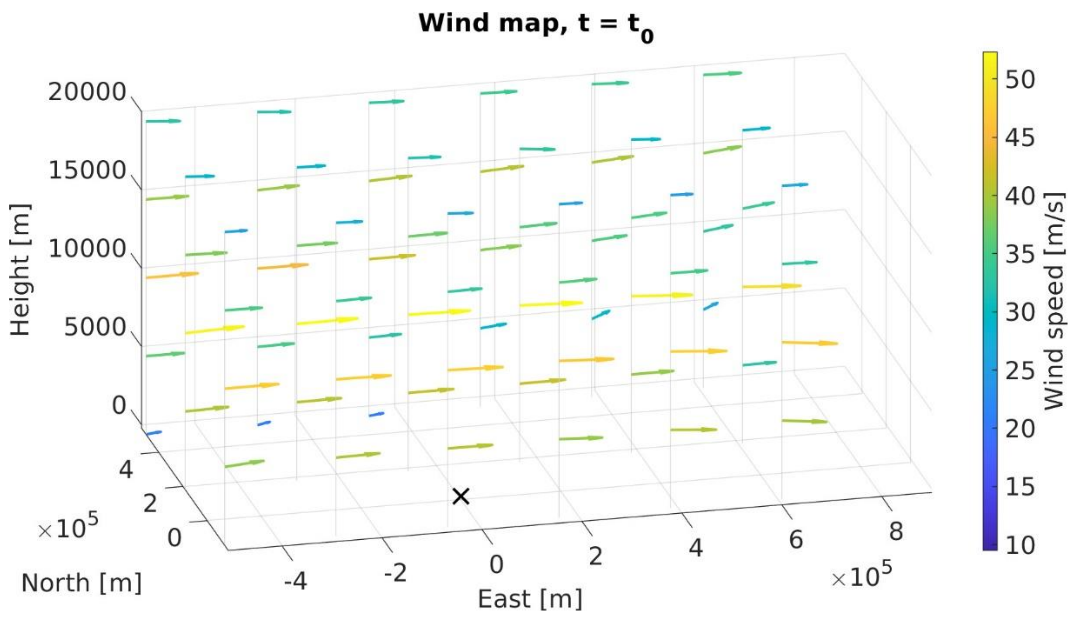

Fig. 3.5: Visualization of a sample wind forecast map for Poland [38]

The map can be imagined as four-dimensional space-time with three spatial dimensions x, y and z, and one temporal dimension t. Formally, a discrete wind map $\mathbf{M}_w^D$ is expressed by equation

$$\mathbf{M}_w^D = \{\vec{v}_{ijk\tau}\} = \Big\{\big(\|\vec{v}_{ijk\tau}\|, \angle\vec{v}_{ijk\tau}\big)\Big\}, \qquad \begin{aligned} i &= 1, 2, ..., N_x, \\ j &= 1, 2, ..., N_y, \\ k &= 1, 2, ..., N_z, \\ \tau &= 1, 2, ..., N_t \end{aligned}$$

where N_x, N_y, N_z, N_t are the number of cells along the map's x, y, z and τ axes, respectively. Axis t denotes discretized time since the earliest forecast stored in $\mathcal{M}_w^D$. Vector $\vec{v}_{ijk\tau}$ represents the wind velocity vector at i-th sample along x axis, at j-th sample along y axis, at k-th sample along z axis and at τ-th sample along t axis [38]. Alternatively, wind vectors can be replaced with pairs of wind speed $\|\vec{v}_{ijk\tau}\|$ and wind direction $\angle\vec{v}_{ijk\tau}$.

3.3.2 Continuous wind map

Wind maps can also be modeled as continuous, if necessary. A limited number of samples stored in $\mathbf{M}_w^D$ can be transformed into continuous 4D space-time using various interpolation method, similarly to terrain maps. A general form of continuous wind map $\mathbf{M}_w^C$ is described by

$$\mathbf{M}_w^C = f_i(x, y, z, t, \mathbf{M}_w^D) \tag{3.9}$$

where $f_i(\cdot)$ is a generic interpolation function along all 4 dimensions and x, y, z and τ represent the coordinates of the queried point. This function bases on the samples available at discrete points stored internally as a discrete map $\mathbf{M}_w^D$ of gridded wind velocity vectors. On demand, $\mathbf{M}_w^C$ performs the interpolation of gridded samples neighboring a given 4D point to provide an approximated but continuous result. In its simplest form, $\mathbf{M}_w^C$ uses linear interpolation to provide C0-continuity in each dimension. However, more intricate interpolation methods can also be employed to generate C1- or C2-continuous results at the cost of increased computation effort [38]. The concept of continuous linearly interpolated wind map is tested in Chapter 5 and also in [38].

3.3.3 Checking wind velocity

For convenience, the $\texttt{wind}(\cdot)$ operator is defined. It returns the wind velocity vector $\vec{v}$ at position of the waypoint $\mathbf{w}$ given as its argument. Formally $\texttt{wind}(\cdot)$ is represented by

$$\texttt{wind}(\mathbf{w}, t, \mathcal{M}_w) = \vec{v} \tag{3.10}$$

where $\vec{v}$ is the wind velocity vector computed for the position of the waypoint $\mathbf{w}$ and time t by the wind map $\mathcal{M}_w$. Variable $\mathcal{M}_w$ denotes a generalized wind map, which can be either $\mathbf{M}_w^D$ or $\mathbf{M}_w^C$.

3.3.4 Limitations and possible improvements

It should be noted that by the time of writing the thesis the publicly available weather maps do not include densely-sampled vertical wind data. Hence, advanced flying strategies, such as soaring, i.e., flying in the masses of warm air [41], are not implemented yet.

While a wind map can also be implemented as an octree (see section 3.2.1), it is assumed measurement points are sparse to reduce the amount of server queries. Hence, performance gain and memory reduction would be negligible, as most values will differ.

To assure stability in simulation, the wind map model described in this research interpolates values between the measurement points to achieve continuous output. Due to performance reasons, the map uses linear interpolation, which grants C0-continuity. If computation time is not crucial, other methods, such as cubic or spline interpolation, can be employed.

3.4 Airspace map

The airspace map $\mathcal{M}_a$ describes the structure of the scene's airspace. The main reason of $\mathcal{M}_a$ is to model airspace limitations enforced by law, which are in fact artificial obstacles in airspace. An example segregation of the airspace over Łódź Lublinek airport (Poland) is presented in Fig. 3.6. This airspace segregation map is the PANSA's official AUP/UUP map for Poland [42]. Note that not all of the regions shown are considered impassable to UAVs.

The Regions of Interest (ROIs) on the airspace map are described by three-dimensional prisms. A prism is defined by a set of pairs of latitude and longitude, which form a horizontal polygon. The polygon is then extruded vertically between two altitude levels to form a prism. Each prism

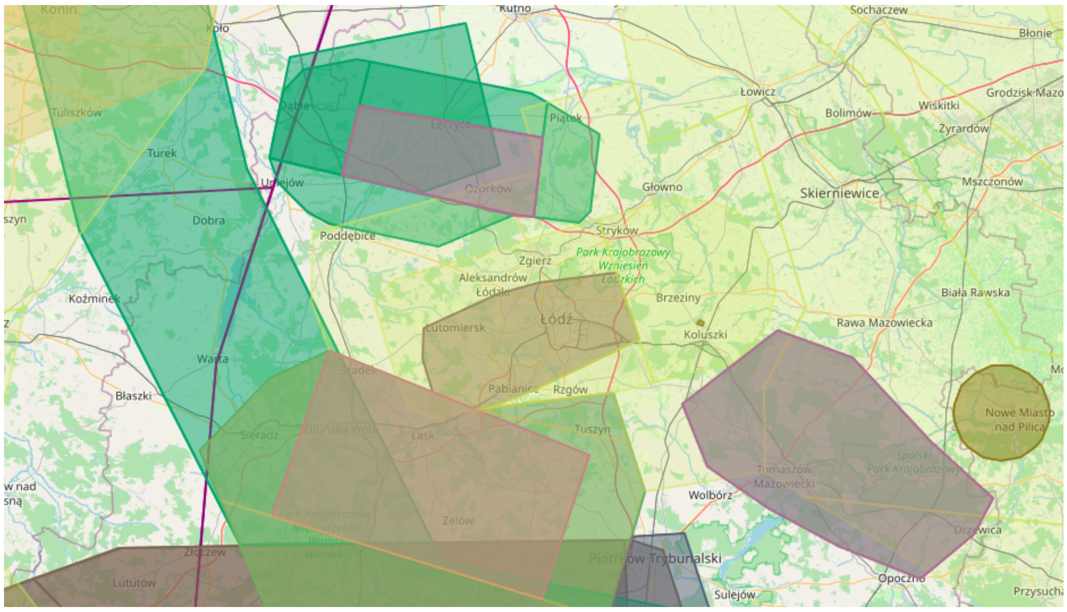

Fig. 3.6: Segregated airspace over Łódź Lublinek airport (ICAO:EPLL), Poland [42]

can be then classified according to the local law, which imposes various constraints on the aerial vehicles flying inside it.

In the thesis, the airspace map is used solely for modeling forbidden airspace regions, though. ROIs inside the map are classified as obstacles. It means, crossing them is forbidden and invalidates the planned path (i.e., the path becomes unfeasible). This covers law-enforced no-fly zones (NFZs) but also extends to ROIs with dangerous weather conditions, such as storm clouds.

3.4.1 Prism-based airspace segmentation

Let $\triangle^{2D}$ be a polygon specified by a set of N horizontal coordinates, that is

$$\triangle^{2D} = \{(x_i, y_i)\}, \qquad i = 3, 4, ..., N \tag{3.11}$$

where N is the number of pairs (at least 3). Variables x_i and y_i denote horizontal coordinates (e.g., longitude and latitude or their local Cartesian equivalents) of i-th vertex of the polygon. To form a prism, The polygon $\triangle^{2D}$ is extended between two altitude levels to form a prism

$$\triangle^{3D} = \{\triangle^{2D}, z_{min}, z_{max}\}, \qquad \triangle^{3D} \in \mathcal{M}_a \tag{3.12}$$

where z_{min} and z_{max} denote the altitudes of the bottom and the top bases of the prism, respectively. Δ^{2D} denotes the two-dimensional polygon, which represents the shape of the bases. An abstract example of an airspace map is shown in Fig. 3.7. ROIs are modeled by colored semi-transparent prisms. Different colors are used solely for presentation purposes.

3.4.2 Collision checking

For convenience, the `free` $(\cdot)$ operator is defined also for $\mathcal{M}_a$, similarly as for terrain maps $\mathcal{M}_t$. The operator returns 1 if a waypoint $\mathbf{w}$ does not lie inside any of the prisms in $\mathcal{M}_a$, or formally

$$\texttt{free}\left(\mathbf{w}, \mathcal{M}_a\right) = \begin{cases} 1 \text{ if } \forall \triangle^{3D} \in \mathcal{M}_a | \mathbf{w} \cap \triangle^{3D} = \{\emptyset\} \\ 0 \text{ otherwise} \end{cases} \tag{3.13}$$

where $\mathbf{w}$ is the queried waypoint.

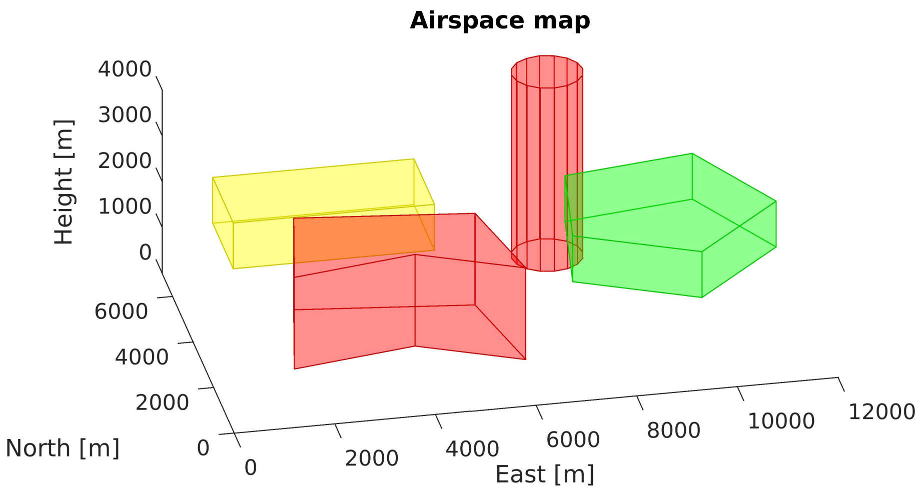

Fig. 3.7: An example abstract airspace map

3.4.3 Limitations

Only static vertically-aligned prismatic shapes may be modeled by the airspace map defined as above. As modeling exact shapes of obstacles is outside of scope of the thesis, this representation is considered sufficient for simplified shapes, e.g., storm clouds or law-enforced NFZs. Nevertheless, if the idea of the airspace map is adapted for a UAV requiring more precise maneuvers, this representation should be revised to include more complex shapes.

3.5 Measurement maps

Measurement maps $\mathcal{M}_m$ are the final but optional layers of the environment map. They are used to model the measurement strategy. For example, a measurement map $\mathcal{M}_m$ may model the estimated pollutant distribution to optimize the location and shape of the measurement path. Then, this measurement layer can be updated during the mission, e.g., with actual measurement data. This way it is possible for the path to adapt to the actual distribution of the pollutant and further optimize the measurement path. Measurement maps are used solely by GPP.

Each $\mathcal{M}_m$ is defined similarly to a wind map $\mathcal{M}_w$. A discrete static three-dimensional measurement map $\mathbf{M}_m^D$ is defined as

$$\mathbf{M}_m^D = \{\mathbf{m}_{mijk}\}, \qquad \begin{aligned} i &= 1, 2, ..., N_x, \\ j &= 1, 2, ..., N_y, \\ k &= 1, 2, ..., N_z \end{aligned}$$

where N_x, N_y, N_z are the number of cells along the map's x, y, z. Vector $\mathbf{m}_{mijk}$ denotes the measurements (either actual or their estimates) acquired at i-th, j-th and k-th cell along x, y and z axes, respectively. Analogous to wind maps, measurement maps can be transformed to continuous by the means of interpolation. A continuous static 3D measurement map takes the form

$$\mathbf{M}_m^C = f_i(x, y, z, \mathbf{M}_m^D)$$

where $f_i(\cdot)$ is a generic interpolation function along the 3 spatial dimensions. This function bases on the samples available at discrete points stored internally as a discrete measurement map $\mathbf{M}_m^D$ defined as above. Alternatively, $\mathbf{M}_m^C$ can be given as an explicit function $f_a(\cdot)$, which approximates the pollution intensity distributed in space, that is

$$\mathbf{M}_m^C = f_a(x, y, z). \tag{3.14}$$

Unless otherwise specified, eq. (3.14) is used as the default pollution model throughout the thesis.

3.5.1 Checking measurement value

To check the measurement value (i.e, pollution intensity) at any given position the $\texttt{pollution}(\cdot)$ operator is defined. The operator returns the value stored inside $\mathcal{M}_m$, which corresponds to the value stored at the position of the waypoint $\mathbf{w}$. The formal definition of the $\texttt{pollution}(\cdot)$ operator is

$$\texttt{pollution}(\mathbf{w}, \mathcal{M}_m) = m \tag{3.15}$$

where m is the quantitative measurement at the position of the waypoint $\mathbf{w}$ returned by the measurement map $\mathcal{M}_m$. Again, variable $\mathcal{M}_m$ denotes a generalized measurement map, i.e., $\mathbf{M}_m^D$ or $\mathbf{M}_m^C$.

3.5.2 Limitations

The measurement maps $\mathbf{M}_m^D$ and $\mathbf{M}_m^C$ are static. It means, in their current form they cannot represent, e.g., pollutant distribution changing over time. The maps, however, can be made dynamic by adding another axis, similarly to the wind maps $\mathbf{M}_w^D$ and $\mathbf{M}_w^C$.

In practice, at least for the first flight, the exact shape and location of the polluted volume are not known. Therefore, an explicit model (3.14) is preferred to guide the UAV to the point of the highest estimated pollutant concentration. For the next flights, the sample-based models may be used to better reflect the exact shape of the polluted region based on the data collected during the first flight.

3.6 Summary

This chapter described the context of an environment map. The chapter begun with a general idea of the environment model, its components and building methodology. Next, each of the 4 layers of the map was described. The environment map as described in this chapter will be used as the default model of the scene throughout the thesis.

4. Adaptive Path Planner

In this chapter the model-based adaptive path planning algorithm for a fixed-wing HALE UAV is proposed. First, a general form of the algorithm is given. Then, its two major components for global and local planning are discussed in detail. The chapter ends with the summary on the major features and limitations of the algorithm.

4.1 Path planning

Path planning means building a path used to travel from the start position in space to the goal position while avoiding obstacles and minimizing a positive cost metric (e.g., length of traverse) [43]. The path is built with the minimal cost according to the criteria and bound by the constraints [5]. Based on a review by Zhao et al. [5], UAV path planning in general can be characterized by:

- *Stealth* – staying undetectable is important especially for military applications as it grants safety; for civil application it means avoiding inhabited areas.
- *Physical feasibility* – concerns all classes of UAVs and refers to the physical limitations of the aircraft, e.g., the maximum path distance (flight range) and the minimum length of a single segment of the path (maneuverability).
- *Performance of the mission* – whether the path satisfies the mission-specific requirements, for example, the airspeed limits of the measuring equipment, but also the compatibility with a specific measurement procedure.
- *Real-time implementation* – refers to the efficiency of the planner, i.e., its capability to quickly recompute the path, which is crucial for a dynamic environment.

Duan et al. in [8] consider three most important requirements of an ideal path planner:

- *Optimality* – whether the planner provides the globally optimal solution, or how close the actual solution is to the global one.
- *Completeness* – to be complete, the algorithm has to find a solution if one exists or report failure otherwise.
- *Computational complexity* – especially important for fast UAVs, such as combat UAVs, which are the main focus of the authors of [8].

4.2 Adaptive planning vs non-adaptive planning

Non-adaptive planning refers to the problem in which the planner searches for feasible paths using information about the whole environment. It is assumed the environment is static and fully determined with some specific constraints applied. That is, the entire scenario, i.e., the states of the UAV, as well as its environment are known *a priori* and predictable. This includes planning the mission before the UAV actually takes off. Zhao et al. mention here several classes of constraints, such as obstacle and threat constraints, velocity and acceleration constraints, and minimum path and fuel consumption constraints [5]. In non-adaptive planning, finding an optimal or quasi-optimal solution is more important than minimizing the computation effort.

Conversely, *adaptive planning* occurs when the states of the UAV or its environment change indefinitely over time. In this case, the planner reacts to random and unpredictable changes in the state space, e.g., unexpected wind gusts causing position errors [5]. It is crucial here to provide a solution in very limited time, i.e., satisfy the computational complexity requirement [8]. Therefore, employing a purely deliberative planning strategy is not reasonable. Adaptive planning favors finding a feasible solution, that is ”good enough”, but not necessarily optimal, within reasonable time. Authors of [5] classify the adaptive[1] path planning as a dynamic multi-objective optimization problem.

The classification above bases on the definitions of *offline* planning and *online* planning from [5]. However, the terms *non-adaptive* and *adaptive*, respectively, are used instead. The reason is the term *offline* limits the planner to be run only before the mission actually begins. As it is desirable to recalculate the path on demand, the term *non-adaptive* seems more adequate. Other parts of the definition still apply. In the context of the thesis, the terms *adaptive* and *online* are interchangeable, but the first is preferred as it is the antonym of *non-adaptive*. In the course of the thesis, adaptive planning means the ability to re-plan the path to counteract events that were not been planned before the mission.

4.3 General form of the algorithm

The Adaptive Path Planner (APP) consists of two major components: Global Path Planner (GPP) and Local Path Planner (LPP). GPP provides a close-to-optimal solution according to various criteria defined in a mission scenario. The output of GPP is the set of sparsely placed *global* waypoints, which approximate the path, and the initial path made of densely-placed *local* waypoints. LPP is a lower-level planner, which re-plans the collision-free path between the global waypoints provided by GPP. LPP outputs only local waypoints. The lowest-level planner is the flight controller of the UAV, which grants some reactivity to dynamic obstacles. It is not covered by APP, however. Alvear et al. implemented a similar approach for their chemotaxis-driven multirotor guidance system. The guidance system was deployed on a higher-level computer (Raspberry Pi), while the UAV controller ran on another board (Pixhawk) [44]. While sufficient for a multirotor, their chemotaxis-driven approach is not convenient for a more constrained fixed-wing UAV. A typical APP-based mission planning and execution methodology is shown in Fig. 4.1.
First, human personnel in the Ground Control Station (GCS) plans the mission according to the current and forecast weather conditions and (optionally) estimated pollution concentration. At this point the flight area and the measurement strategy are chosen. Also, decisions concerning UAV payload are made.

Next, the environment map is build using the parameters supplied by human experts and data queried from external sources, e.g., weather forecast servers or pre-computed weather models. The map models at least static terrain height map, the airspace regions and forecast dynamic wind conditions. It also includes measurement map if estimated pollution concentration was specified. When the map is ready, it is supplied to APP.

Then, GPP plans the global path and initial local path by optimizing the placement of global waypoints and mission parameters. GPP considers many optimization criteria including wind velocity, preferred flight areas, estimated energy expenditure and measurement-specific requirements. The global waypoints supplied by GPP only approximate the final path, however. That is, they only specify the positions, which must be reached by the UAV flying along an adaptive local path. The path is then validated in simulation using Model-in-the-Loop (MIL) and/or Software-in-the-Loop (SIL) methods. If successful, the path must be approved by a human.

[1] They use the term *online* instead of *adaptive*, but in the context of the thesis it means the same [5].

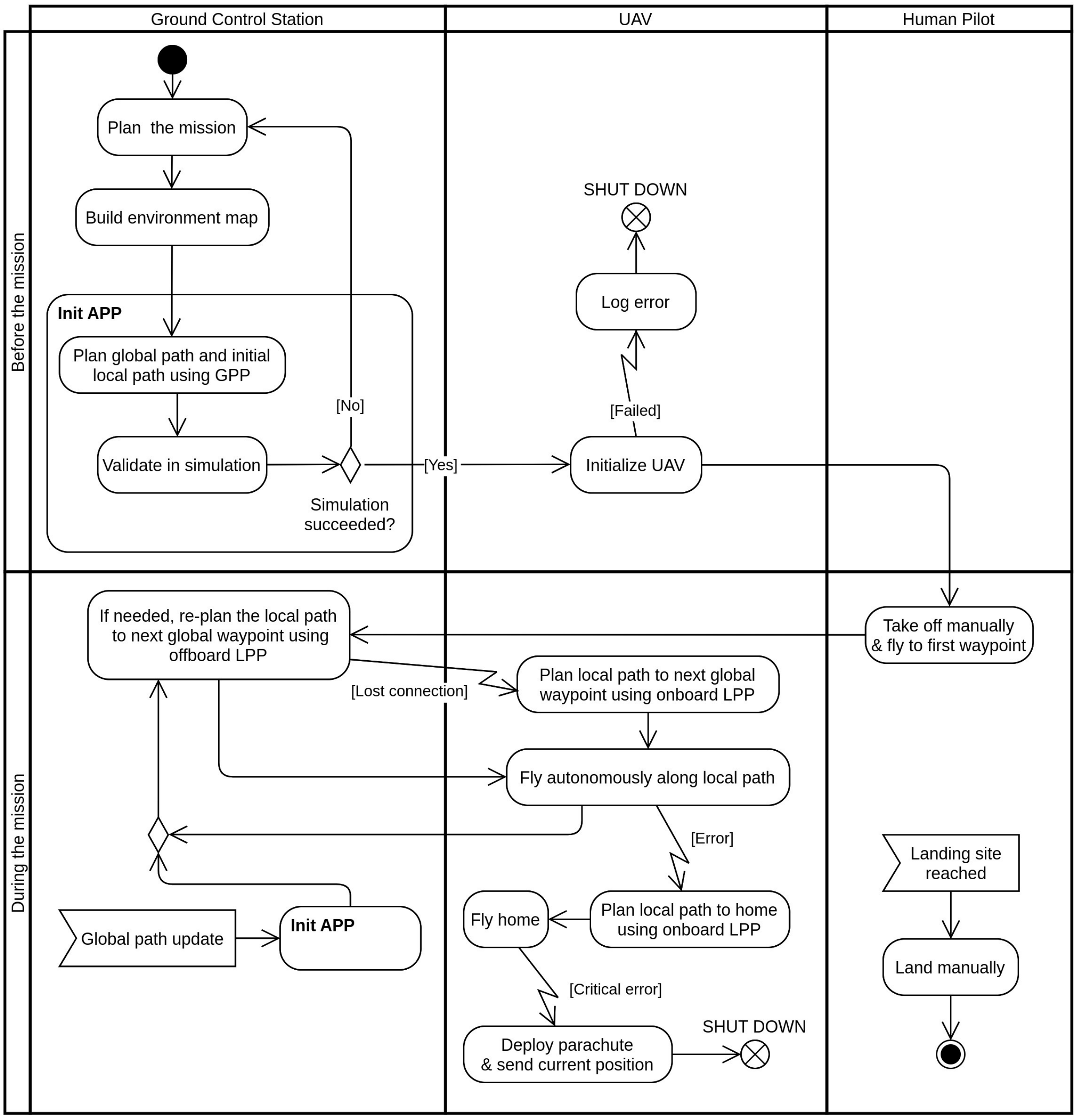
Ground Control Station
UAV
Human Pilot
Before the mission
During the mission
Plan the mission
Build environment map
Init APP
Plan global path and initial local path using GPP
Validate in simulation
[No]
[Yes]
Simulation succeeded?
SHUT DOWN
Log error
[Failed]
Initialize UAV
If needed, re-plan the local path to next global waypoint using offboard LPP
[Lost connection]
Plan local path to next global waypoint using onboard LPP
Take off manually & fly to first waypoint
Fly autonomously along local path
[Error]
Landing site reached
Global path update
Init APP
Fly home
Plan local path to home using onboard LPP
Land manually
[Critical error]
SHUT DOWN
Deploy parachute & send current position

Fig. 4.1: Typical APP-based mission planning and execution methodology

The path is send to the UAV. Before the flight starts, the UAV initializes. If errors are detected, they are logged and the platform shuts down, aborting the mission. Otherwise, the mission begins and control is passed to a human pilot. The pilot takes off manually and approaches the first global waypoint. After that, control is passed to the autonomous system and the aircraft flies along the initial path. If the next waypoint cannot be reached or the environment map is updated without supplying a new local path, LPP launches to plan a fallback path.

The local fallback path to the next global waypoint is computed offboard (i.e., in GCS) using LPP and is sent to the UAV. If connection is lost, LPP runs directly onboard. Depending on the scenario, onboard computations can be switched on by default. Now, the UAV controller tries to follow the path indicated by the local waypoints. If an error occurs, the UAV plans a path to a predefined emergency landing site to perform a Return-To-Home (RTH) maneuver.

If a critical error occurs during RTH, the UAV enters emergency mode, deploys a parachute and tries to send a call for help with its current location. Then, the major systems of the UAV shut down for safety. The exact behavior of the controller and a decisive module, which supports it, is a complex problem on its own. Also, it is not directly related to APP. Hence, it is not discussed in the thesis.

The steps as above are repeated until the UAV reaches the final global waypoint. In that case, the human pilot takes control over the UAV and lands it manually, which completes the mission.

Optionally, the global path can be also re-planned during the mission. After updating the settings, APP re-initializes while the UAV is still airborne. If the new path is valid and approved, it is sent to the UAV. LPP automatically adapts to it, if needed.

4.4 Global Path Planner

GPP is an optimization tool used to provide a quasi-optimal path according to a wide set of criteria. They include collision avoidance, environmental conditions, energy expenditure, minimal path length and mission-specific parameters, e.g., profiling parameters. The global waypoints provided by GPP are the input to LPP.

4.4.1 Problem statement

The outcome of GPP strongly depends on the general mission scenario $\mathcal{S}$, which can be described by general mission parameters, such as the placement of the human-specified waypoints, airspeed between the waypoints, parameters relating to measurement profiles etc. These factors influence several criteria such as total energy consumption, flight endurance (longevity), flight distance and the quality of measurement data. Therefore, they must be adjusted correctly.

The main purpose of the optimization process is to search for the optimal scenario $\mathcal{S}$ of the mission and the optimal values of mission parameters $\mathbf{p}$ in order to obtain the best global path $\mathcal{P}^G$. This issue is viewed as a multi-objective optimization problem and hence it is stated as

$$\begin{aligned} &\text{minimize } \mathbf{C}(\mathcal{S},\mathbf{p}) = c_1(\mathcal{S},\mathbf{p}), c_2(\mathcal{S},\mathbf{p}), ..., c_n(\mathcal{S},\mathbf{p}), ..., c_N(\mathcal{S},\mathbf{p}) \\ &\text{subject to } \Omega(\mathcal{S},\mathbf{p}) \end{aligned} \tag{4.1}$$

where $\mathcal{S}$ and $\mathbf{p}$ represent decision variables as above, c_n is the n-th criterion function of non-conflicting objectives, $n = 1, 2, ..., N$. Ω denotes the set of the constraints and boundaries resulting from general mission requirements, law regulations, expert's knowledge etc. The general mission scenario $\mathcal{S}$ can be determined using a part of decision variables corresponding to mission parameters $\mathbf{p}$ to reduce the complexity of the problem in some cases.

Mission parameters

The mission parameters $\mathbf{p}$ are expressed in a general vector form

$$\mathbf{p} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_n & \cdots & \mathbf{w}_{N_G} & \mathbf{g} \end{bmatrix}$$

where $\mathbf{w}_n$ is the n-th vector of waypoints, $n = 1, 2, ..., N_G$, and $\mathbf{g}$ represents the vector of global mission-specific settings. The number of waypoints optimized by GPP is expressed by N_G. The parameters in $\mathbf{w}_n$ are defined per-waypoint, while $\mathbf{g}$ contains the parameters defined globally for the whole mission. Each *global waypoint* w has the similar form as in eq. (3.4), but extended by heading angle. It is given by

$$\mathbf{w} = \begin{bmatrix} w_x & w_y & w_z & w_\chi \end{bmatrix} \tag{4.2}$$

where w_x, w_y, w_z represent the requested position of the UAV relative to the inertial frame $\mathcal{F}^i$. Variable w_χ denotes its heading in $\mathcal{F}^i$. Thus, each waypoint $\mathbf{w}$ represents a position with 4 degrees of freedom. The vector $\mathbf{g}$ is highly dependent on the actual scenario S, but in general case it has the form

$$\mathbf{g} = \begin{bmatrix} g_1 & g_2 & \cdots & g_n & \cdots & g_{N_g} \end{bmatrix} \tag{4.3}$$

where g_n is the n-th of a total of N_g global mission parameters.

Generally, multi-objective problems can have infinite number of local as well as global extrema and therefore one should investigate a set of points, each of which satisfies the objectives. Because of this, the predominant Pareto optimality concept [45] is adopted.

The Pareto-optimal solution can be considered the same as a non-dominated solution. It exists if no solution exists that improves at least one objective function without worsening others. In this research it is decided to apply the estimation of fundamental measures as objectives, which are described in-detail in the section 4.4.2.

Reducing to single-objective cost function

Alternatively, the global criterion method can be applied to transform a multiple-objective cost function (4.1) into a single-objective one [45]. Therefore, an indirect utility function can be expressed in its simplest form of meta-criterion function as the weighted sum of objectives

$$\mathbf{C}(\mathcal{S}, \mathbf{p}) = C(\mathcal{S}, \mathbf{p}) = \sum_{n=1}^{N_c} \omega_n c_n \tag{4.4}$$

where ω_n is the weight of the n-th criterion c_n. N_c stands for the number of criteria. The optimal solution is found if the criterion function C has a relative minimum at $\mathbf{p}^*$, i.e.,

$$\mathbf{p}^* = \underbrace{\operatorname{argmin}}_{\mathbf{p}\in\Omega} \left(\mathbf{C}(\mathcal{S}, \mathbf{p})\right).$$

Different optimization algorithms can be utilized for solving the problem stated in (4.1) or (4.4). The objectives are stochastic and $\mathbf{p}$ contains both continuous and discrete decision variables. Therefore, hard computing optimization methods, such as gradient-based approaches, cannot be adapted in this study.

Moreover, pure stochastic optimization methods, for example Monte Carlo techniques, will not be able to find an accurate solution while guaranteeing polynomial-time convergence. Instead, soft computing optimization methods may be used to find the global minimum of the function. In the thesis, different heuristic algorithms are applied, such as I-GWO, PSO, GA and ACO$_\mathbb{R}$. Most of these algorithms are described in appendix B.

General form of a measurement mission

GPP optimizes the placement of global waypoints as well as mission-specific parameters, e.g., parameters of the measurement path. The path, however, must be subject to the kinematic constraints of the UAV. To reduce the number of optimized waypoints, GPP only optimizes the waypoints, which act as control points. Then, these points are connected using Dubins airplane paths to form smooth kinematically-feasible path. The algorithm computes all feasible Dubins paths for a given pair of control waypoints and picks the shortest. Now, assume a mission scenario as follows:

1. The UAV takes off and flies to the area with the highest pollution concentration.

2. The aircraft flies through the area following a parametric measurement path.
3. The UAV returns to the landing site and lands.

Note, the takeoff and landing are handled either manually by a human operator or by a dedicated autonomous system, which is not a part of the planning algorithm described in the thesis.

The measurement path can be modeled by a parametric polygon, which represents the area, where the measurements should be taken. The *measurement polygon* can be then converted to a path by employing an area coverage algorithm configured to respect the kinematic constraints of the aircraft, e.g, the maximum coverage algorithm by Torres et al. [46].

In the thesis it is assumed the exact shape of the polluted area is not known *a priori*. Hence, to cover the area uniformly in all directions, the measurement polygon should be a circle. For computation purposes, however, it is approximated by a regular polygon. Therefore, the vector of optimized variables finally takes the form

$$\mathbf{p} = \begin{bmatrix} \mathbf{W}^S & \mathbf{W}^G & N_S & N_G & \mathbf{w}^M \end{bmatrix} \tag{4.5}$$

where $\mathbf{W}^S$ and $\mathbf{W}^G$ are the vectors of global waypoints from start (takeoff end point) to the beginning of the measurement path, and from the end of the measurement path to goal (the landing start point), respectively. The start and goal point are the points, where the control is passed to/from the path planner.

To keep the form of the optimization vector constant, the variables N_S and N_G are used to optimize the exact number usable of waypoints in $\mathbf{W}^S$ and $\mathbf{W}^G$. The minimal number is 0 (start or goal waypoint connected with a single Dubins path to the measurement path) and the maximal number is a parameter of GPP. The variable $\mathbf{w}^M$ represents the position of the geometric center of the measurement polygon. The exact shape of the polygon is defined by parameters not optimized by GPP.

Depending on the algorithm used GPP can plan the path in a such way it bases on a template path set by a human expert prior to the optimization phase. In that case, a human expert defines the initial values of the optimization vector $\mathbf{p}$ based on their knowledge. GPP tries then to further optimize this paths by solving the optimization problem as above.

4.4.2 Defining the criteria

Below the criteria used by GPP are given. The criteria are defined in a such way, they provide a numerical value inverse proportional to the optimality of the solution according to the specific criterion. That is, the further from the optimal solution, greater the value of the criterion, and thus the value of the cost function.

The criteria considered by GPP in the thesis are given in Tab. 4.1. It should be noted, that only a subset of the criteria will be used during an individual mission, depending on the chosen scenario. The list can be further expanded with other criteria, for example, considering aircraft icing as mentioned by Andrade in [40].

Obstacle avoidance

Obstacle avoidance is modeled by penalizing the optimizer for placing the waypoints in a such way, that the shortest Dubins path $\mathcal{D}$ computed for a pair of control waypoints crosses an obstacle. GPP assumes the path is infeasible if it spans below the ground level specified by the terrain map $\mathcal{M}_t$ or if it crosses an NFZ defined by the ROIs in the airspace map $\mathcal{M}_a$. Therefore, the criterion function has two distinct parts.

Tab. 4.1: Summary of the criteria used by GPP

#	Criterion	Necessity	Comments
1	Obstacle avoidance	Required	Penalizes the planner if the Dubins path goes through an infeasible region
2	Minimal path length	Optional	Increases the cost proportional to the total length of the Dubins paths between the consequent control waypoints
3	Wind influence	Optional	Modifies the path using the heuristic rules, which define the preferred behavior in the presence of wind
4	Energy expenditure	Optional	Increases the cost proportional to the energy expended during the flight; estimated using heuristic rules
5	Pollutant concentration	Optional	Places the geometric center of the measurement area in the point of highest pollution

The first part checks if any *Dubins waypoint* $\mathbf{w}_n^{\mathcal{D}}$ on the shortest Dubins airplane path $\mathcal{D}$ is placed inside terrain. This check is done using the `free` $(\cdot)$ operator defined in eq. (3.5), that is

$$c_{o1} = \sum_{n=1}^{N_{\mathcal{D}}} \texttt{free}\,(\mathbf{w}_n^{\mathcal{D}}, \mathcal{M}_t), \qquad \mathbf{w}_n^{\mathcal{D}} \in \mathcal{D} \tag{4.6}$$

where $\mathbf{w}_n^{\mathcal{D}}$ is the n-th Dubins waypoint sampled along the shortest Dubins airplane path $\mathcal{D}$. $N_{\mathcal{D}}$ is the total number of Dubins waypoints sampled on $\mathcal{D}$. Sampling density is a parameter of GPP. $\mathcal{M}_t$ denotes a generic terrain map, that is, any of $\mathbf{M}_t^V$, $\mathbf{M}_t^D$ or $\mathbf{M}_t^C$.

The second part tests if the same Dubins waypoint $\mathbf{w}_n^{\mathcal{D}}$ intersects with an NFZ ROI. Again, the check is made using the `free` $(\cdot)$ operator. This time, however, the definition for the airspace map $\mathcal{M}_a$, i.e., from eq. (3.13), is used. The second part of the criterion function is given by

$$c_{o2} = \sum_{n=1}^{N_{\mathcal{D}}} \texttt{free}\,(\mathbf{w}_n^{\mathcal{D}}, \mathcal{M}_a), \qquad \mathbf{w}_n^{\mathcal{D}} \in \mathcal{D} \tag{4.7}$$

where $\mathcal{M}_a$ is the airspace map. Finally, the complete obstacle avoidance criterion can be formally described by

$$c_o = s_{fo}(c_{o1} + c_{o2}) = s_{fo} \sum_{n=1}^{N_{\mathcal{D}}} \left(\texttt{free}\,(\mathbf{w}_n^{\mathcal{D}}, \mathcal{M}_t) + \texttt{free}\,(\mathbf{w}_n^{\mathcal{D}}, \mathcal{M}_a) \right) \tag{4.8}$$

where s_{fo} is a scaling factor. Note, while both `free` $(\cdot)$ operators are logically the same, their implementation differs depending on the kind of the supplied map.

For 2.5D representation (elevation-based maps $\mathbf{M}_t^D$ or $\mathbf{M}_t^C$), terrain-based checks are made using a simple inequality as in eqs. (3.7) and (3.8). Therefore, the first check can be implemented as a constraint instead. Now, using a constrained optimization method and violating the constraint makes the path infeasible. In the case of constrained optimization, c_o is reduced to its second part, which checks the airspace map only, that is

$$c_o = s_{fo} c_{o2} = s_{fo} \sum_{n=1}^{N_{\mathcal{D}}} \texttt{free}\,(\mathbf{w}_n^{\mathcal{D}}, \mathcal{M}_a). \tag{4.9}$$

Note, the cost in eq. (4.8) and (4.9) could be normalized by dividing by the total number of waypoints. However, this could result in relatively low cost of long paths with just a few colliding waypoints, unless s_{fo} is set to a very high value. Therefore, the criterion function was purposefully left not normalized to induce severe penalty for just a single collision, but also to differentiate between paths with varying number of colliding waypoints.

Minimal path length

Minimizing the length of the path is maintained by the cost function c_m, which sums the lengths of the shortest Dubins paths $\mathcal{D}$ computed for each pair of the control waypoints. The result is then scaled to maintain consistency with the other criterion functions. The function is given by

$$c_m = s_{fm} \sum_{n=1}^{N_{\mathcal{D}}} \|\mathcal{D}_n\| = s_{fm} \|\mathcal{D}\|, \qquad \mathcal{D}_n \in \mathcal{D} \tag{4.10}$$

where $\|\mathcal{D}_n\|$ denotes the length of the n-th Dubins path segment of the whole Dubins airplane path $\mathcal{D}$. Variable s_{fm} is the scaling factor and $N_{\mathcal{D}}$ denotes the number of Dubins waypoints in $\mathcal{D}$.

Wind influence

To reduce the computation complexity, the effects of wind in this study are modeled using simplified heuristic metrics. They are formed considering the heading of the UAV versus wind direction and altitude changes. The metrics follow the simple rules listed below:

- Crosswind is always considered disadvantageous, that is, UAV should minimize flying perpendicularly or near-perpendicularly to wind.
- Headwind is beneficial when rapidly changing altitude, i.e., if the next waypoint is located significantly higher, but horizontally close.
- Tailwind is found disadvantageous when gaining or losing height, but otherwise it is considered beneficial as it increases the ground speed of the UAV.

Given the waypoints $\mathbf{w}_n^{\mathcal{D}}$ and $\mathbf{w}_{n+1}^{\mathcal{D}}$ placed on a Dubins path $\mathcal{D}$, let Δ_n^v be the vertical difference between the position of waypoints, that is

$$\Delta_n^v = |z_n - z_{n+1}|, \qquad \mathbf{w}_n^{\mathcal{D}} = \begin{bmatrix} x_n & y_n & z_n & \chi_n \end{bmatrix}, \qquad \mathbf{w}_n^{\mathcal{D}} \in \mathcal{D}$$

where z_n is the absolute altitude of waypoint $\mathbf{w}_n^{\mathcal{D}}$ in $\mathcal{F}^i$ and N is the total number of waypoints sampled with a constant step along $\mathcal{D}$. Similarly, let Δ_n^h denote the horizontal difference between the waypoints

$$\Delta_n^h = \sqrt{(x_n - x_{n+1})^2 + (y_n - y_{n+1})^2}, \qquad \mathbf{w}_n^{\mathcal{D}} = \begin{bmatrix} x_n & y_n & z_n & \chi_n \end{bmatrix}, \qquad \mathbf{w}_n^{\mathcal{D}} \in \mathcal{D}$$

where x_n and y_n are the absolute distances of waypoint $\mathbf{w}_n^{\mathcal{D}}$ in $\mathcal{F}^i$, i.e., respectively along the x and y axes. N denotes the total number of waypoints sampled along $\mathcal{D}$. Now, assume v_{wn} is the speed and χ_{wn} is the direction of wind[2] in the vehicle frame $\mathcal{F}^v$ measured at $\mathbf{w}_n^{\mathcal{D}}$. Let the variable Δ_n^χ express a relative angle measured between $\overline{\mathbf{w}_n^{\mathcal{D}}\mathbf{w}_{n+1}^{\mathcal{D}}}$ and χ_{wn}

$$\Delta_n^\chi = \angle\left(\overline{\mathbf{w}_n^{\mathcal{D}}\mathbf{w}_{n+1}^{\mathcal{D}}}\right) - \chi_{wn}$$

[2] The wind direction uses the same convention as the UAV, i.e., $0°$ denote the wind blowing along the x axis, that is, from $-x$ to $+x$. The direction of $90°$ denote the wind blowing along the y axis, that is, from $-y$ to $+y$ etc.

where Δ_n^χ, $\angle\left(\overline{\mathbf{w}_n^\mathcal{D}\mathbf{w}_{n+1}^\mathcal{D}}\right)$ and χ_{wn} are given in $\mathcal{F}^i$. The wind-induced cost is modeled by normal (Gaussian) distribution

$$\mathcal{N}(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)}$$

where μ and σ represent the parameters of the distribution, i.e., the mean and the standard deviation, respectively. To increase readability, each of the characteristic wind components will be defined separately. Crosswind is modeled as two Gaussian distributions centered at the sides of the aircraft, that is, at $\pm\frac{\pi}{2}$. Thus, crosswind model is given by

$$c_{w1} = \sum_n^{N_\mathcal{D}-1}\left(\mathcal{N}\left(\Delta_n^\chi,\frac{\pi}{2},\frac{\pi}{6}\right)+\mathcal{N}\left(\Delta_n^\chi,\frac{\pi}{2},\frac{\pi}{6}\right)\right)$$

where the standard deviation of $\frac{\pi}{6}$ was chosen empirically and $N_\mathcal{D}$ denotes the number of waypoints sampled along $\mathcal{D}$. Crosswind-related cost is static in terms of the UAV behavior. Headwind is defined by $\mathcal{N}(\cdot)$ centered at the nose of the aircraft, i.e., at the angle of 0. Headwind can be expressed as

$$c_{w2} = \sum_n^{N_\mathcal{D}-1}\left(\mathcal{N}\left(\Delta_n^\chi,0,\frac{\pi}{6}\right)\left(\frac{\pi}{4}-\text{atan2}\left(\Delta_n^v,\Delta_n^h\right)\right)\right)$$

where Δ_n^v and Δ_n^h are respectively the vertical and horizontal distance between $\mathbf{w}_n^\mathcal{D}$ and $\mathbf{w}_{n+1}^\mathcal{D}$. The last component, tailwind, is modeled as

$$c_{w3} = \sum_n^{N_\mathcal{D}-1}\left(\left(\mathcal{N}\left(\Delta_n^\chi,-\pi,\frac{\pi}{6}\right)+\mathcal{N}\left(\Delta_n^\chi,\pi,\frac{\pi}{6}\right)\right)\left(\text{atan2}\left(\Delta_n^v,\Delta_n^h\right)-\frac{\pi}{4}\right)\right)$$

Finally, the heuristic rules to model wind influence are given by

$$c_w = s_{fw}(c_{w1}+c_{w2}+c_{w3}) \tag{4.11}$$

where s_{fw} is a scaling factor. a similar approach, i.e., summing several Gaussian distributions, was employed by Socha and Dorigo in their ACO$_\mathbb{R}$ algorithm. They used a weighted sum (called Gaussian kernel), however [47].

The impact of this criterion depending on the current behavior of the UAV is shown in Figs. 4.2, 4.3 and 4.4. Line plots on the left side show the unscaled cost as a function of Δ_n^χ. Polar plots on the right side show total wind-induced cost. Red areas penalize the optimizer with positive cost, while green areas generate negative cost. If the aircraft must ascend/descend steeply, headwind is highly beneficial, as in Fig. 4.2. As the steepness decreases, neither headwind nor tailwind is preferred (Fig. 4.3). If the UAV should maintain height, tailwind is favored, as in Fig. 4.4, to promote flying with higher horizontal speed.

Energy expenditure

Energy expenditure is modeled using a similar approach to the one used by Andrade [40]. Power required by the propeller-driven aircraft is expressed through

$$P_r = F_{drag}v_a = \sqrt{\frac{2(mg)^3}{\rho S}\frac{C_D^2}{C_L^3}} \tag{4.12}$$

where m is mass of the aircraft, g is standard acceleration due to gravity, F_{drag} denotes drag force, C_D and C_L are its aerodynamic drag and lift coefficients, respectively. Air density is denoted by

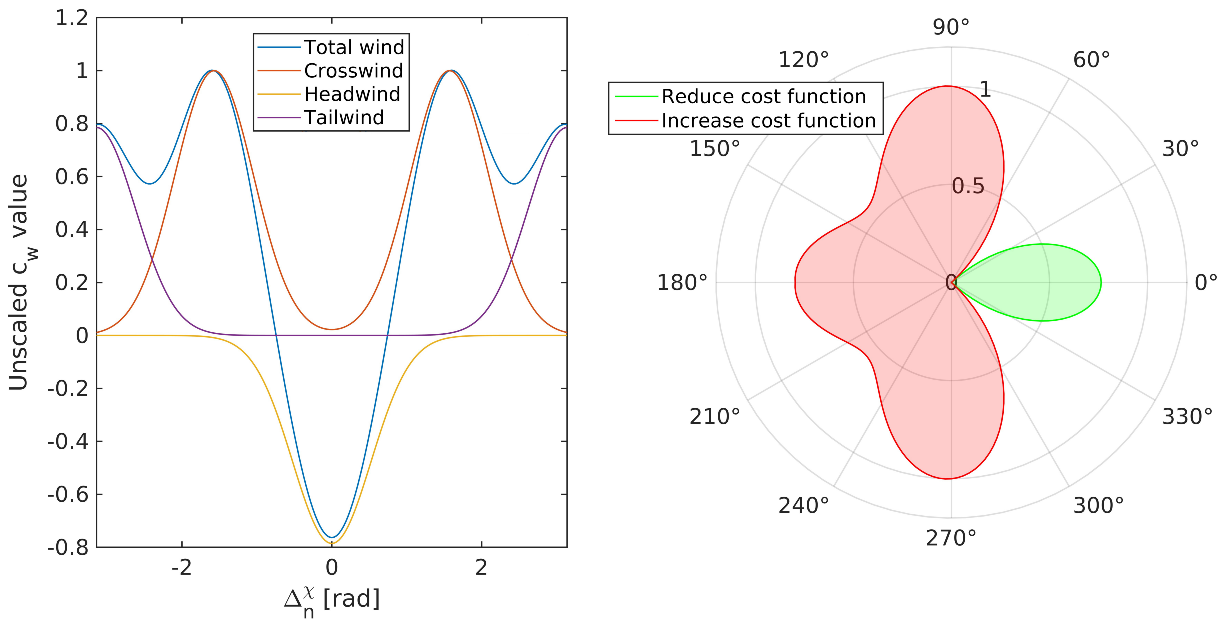

Fig. 4.2: Wind components (left) and total cost (right) for steep ascend/descend

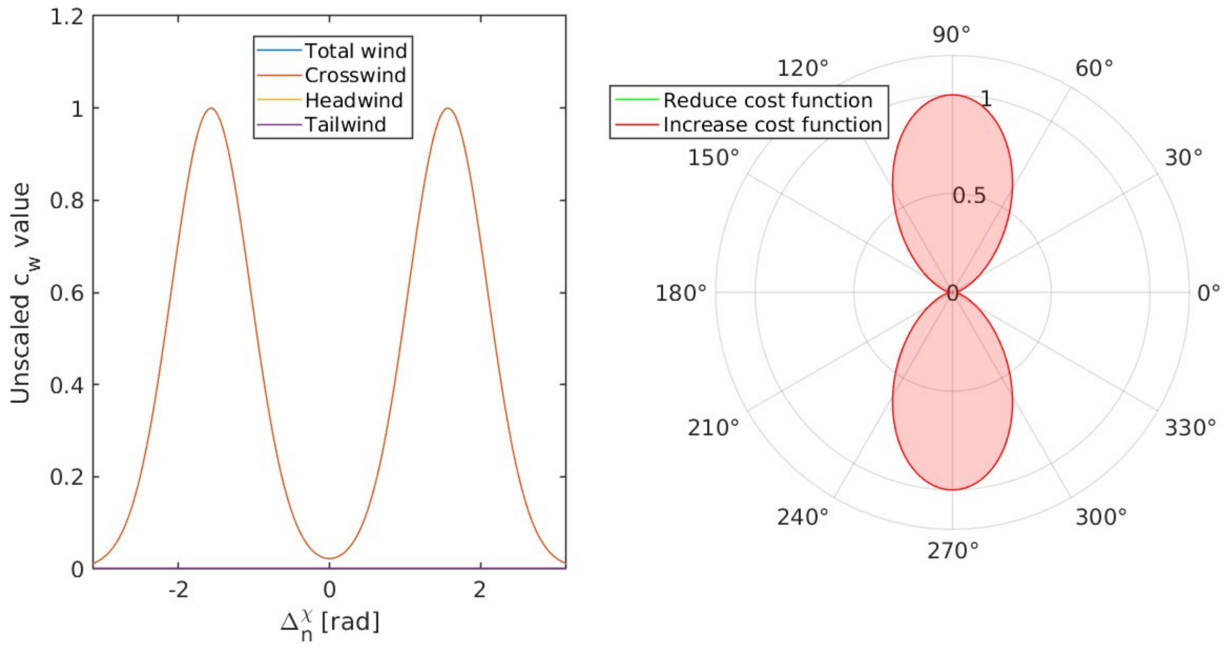

Fig. 4.3: Wind components (left) and total cost (right) for mild ascend/descend

ρ and S represents the aircraft's effective wing surface [40, 48]. Alternatively, an interpolated lookup function can be formed from experimental data if available. The experiments in the thesis use the second approach. The energy expenditure criterion function is then given as

$$c_e = s_{fe}\Big(P_{ctrl}t_{max} + \int_0^{t_{max}} f\big(v_a(t), z(t), \theta(t), \mathbf{E}\big)dt\Big) \tag{4.13}$$

where P_{ctrl} is constant estimated power required by the controller and peripherals, $f(\cdot)$ is an approximating function, $v_a(t)$ denotes airspeed, $z(t)$ is altitude AGL, $\theta(t)$ represents pitch angle, t_{max} denotes flight maximum time and s_{fe} is a scaling factor. $\mathbf{E}$ holds the reference energy data, similarly as for a continuous wind map $\mathbf{M}_w^C$ in eq. (3.9). Estimated flight time is denoted by t.

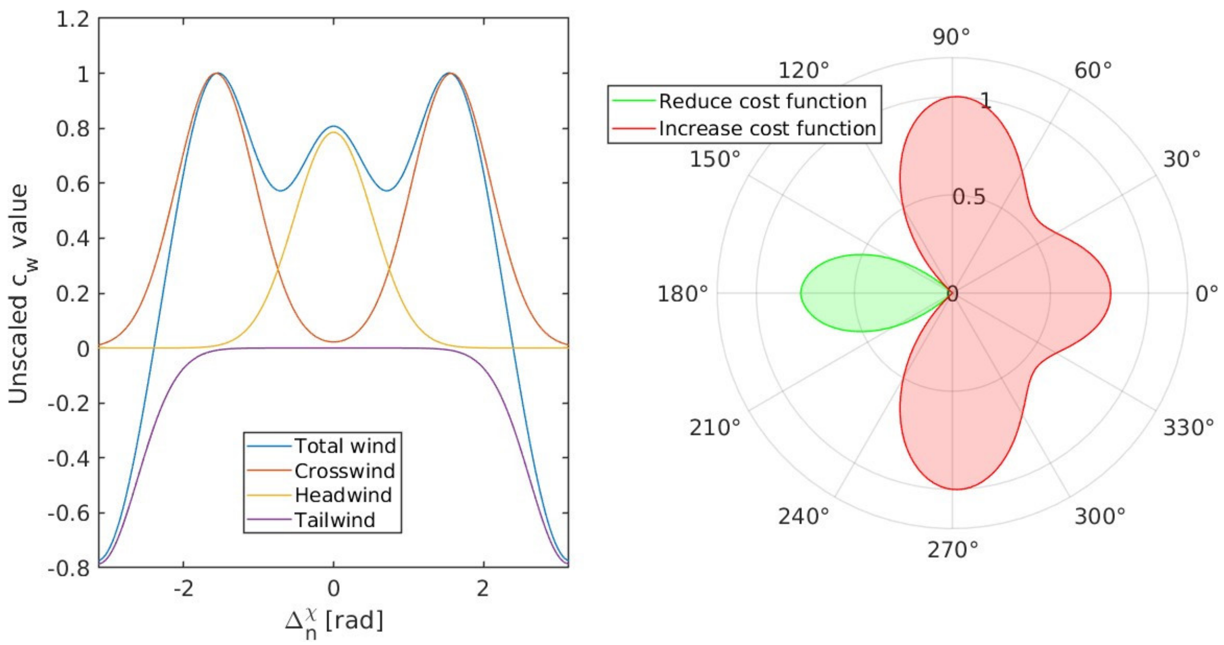

Fig. 4.4: Wind components (left) and total cost (right) for maintaining height

If wind influence is not considered and v_a is constant, t_max is equal to the total path length divided by v_a.

Unless specified otherwise, the thesis uses simplified criterion function (4.13) by default. An example propeller power estimation using simulation data is presented in Fig. 4.5. Power requirement was calculated for a range of feasible angles of attack α. Simulation data was acquired thanks to the courtesy of Kamil Zenowicz.

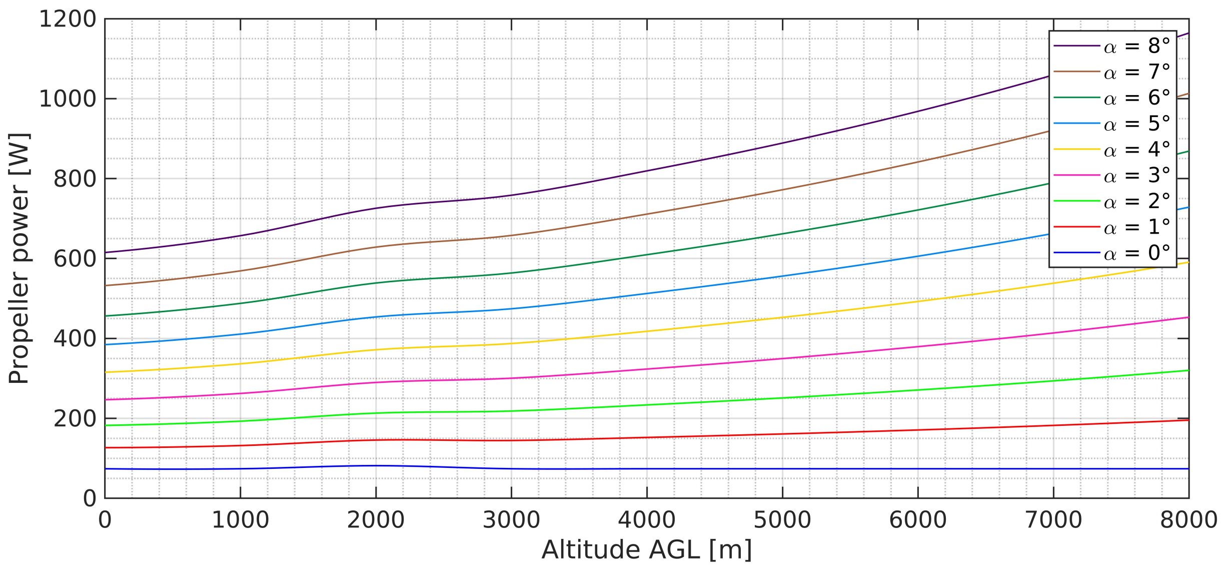

Fig. 4.5: Propeller power interpolated from simulation data

4.4.3 Pollutant concentration

The final criterion attracts the UAV to the region of the highest pollutant concentration and optimizes the placement of the measurement path used for pollution sampling. However, the exact shape of the polluted region is not known *a priori* and is revealed after taking measurements with the UAV. Therefore, the shape of the measurement area is not optimized and must be supplied externally, e.g., by a human expert. The problem reduces then to optimizing the placement of the measurement polygon.

Assume the measurement polygon $\triangle_m$ is defined similarly as bases of ROIs in eq. (3.11) for airspace maps, but has a third dimension z shared between all the vertices, that is

$$\triangle_m = \{(x_i, y_i, z)\}, \qquad i = 3, 4, ..., N \tag{4.14}$$

where x_i and y_i are the horizontal coordinates of the i-th vertex of the polygon. Now, assume $\mathbf{w}^m$ denotes the geometric center (centroid) of $\triangle_m$. The criterion generates cost proportional to pollutant concentration measured at $\mathbf{w}^m$. Formally it is expressed by equation

$$c_p = s_{fp}\,\texttt{pollution}\,(\mathbf{w}^m, \mathcal{M}_m) \tag{4.15}$$

where s_{fp} is a scaling factor[3], $\mathcal{M}_m$ is a measurement map and $\texttt{pollution}\,(\cdot)$ is the operator defined in section 3.5.1.

Note, that if the weight of this criterion is set to zero, finding polluted areas becomes impossible. The optimization problem reduces then to connecting the start and goal waypoints without performing the measurements. Thus, generating the measurement path is effectively skipped and the algorithm can be used to solve simple point-to-point flights.

4.5 Local Path Planner

LPP is an algorithm designed to quickly provide a feasible solution, even if it is not the globally optimal one. While GPP is meant to be used offboard only, i.e., in the ground control station, LPP is designed to be feasible also onboard. Thus, while GPP targets optimality, LPP focuses on responsiveness, simplicity and minimal computation effort. LPP uses a mix of a fast 3D path planning algorithm (e.g., RRT or RRT*) and Dubins airplane paths, which respect the kinematic constraints of the fixed-wing UAV.

4.5.1 Problem statement

The main and only task of LPP is to provide a collision-free 3D Dubins path from the current position of the fixed-wing UAV to the next waypoint supplied by GPP, while conforming to the kinematic constraints of the aircraft. First, assume that all the obstacles in the scene are elements of an obstacle map $\mathcal{M}_o$, such that

$$\mathcal{M}_o = \mathcal{M}_t \cup \mathcal{M}_a$$

where $\mathcal{M}_t$ is a terrain map and $\mathcal{M}_a$ denotes the obstacles inside the airspace map. Now, let $\mathcal{P}^L$ be the *desired local path* planned by LPP. Hence, the problem can be expressed as

$$\begin{aligned}&\text{find } \mathcal{P}^L(\mathcal{S}) \in \mathcal{D} | \mathcal{P}^L(\mathcal{S}) \cap \mathcal{M}_o(\mathcal{S}) = \{\emptyset\}\\ &\text{subject to } \Omega_k(\mathcal{S})\end{aligned}$$

where $\mathcal{S}$ denotes the current scenario and $\mathcal{D}$ represents the set of admissible paths of a Dubins airplane as in eq. (2.7). $\mathcal{M}_o(\mathcal{S})$ denotes a scenario-dependent obstacle map and $\Omega_k(\mathcal{S})$ is the set of scenario-dependent kinematic constraints of the UAV.

[3] Depending on the implementation of $\mathcal{M}_m$, the sign of s_{fp} must be chosen accordingly to form a cost function. If $\texttt{pollution}\,(\cdot)$ provides positive values, s_{fp} must be also positive, for example.

As actual environment conditions during flight are unknown *a priori*, some reactivity of the onboard controller is desired. APP assumes the controller can counteract, e.g., wind gusts that are impossible to accurately predict using only APP.

In practice, $\mathcal{P}^L$ is approximated by a vector $\mathbf{W}^L$ of waypoints $\mathbf{w}^L$, which formulates a *local path*, or formally

$$\mathbf{W}^L = \begin{bmatrix}\mathbf{w}_1^L & \mathbf{w}_2^L & \cdots & \mathbf{w}_n^L & \cdots & \mathbf{w}_N^L\end{bmatrix}^T \in \mathcal{W}^L$$

where $\mathbf{w}_n^L$ represents the n-th *local waypoint* of a total N local waypoints defined similarly to (4.2). Local waypoint $\mathbf{w}_1^L$ always represents the current state of the UAV at the time of calculating the local path. Hence, the final form of the problem statement is given as

$$\begin{aligned}&\text{find } \mathbf{W}^L = \begin{bmatrix}\mathbf{w}_1^L & \mathbf{w}_2^L & \cdots & \mathbf{w}_n^L & \cdots & \mathbf{w}_N^L\end{bmatrix}^T \in \mathcal{D} | \, \texttt{free}\left(\mathbf{W}^L, \mathbf{M}_o\right) = 1 \\ &\text{subject to } \Omega_k[n], \qquad\qquad n = 1, 2, ..., N_L\end{aligned} \tag{4.16}$$

where $\texttt{free}\ (\cdot)$ denotes the obstacle-checking functions defined in eq. (3.5) or eq. (3.13), depending on the kind of used map. $\Omega_k[n]$ is the discretized set of UAV kinematic constraints, updated after reaching $\mathbf{w}_n^L$. N_L is the number of local waypoints $\mathbf{w}^L$.

4.5.2 Kinematic constraints

To be feasible, the local path must respect the kinematic constraints of the UAV. The constraints are similar to those introduced in the kinematic guidance model of a fixed-wing (2.3). The set of kinematic constraints Ω_k is therefore given as

$$\Omega_k = \{v_a, \gamma_{min}, \gamma_{max}, \theta_{max}\} \tag{4.17}$$

where v_a represents airspeed maintained between the waypoints. γ_{min} denotes the minimum value of flight-path angle (if descending), whereas γ_{max} is its maximum value (if ascending) with respect to $\mathcal{F}^{v2}$. Finally, θ_{max} refers to the maximum allowed roll angle with respect to $\mathcal{F}^b$. The admissible set $\mathcal{D}_{3D}$ consists of the paths which respect Ω_k.

It should be noted that in practice the parameters in eq. (4.17) are constraints imposed on the path planner only, not the physical limitations of the actual aircraft[4]. Hence, Ω_k should be understood as the set of kinematic constraints required for maintaining operational state of the UAV.

4.5.3 Finding obstacle-free admissible path

Obstacle avoidance is implemented similarly as in obstacle avoidance criterion function (4.4.2) formulated for GPP. However, instead of checking for the overlapping between a waypoint and obstacles directly, LPP first chooses a collision-free path $\mathcal{D}^*$ from the set of admissible Dubins airplane paths $\mathcal{D}$ (2.7). Thus, the obstacle-free path can be formally given as

$$\mathcal{D}^* \in \mathcal{D} | \mathcal{D}^* \cap \mathcal{M}_o = \{\emptyset\}$$

where $\mathcal{D}^*$ is the chosen admissible path and $\mathcal{M}_o$ is the obstacle map. Similarly to GPP, LPP supplies a set of waypoints. However, these *local waypoints* are dense enough to reflect the actual path subject to the kinematic constraints of the UAV. a path provided by LPP can be then validated in simulation employing either a kinematic or a dynamic model of the UAV. a general concept of LPP is presented in Fig.4.6.

[4] For example, γ_{min} in a real fixed-wing UAV can be as low as $(-\frac{\pi}{2})$, denoting UAV diving straight into ground. While theoretically possible, such dangerous situation should be avoided by the planner. Moreover, physical counterparts are not constant and depend on multiple parameters, such as air density and the durability of the airframe.

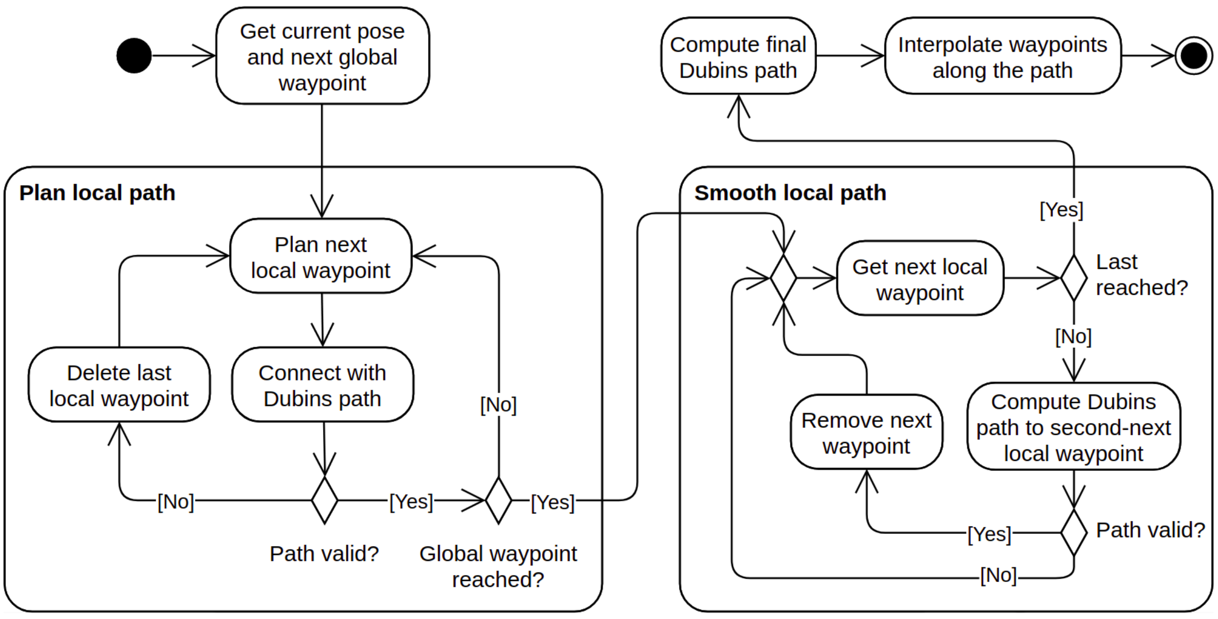

Fig. 4.6: General concept of LPP

LPP is supplied with the obstacle map, the current pose of the UAV acquired from the onboard controller and next waypoint from GPP. The planner uses then RRT or RRT* (see section B.2.5 and B.2.6) to randomly generate the next local waypoint.

The path is then validated by connecting the resulting waypoint with the previous one (or the current pose for the first iteration) with an admissible Dubins airplane path. If not valid, the waypoint is rejected and a new one is drawn. The steps repeat until the global waypoint has been reached.

Then, the path is smoothed by removing unnecessary transitional waypoints (see Algorithm 6). It is done by sequentially generating Dubins paths from a given local waypoint to the second-next one. If valid, the waypoint between is removed. Otherwise it is kept and becomes the new origin used to generate Dubins paths. Again, the process repeats until the global waypoint has been reached.

Finally, Dubins paths connecting all the local waypoints are computed and connected together. After that, the path is interpolated to provide waypoints that are evenly placed along the path to be used by the onboard controller.

To reduce energy expenditure due to frequent computations, LPP is meant to be run on demand, e.g., if the current goal is reached or the UAV deviates too much of the previously planned path. Nevertheless, depending on the control strategy employed, LPP can also be used for *proportional navigation* on a similar manner to tactical missiles. In that case only the closest local waypoint is used as an imaginary pseudo target and LPP recalculates repeatedly [49].

4.6 Simulation

Simulation is the final and also optional step used to validate the local path using the model-based approach. While necessary during mission planning in GCS, it is costly in terms of computational effort. Moreover, its results are difficult to interpret on an embedded platform with limited inference capabilities and without support from human operator. Thus, validation phase looses its importance for onboard computations. Therefore, by default simulations are run only offboard (in GCS).

Two distinct use cases of model-based simulation are considered: (1) point-to point local path and (2) complete local path. First one is to use LPP to plan a path from the current position of the UAV to the next global waypoint, that has not been reached yet. As discussed above, it allows the controller to compensate possible deviations from the initial path, e.g. due to wind gusts. This is a common behavior during the mission.

The other approach uses either the initial local path generated by GPP or generates a complete local path using LPP by sequentially supplying the last state of the current simulation as the initial condition of the next simulation to be carried out. This way, the whole path can be simulated and validated before the mission begins. The GPP-based approach is preferred for optimality, while LPP-based is favored when minimal computation time is expected.

4.7 Summary

The chapter described the model-based Adaptive Path Planner (APP). Both Global Path Planner (GPP) and Local Path Planner (LPP) were introduced starting with a problem statement, followed by the explanation of used terminology, methodology and algorithms.

The formulation of the optimization problem for single and multiple objective optimization opened the part focused on GPP. After that, each criterion function considered in the thesis was described. The next section described LPP and the methods used to implement thereof. The section covered also the simplification of kinematic constraints and the implementation of obstacle avoidance using the environment model.

The final section described the importance of model-based simulation to validate the path before finally approving it. Two alternative methods of the model-based path validation were covered.

5. Verification study

This chapter describes the verification tests performed to evaluate APP and its components: GPP and LPP. Both algorithms were implemented in MATLAB R2022a and verified in Model-in-the-Loop simulations, which employed the models of the UAV and its environment.

5.1 Specification of Twin Stratos

By the time of conducting the research the complete dynamic model of the target HALE UAV was not available yet. Building the full non-linear dynamic model of the UAV as described in [20] is also beyond the scope of the thesis. Therefore, the verification study will employ the simplified kinematic and dynamic guidance models described in Chapter 2.

First, guidance models were configured according to the mechanical parameters of the UAV. The concept of Twin Stratos (TS), the UAV designed during LEADER project [14], is shown in Fig. 5.1.

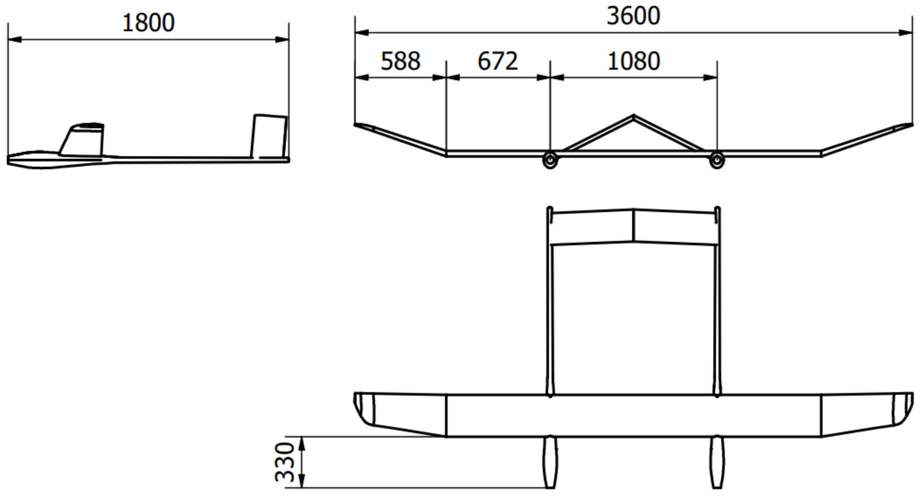

Fig. 5.1: The concept of Twin Stratos HALE UAV; courtesy of Paulina Zenowicz

TS is a fixed-wing HALE aircraft designed and developed as a part of the LEADER project [14]. TS has two fuselages, each with its own independent electrically-powered propulsion system. The fuselages contain the internal controller of the UAV, as well as basic measurement equipment used during flight. The fuselages are connected by a single wing spanning over 3600 mm in total. The wing stores lithium batteries, which are recharged during flight by solar panels placed on top of the wing. The aircraft has an A-shaped tail hosting elevons. A detailed technical specification of TS is summarized in Tab. 5.1.

5.2 Path quality metrics

To quantitatively describe the results of the experiments, a set of path quality metrics was defined. The metrics are divided into general metrics and the metrics specifically defined for GPP and for LPP.

Tab. 5.1: Technical specification of Twin Stratos; courtesy of Paulina Zenowicz

#	Parameter	Value	#	Parameter	Value
1	Wing span $[m]$	3.6	**12**	$Xc.g.$ $[m]$	0.478
2	Wing area $[m^2]$	0.896	**13**	$Yc.g.$ $[m]$	0
3	Mean aerodynamic chord $[m]$	0.238	**14**	$Zc.g.$ $[m]$	0.237
4	Wing sweep $[°]$	0	**15**	I_x $[kgm^2]$	0.0062
5	Wing dihedral $[°]$	0	**16**	I_y $[kgm^2]$	0.0199
6	Wing profile	HQ/W2,5/12	**17**	I_z $[kgm^2]$	0.0242
7	Wing profile	HQ/W2,5/11	**18**	h (flight) $[m]$	5000
8	Wing profile	HQ/W3/10	**19**	ρ $[\frac{kg}{m^3}]$	0.738
9	Fuselage lenght $[m]$	1.86	**20**	α $[°]$	0
10	Max. take-off weight $[kg]$	11.69	**21**	β $[°]$	3
11	Empty weight $[kg]$	9.19	**22**	Min speed $[\frac{m}{s}]$	19

5.2.1 General metrics

This section lists the general metrics usable for paths generated by GPP or LPP. The metrics specific to each planner are given in the further corresponding sections.

Path length

Lenght of a path directly relates to the energy expenditure and total mission time. In the thesis, path length will be measured as the sum of Euclidean distances in meters between the consequent waypoints $\mathbf{w}$ of a waypoint vector $\mathbf{W}$. The metric used to evaluate the paths is expressed similarly to the minimal length criterion (4.10), that is

$$\text{LEN} = \sum_{n=1}^{N_w-1} \|\overline{\mathbf{w}_n\mathbf{w}_{n+1}}\|, \qquad \mathbf{w}_n, \mathbf{w}_{n+1} \in \mathbf{W} \tag{5.1}$$

where N_w denotes the number of waypoints in $\mathbf{W}$. Note that LEN can be applied to the global path, the local path or the simulation path as well. The source of data will be explicitly specified each time LEN is used.

Path smoothness

The quality of a path is also described by its smoothness. Smoothness relates directly to energy and time consumption [50]. Thus, a smooth trajectory allows translates into energy and time savings. Moreover, it is beneficial to the mechanical structure of the vehicle [51]. While path smoothness is not crucial for UAVs with many degrees of freedom, such as multicopters, it becomes a key factor for more constrained airplanes. Smoothness is especially valuable for HALE UAVs due to their strict energy budget. Guillén Ruiz et al. define smoothness of a discrete path by the equation

$$\text{SMOO} = \sum_{n=1}^{N_w-1} \min\left(\alpha(Sg_n, Sg_{n+1}), \beta(Sg_n, Sg_{n+1})\right)$$

where $\alpha(Sg_{n},_{n+1})$ and $\beta(Sg_n, Sg_{n+1})$ denote the angles between the n-th path segment Sg_n and its direct successor Sg_{n+1}. N_w is the number of considered waypoints. The thesis addresses specifically the three-dimensional path planning problem, so the smoothness metric has been modified accordingly. In the 3D form, smoothness is finally defined as

$$\text{SMOO} = 1 - \frac{1}{\pi(N-2)} \sum_{n=1}^{N_w-2} \left|\angle_{min}(\overline{\mathbf{w}_n\mathbf{w}_{n+1}}, \overline{\mathbf{w}_{n+1}\mathbf{w}_{n+2}})\right| \qquad \mathbf{w}_n, \mathbf{w}_{n+1}, \mathbf{w}_{n+2} \in \mathbf{W} \tag{5.2}$$

where $\mathbf{w}_n$ denotes n-th waypoint of the waypoint vector $\mathbf{W}$, N_w is the number of waypoints in $\mathbf{W}$, and $\angle_{min}(\overline{\mathbf{w}_n\mathbf{w}_{n+1}}, \overline{\mathbf{w}_{n+1}\mathbf{w}_{n+2}})$ is the smallest angle between the path segments $\overline{\mathbf{w}_n\mathbf{w}_{n+1}}$ and $\overline{\mathbf{w}_{n+1}\mathbf{w}_{n+2}}$, $\angle_{min}(\cdot) \in \langle -\pi, \pi \rangle$. As in the case of LEN, SMOO can be applied to any path described by an one-dimensional array of waypoints. However, for densely-sampled waypoints the path must be down-sampled prior to computing its SMOO.

Fig. 5.2 shows different paths in an abstract 2D Cartesian coordinates system. The SMOO metric (5.2) returns the maximal value of 1 for a straight line (top) and the minimal value of 0 for a path, where the angles between every pair of consequent segments equal π (bottom). For example, a path with constant angle of $\frac{\pi}{2}$ between the consequent segments has smoothness of 0.5 (middle). The paths presented in Fig. 5.2 have equal length and the same number of waypoints.

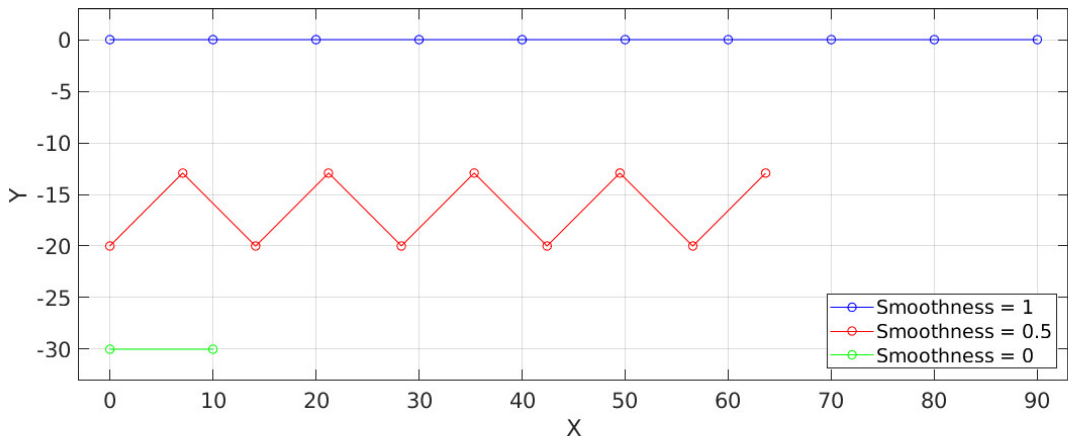

Fig. 5.2: Smoothness for different paths; the bottom path has 9 overlapping segments

Number of waypoints along the path

The metric NPTS is the number of waypoints sampled along the path. Assuming constant sampling resolution, this metric is influenced by path length and complexity. NPTS is given by

$$\text{NPTS} = \texttt{card}(\mathbf{W}) \tag{5.3}$$

where $\texttt{card}(\cdot)$ denotes the cardinality operator and $\mathbf{W}$ represents the waypoints, which form the path.

Flight time

Time of flight (FT) is the total time in seconds required to reach the goal waypoint, which marks the end of the mission[1]. Depending on the context, FT may refer to the time of flight

[1] In the thesis it is the point, where the human operator takes control over the UAV to land it manually.

estimated by the planner or the time of simulated flight.

Computation time

Computation time (CT) denotes the total time in seconds required to complete a computation task on a computer of given performance. For example, CT may refer to the time required to plan the path or, alternatively, time to complete the simulation. Note that CT of a simulation should not be confused with FT . CT measures time required to compute the simulation, while FT refers to virtual time inside the simulated environment.

Number of colliding waypoints

This metric counts the number of collisions between the waypoints generated by GPP or LPP and the obstacles. It counts the events, when the planned or simulated path intersects an object considered as obstacle, similarly to the GPP's obstacle avoidance criterion (4.8). It is formally defined as

$$\mathrm{NCOL} = \sum_{n=1}^{N} \left(\mathtt{free}\left(\mathbf{w}_n, \mathcal{M}_t\right) + \mathtt{free}\left(\mathbf{w}_n, \mathcal{M}_a\right)\right), \qquad \mathbf{w}_n \in \mathbf{W} \tag{5.4}$$

where $\mathbf{w}_n$ indicates the n-th waypoint of the total of N waypoints in the vector $\mathbf{W}_n$ of waypoints generated by LPP, GPP or computed during flight simulation. Operator $\mathtt{free}(\cdot)$ is defined as in eqs. (3.5) and (3.13), depending on whether the terrain map $\mathcal{M}_t$ or the airspace map $\mathcal{M}_a$ is checked. An example interpretation of NCOL is illustrated in Fig. 5.3. Filled circles indicate waypoints inside obstacles.

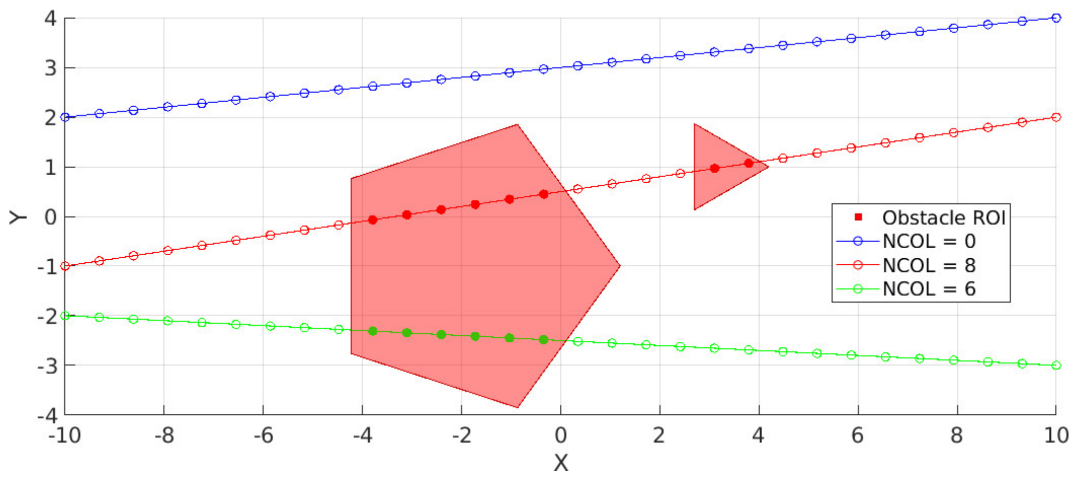

Fig. 5.3: An example of NCOL usage

5.2.2 GPP-specific metrics

A specialized set of metrics is defined to describe the output of GPP. These metrics specifically address the problem of global path optimization.

Number of cost function evaluations

GPP employs optimization algorithms with different computation complexity. To compare them, the CFE is defined as the total number of the cost function calls during a single optimization session.

Path cost

Quality of generated paths can be evaluated by comparing their COST, which equals the numerical cost returned by the cost function for the best solution computed by each algorithm.

Estimated energy expenditure

The last metric, EEE, returns the total estimated energy expanded by the UAV to travel the entire path. As the more feasible model of energy expenditure of TS is not ready by the time of writing the thesis, a simplified approach will be used. This approach uses the same equation as in the energy expenditure criterion (4.13) with the scaling factor s_{f_e} set to 1, that is

$$\text{EEE} = t(P_{ctrl} + f(v_a, z, \theta, \mathbf{E})). \tag{5.5}$$

If simulation data[2] is available, v_a can be replaced with v_g, which includes wind influence. This way the energy estimation is more accurate and the metric becomes

$$\text{EEE} = t(P_{ctrl} + f(v_g, z, \theta, \mathbf{E})). \tag{5.6}$$

5.2.3 LPP-specific metrics

A specialized set of metrics is defined to describe the output of LPP. The metrics consider the specific parameters of the underlying RRT algorithms.

Number of RRT vertices

LPP employs different RRT-based algorithms to provide an obstacle-free path. Two metrics are defined to quantify the complexity of a generated random tree. NTREE is the number of vertices of the tree. Hence, NTREE conveys information about the complexity of the whole tree. NTREE is defined as

$$\text{NTREE} = \texttt{card}\,(V) \tag{5.7}$$

where V represents the number of vertices of the tree, as in Algorithms 5 and 7. In BiRRT, two trees grow separately from the start and goal waypoints until their branches connect. Hence, for RRT $V = V_s \cup V_g$, where V_s and V_g are the vertices of the trees originating from the start and goal waypoint, respectively.

Number of RRT vertices in the chosen branch

NRRT metric equals the number of vertices of the chosen branch of the tree, which connects the start and goal waypoints. These vertices act as control waypoints for Dubins paths, which provide the final smoothed form of the path. Thus, NRRT positively correlates with the complexity of the final path. Intuitively, the smaller NRRT means simpler path. The metrics is formally given by

$$\text{NRRT} = \texttt{card}\,(\mathcal{W}) \tag{5.8}$$

where $\mathcal{W}$ indicates only the vertices forming the shortest path, as in Algorithms 5 and 7. The minimal value of both NTREE and NRRT is 2 for a path consisting of a single tree edge. Comparing NRRT with NTREE tells about the effectiveness of the search and the algorithm's convergence.

[2] Although wind data is available during the optimization, any changes in the path are not reflected until the simulation phase. Therefore, EEE would still be inaccurate even though v_g could be calculated.

5.3 General research plan

Below a general plan of the verification study on APP is given. The study considers the verification of used models, validation of GPP and LPP independently and finally compound use cases, which mix the functionality of both.

1. **Verification of the models**
   (a) **Terrain map**
   The terrain elevation map built according to the described methodology is compared with the alternative approach, i.e., the voxel-based occupancy map.
   (b) **Wind map**
   The map of wind speed and velocity forecasts is validated by comparing it with data from publicly available meteorological web services.
   (c) **Kinematic guidance model**
   Parameters of the fixed-wing kinematic guidance model from eq. (2.3) are tuned and evaluated in simulation.
   (d) **Discussion**
2. **Global Path Planner**
   (a) **Effects of the criteria**
   The actual effects of an individual criterion are evaluated on abstract use cases.
   (b) **Comparison of the chosen single-objective optimization methods**
   Different single-objective optimization methods, i.e., I-GWO, PSO, GA and $ACO_{\mathbb{R}}$, are implemented in GPP and tested on a simple mission scenario.
   (c) **Discussion**
3. **Local Path Planning**
   (a) **Comparison of chosen RRT-based algorithms**
   Diverse RRT-based algorithms are compared and the basic functionalities of LPP are validated. Then, the optimal algorithm is chosen for the following experiments.
   (b) **Tuning the chosen algorithm**
   Chosen RRT-based algorithm is fine-tuned to provide the optimal results for LPP.
   (c) **Validation of dynamic replanning in simulation**
   LPP is used to dynamically generate obstacle-free paths during simulated flight.
   (d) **Discussion**
4. **Adaptive Path Planning for pollution sampling**
   (a) **Smog profiling**
   GPP and LPP are verified in simulation of a smog profiling mission over the city of Żywiec (Poland).
   (b) **Black carbon concentration**
   Simulated measuring of the concentration of black carbon over the Kongsvegen glacier (Svalbard, Norway).
   (c) **Discussion**
5. **Summary of the experiments**

5.4 Verification of the models

This section targets the environment models defined in chapter 3, and also the kinematic guidance model of the fixed-wing.

5.4.1 Terrain map

Section 3.2 describes two alternative implementations of terrain maps: 3D voxel-based and 2.5D elevation-based. In the preliminary research the first was used for its ability to model complex 3D shapes while maintaining high computation performance. Nevertheless, as the thesis focuses on a HALE-class UAVs, a simpler approach was found.

The performance of a voxel-based full 3D terrain map was compared with simplified 2.5D elevation-based representation in MATLAB 2022a. The elevation map was implemented as a custom class, while the voxel map used a modified *occupancyMap3D* class supplied in MATLAB. The test involved building analogous terrain maps using voxel-based terrain map and elevation-based continuous terrain map as illustrated in Fig. 5.4. Then, a random set of 1e6 waypoints was drawn, and collision checks for the whole set were performed. The first experiment involved querying the maps point-by-point using a for loop. The second one queried the maps for all the points at once (as vectorized input). Fig. 5.5 compares the computation times for both experiments.

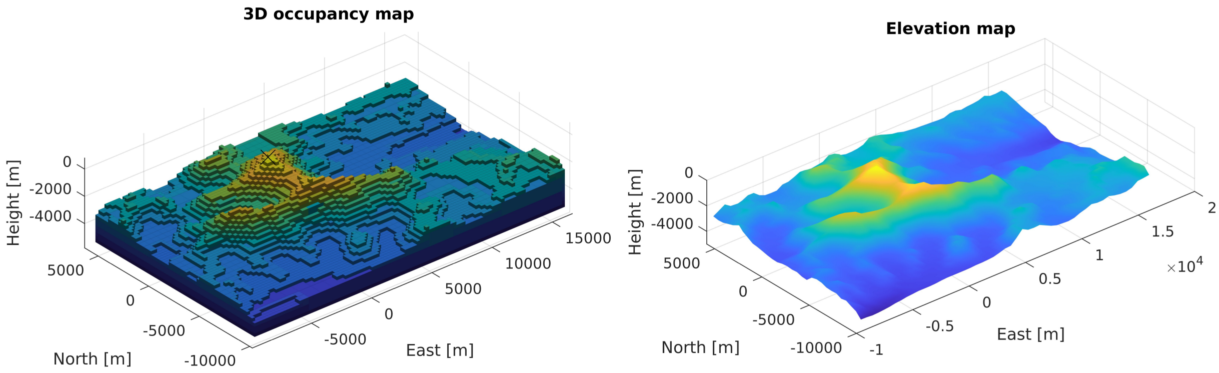

Fig. 5.4: Voxel-based occupancy map (left) and continuous elevation map (right)

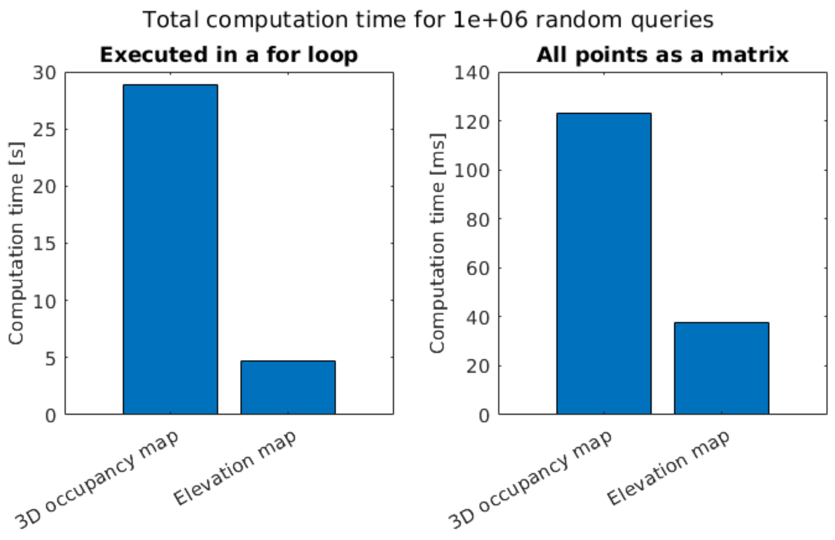

Fig. 5.5: Performance of the terrain maps: elevation-based (continuous) vs voxel-based

Using the simpler elevation-based model significantly reduced computation time. For point-by-point queries the total computation over 5 times shorter for the elevation map. Matrix-based queries resulted in smaller difference. Nonetheless, the queries to the voxel-based map still took about 3 times longer than for the elevation map.

5.4.2 Wind map

The wind map was implemented as a custom class in MATLAB R2021b. The class contained a four-dimensional lookup table and methods, which return interpolated and non-interpolated wind vectors given position and time as arguments. In this study wind is stored as two-dimensional vectors[3] with their vertical components discarded. Values between gridded samples are interpolated using 4D linear interpolation. The experiment is also described in [38].

The wind map was verified in a series of comparison tests. The measurements returned by the wind map were compared with data acquired manually from websites, which provide wind forecasts at different altitudes. Two weather models were considered: GFS (provided by Ventusky.com) and ECMFW (provided by Windy.com) [39]. Data for the wind maps was provided using mixed weather model by Meteomatics Weather API (https://www.meteomatics.com). Tab. 5.2 summarizes the results of the experiment.

The measurements were taken at several different points described by latitude, longitude and altitude above ground level (AGL), and time offset. Reference timestamp, where time offset is zero, was 17-03-2022 12:00 (UTC+1). The measurements 1 to 5 overlapped the wind map's sampling points, while the rest employed linear interpolation. The measured variables were the horizontal components of wind speed in m/s and its direction in degrees. The average relative error for wind speed was 22.09% for GFS and 16.63% for ECMFW. The average relative error for wind direction was 5.82% for GFS and 29.39% for ECMFW. Note that sometimes the reference models also return different results, e.g., speeds in the 9th row.

Tab. 5.2: Comparison of wind speeds and directions for different data providers [38]

#	Latitude	Longitude	Altitude	Time	Wind map		GFS		ECMFW	
	[°]	[°]	$[m]$	$[h]$	v $[\frac{m}{s}]$	ϕ [°]	v $[\frac{m}{s}]$	ϕ [°]	v $[\frac{m}{s}]$	ϕ [°]
1	50.2945	18.6714	2000	0	4.79	89.19	5	100	5	90
2	50.1022	18.5463	4200	+7	8.09	28.22	6	28	7	30
3	49.7716	19.2207	5500	+4	16.31	16.52	17	17	17	5
4	78.9235	11.9099	12000	+4	37.27	277.83	36	228	37	233
5	78.2232	15.6267	5500	+10	27.58	246.31	23	238	18	232
6	50.2945	18.6714	9000	+10	25.05	346.12	25	348	26	343
7	49.7228	19.2089	9000	+10	28.75	325.08	31	350	31	344
8	78.8147	12.8169	4200	+16	40.69	265.68	22	222	32	258
9	78.2669	15.6308	1500	+25	12.04	253.83	20	237	9	232

The wind map provided a compact and continuous model for predicting dynamic wind conditions. Based on the difference of predictions returned by the reference models themselves, the accuracy of the map itself is acceptable for flight planning and rough simulation. The error for

[3] Precisely, as pairs of horizontal wind speed component and horizontal wind direction, where direction of 0° means northerly wind (blowing from north to south) and 90° is easterly wind (blowing from east to west)

queried vs interpolated measurements is comparable. The efficiency of the map will be adressed in further research.

5.4.3 Kinematic guidance model

To simulate the UAV, different models of the controller and the plant can be used. By the time of writing the thesis, the full non-linear dynamic model of TS and its controller developed by other teams involved in the LEADED project had not been finished yet. Also, as mentioned in section1.4, providing such a model is beyond the scope of this thesis. Thus, to verify the performance of APP a simplified kinematic guidance model was employed as described in section 2.3.

Modeling the aircraft

To model the plant, a modified form of the kinematic guidance model from eq. 2.3 was used. The model was modified according to its MATLAB implementation described in [52]. Similarly to the eq. (2.3), the controlled variables are airspeed v_a^c, flight-path angle γ^c and roll angle ϕ^c. These variables are controlled using proportional controllers. It also implements a PD controller for roll angle ϕ. The modification imposes, however, some additional constraints depending on height and course angle. The corresponding equations of motion are

$$\begin{cases} \dot{x} = v_g \cos\chi \cos\gamma \\ \dot{y} = v_g \sin\chi \cos\gamma \\ \dot{h} = v_g \sin\gamma \\ \dot{\chi} = \dfrac{g\cos(\chi-\psi)}{v_g}\tan\phi \\ v_g \sin\gamma^c = \min\Big(\max\left(b_h\left(h^c - h\right), -v_g\right), v_g \Big) \\ \dot{\gamma} = b_\gamma\left(\gamma^c - \gamma\right) \\ \dot{v_a} = b_{v_a}\left({v_a}^c - v_a\right) \\ \dfrac{g\cos(\chi-\psi)}{v_g}\tan\phi^c = b_\chi\left(\chi^c - \chi\right) \\ \ddot{\phi} = b_{P\phi}\left(\phi^c - \phi\right) - b_{D\phi}\dot{\phi} \end{cases} \tag{5.9}$$

where x, y and h are coordinates of the UAV in $\mathcal{F}^i$. Variables $b_{P\phi}$ and $b_{D\phi}$ denote the coefficients of the proportional and derivative components of the roll PD controller. Conversion between v_a and v_g is done using the wing triangle, i.e., eq. (2.2).

Waypoint follower

LPP provides a set of waypoints that should be then followed by the actual aircraft. Due to stochastic nature of the real flight conditions, this waypoint-following behavior is assumed to be handled by the low-level controller. Therefore, it is outside of scope of APP presented in the thesis. However, simulating such a behavior is required to validate paths provided by LPP before flight. Hence, for simulation purposes the waypoint follower algorithm developed by Park et al. [49] is employed as the controller of the virtual UAV.

The waypoint follower utilizes nonlinear guidance logic. If following a straight path, it approximates the behavior of a PD controller. However, if a curved path is to be followed, the logic employs anticipatory control to assure tight path tracking. The controller computes commanded lateral acceleration using inertial speed, thus adding reactivity to external disturbances, such as wind [49]. The idea behind the waypoint follower is illustrated in Fig. 5.6.

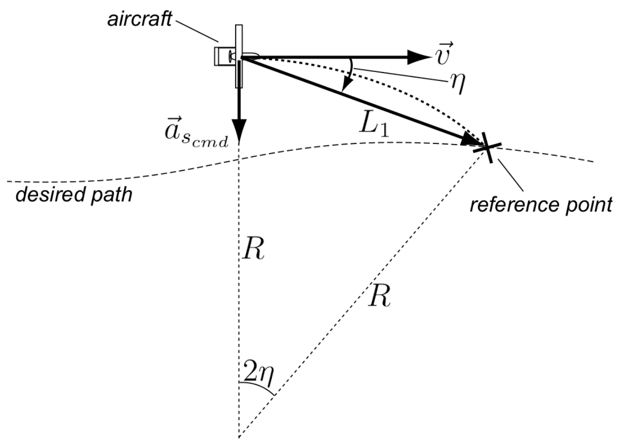

Fig. 5.6: Guidance logic of the waypoint follower [49]

First, the algorithm chooses a reference (way)point placed on the desired path at L_1 distance from the aircraft. Then, it commands the aircraft to smoothly turn towards the reference point by applying lateral acceleration $a_{s_{cmd}}$ given by

$$a_{s_{cmd}} = 2\frac{v^2}{L_1}\sin\eta = \frac{v^2}{R}$$

where v is the current velocity of the aircraft and η is the angle between $\vec{v}$ and L_1. The acceleration $a_{s_{cmd}}$ mimics centripetal acceleration of a point mass moving on a circle of radius R. The direction of $a_{s_{cmd}}$ depends on the sign of η. That is, the aircraft tries to align $\vec{v}$ with L_1. According to the authors, this guidance law is appropriate to follow any circular path [49].

A thorough discussion of the algorithm, including its linear analysis and verification study is found in [49]. Other approaches to the waypoint-following problem not covered here, specifically straight-line and orbit following, are also discussed in [20].

Calibrating the model

Before using it to verify APP, the model was re-calibrated. The kinematic constraints of the virtual fixed-wing UAV were modified to conform to the estimated constraints of the designed TS. The final kinematic constraints of the guidance model are summarized in Tab. 5.3. Note, that these are the “soft” limits proposed by the designers of TS to provide the optimal flight performance of the UAV, not the maximal allowable mechanically-imposed “hard” limits.

Tab. 5.3: Kinematic constraints of TS used by the guidance model

#	Symbol	Unit	Value	Description
1	γ_{min}	rad	$-\frac{\pi}{90}$	Flight-path angle limit when ascending
2	γ_{max}	rad	$\frac{\pi}{12}$	Flight-path angle limit when descending
3	θ_{max}	rad	$\frac{\pi}{30}$	Roll angle limit (symmetrical)
4	v_a	$\frac{\mathrm{m}}{\mathrm{s}}$	22	Optimal airspeed due to energy expenditure

The performance of the modified kinematic guidance model was verified afterwards to assure the controller is able to provide stable flight. The pre-configured values already implemented

Tab. 5.4: Default values of the controller parameters

#	Symbol	Value	Description
1	b_h	3.9	P-component of the height controller
2	b_γ	39	P-component of the flight-path angle controller
3	b_{v_a}	0	P-component of the airspeed controller, purposefully disabled to always maintain the optimal airspeed
4	b_χ	2	P-component of the heading angle controller
5	b_{P_θ}	40	P-component of the roll angle controller
6	b_{D_θ}	3.9	D-component of the roll angle controller

in MATLAB were used as the initial values of the controller parameters. These settings are presented in Tab. 5.4. See also eq. (5.9).

The model was then verified using an artificially generated helical path (Fig. 5.7). The path was defined by a set of 100 waypoints, which are indicated by circles. The first and last waypoints are shown as tetrahedral markers (red and green, respectively) to show the heading angle of the UAV. After that, a Dubins airplane path (red line) was generated connecting sequentially all the waypoints from start to goal. The path was subject to kinematic constraints of the UAV (as in Tab. 5.3) and was used as the reference afterwards. Length of the path was approximately 18837 m.

Finally, the kinematic guidance model calibrated as in Tabs. 5.3 and 5.4 was employed to simulate the flight through the waypoints . The simulated path is represented by the blue line in Fig. 5.7. Wind velocity was set to zero.

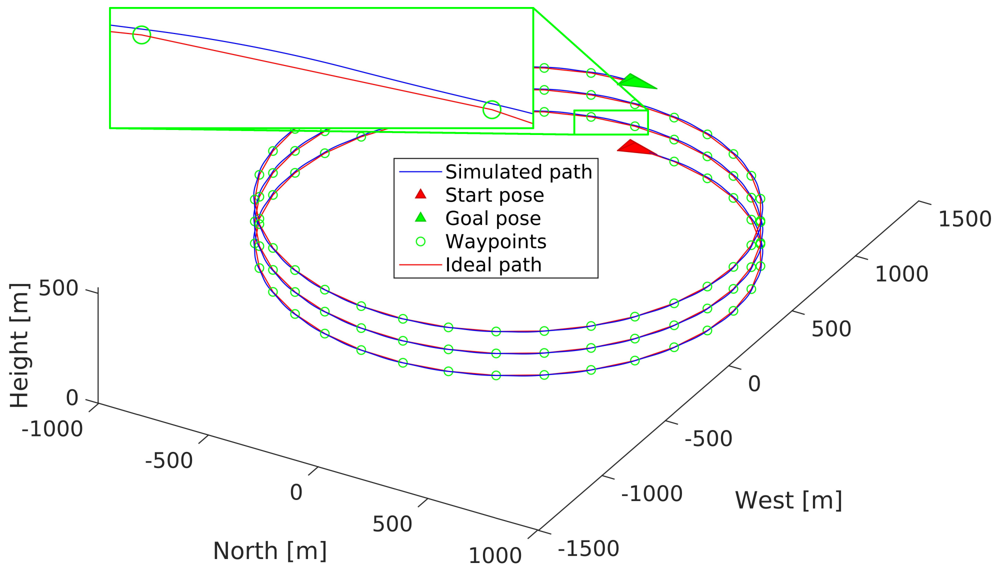

Fig. 5.7: Calibrated kinematic guidance model on a helical path

Fig. 5.8 illustrates position errors measured as the minimal 3D Euclidean distance between the simulated path and each waypoint. The mean error (red line) is 3.37 m (approximately 0.017% of the path length) with standard deviation of 0.40 m.

The same metric was applied to the simulated path vs ideal path. This time, the mean error is 4.88 m (0.026% of the path length). These errors are considered acceptable for pollution sampling missions. Hence, the kinematic guidance model calibrated as above is found feasible for APP validation.

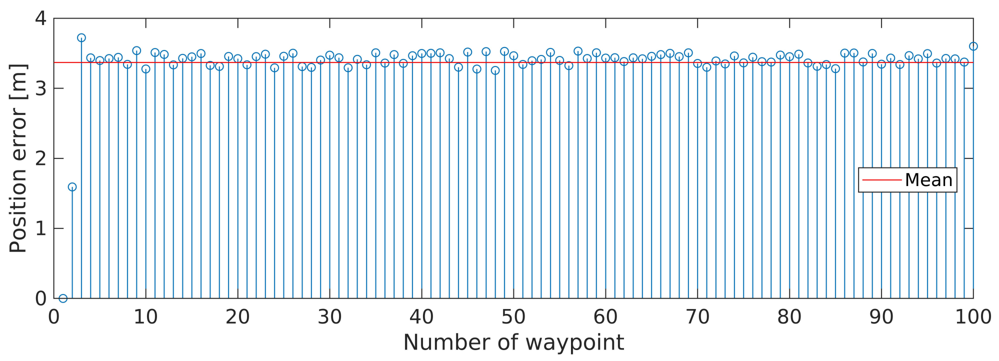

Fig. 5.8: Minimal distance measured from the simulated path to each waypoint

5.4.4 Discussion

The experiments shown that replacing a 3D occupancy map with a 2.5D elevation-based map significantly reduced computation time measured while querying the map. For the HALE UAV considered in the thesis the simplifications introduced by the elevation-based terrain map are acceptable. Moreover, reducing computation time is valuable, especially when the map will deployed on the target embedded hardware.

The differences between the measurements for queried sample points of the wind map and web servers (Tab. 5.2, rows 1 to 5) were comparable to the differences between the interpolated measurements (Tab. 5.2, rows 6 to 9) and web servers. The computation efficiency of the map requires, however, verification through dedicated research.

The kinematic guidance model implemented in MATLAB was found feasible for validating APP. However, if greater accuracy is expected, the full non-linear dynamic model should be used instead. As mentioned before, such a model is developed over the course of the LEADER project. Therefore, APP will be tested on a higher-fidelity aircraft model before Hardware-in-the-Loop (HIL) testing on the real TS begins. This model becomes indispensable for the validation of missions performed at higher altitudes.

5.5 Global Path Planner

GPP was validated on a finite set of abstract simplified mission scenarios. First, the effects of criteria were studied in-detail. Then, the weights of the criteria were optimized for a simple pollution sampling mission. Different optimization algorithms were then verified using this simplified scenario.

5.5.1 Effects of the criteria

First series experiments focused on validating the effects of each criterion independently. All the weights of the cost function except the one related to the tested criterion were zeroed. Then, the planning algorithm was employed to produce the final path. The path was then compared with a reference path without optimization.

The independent criteria tests shared the same configuration of GPP shown in Tab. 5.5. Parameters not specified here used the default values as implemented in MATLAB. I-GWO (see Algorithm 11) was used as the test algorithm for its relative simplicity, i.e., small number of configuration parameters, which could strongly affect the results. The tests used MATLAB I-GWO implementation by Mirjalili [53].

Tab. 5.5: Parameters of GPP during the criteria testing

#	**Parameter**	**Value**	**Unit**	**Description**
1	Airspeed	22	$\frac{\text{m}}{\text{s}}$	Constant UAV airspeed for defining Dubins paths
2	Flight-path angle limits	$\langle -\frac{\pi}{90}, \frac{\pi}{12} \rangle$	rad	Flight-path angle limit when ascending and descending, respectively
3	Max roll angle	$\frac{\pi}{30}$	rad	Symmetrical roll angle limit
4	Safety margin	100	m	Minimal distance to obstacles (terrain and airspace NFZ ROIs)
5	Algorithm	I-GWO	-	The algorithm used for the optimization of the mission parameters
6	Search agents	50	-	The number of potential solutions
7	Max iterations	30	-	The maximal number of iterations

Obstacle avoidance

Obstacle avoidance perhaps is the most important of the criteria, as failing to avoid an obstacle results not only in a failed mission, but also in a damaged, or even destroyed UAV. And this, in turn, poses a serious security risk. Therefore, this criterion was tested first. Obstacle avoidance functionality of GPP bases on two constraints:

- The path never goes below ground level defined by the terrain map $\mathcal{M}_t$.
- The path never crosses any NFZ ROI defined by the airspace map $\mathcal{M}_a$.

For safety reasons and to counter unpredictable events such as wind gusts, safety margin can be added, as seen in Tab. 5.5. To put an emphasis on both of the constraints, an artificial obstacle-dense map was created. The map used elevation map of Himalayas, around Mt. Everest. Although this area is not meant to be a target for measurement missions of the real UAV, it was chosen purposefully to provide a challenging environment with diverse elevation levels. Then, the map was filled with artificial obstacles using the airspace map's NFZs.

The map illustrating an example output of GPP while testing the obstacle avoidance criterion is shown in Fig. 5.9. The map shows isometric (upper pair) and top-down views (lower pair) of the generated path. The start and goal waypoints are depicted by red and green tetrahedrons, respectively. Control (optimized) waypoints are shown in yellow. Red transparent prisms depict NFZs. Regions with collisions are marked with green frames.

The figures on the left side depict the path generated with the obstacle avoidance criterion zeroed, which resulted in several collisions with NFZs and terrain. Activating the criterion successfully prevented both kind of collisions, providing a feasible path, as shown by the right pair. Note, only the collision avoidance was verified in this experiment, so the path is not optimized considering its length or complexity, for example.

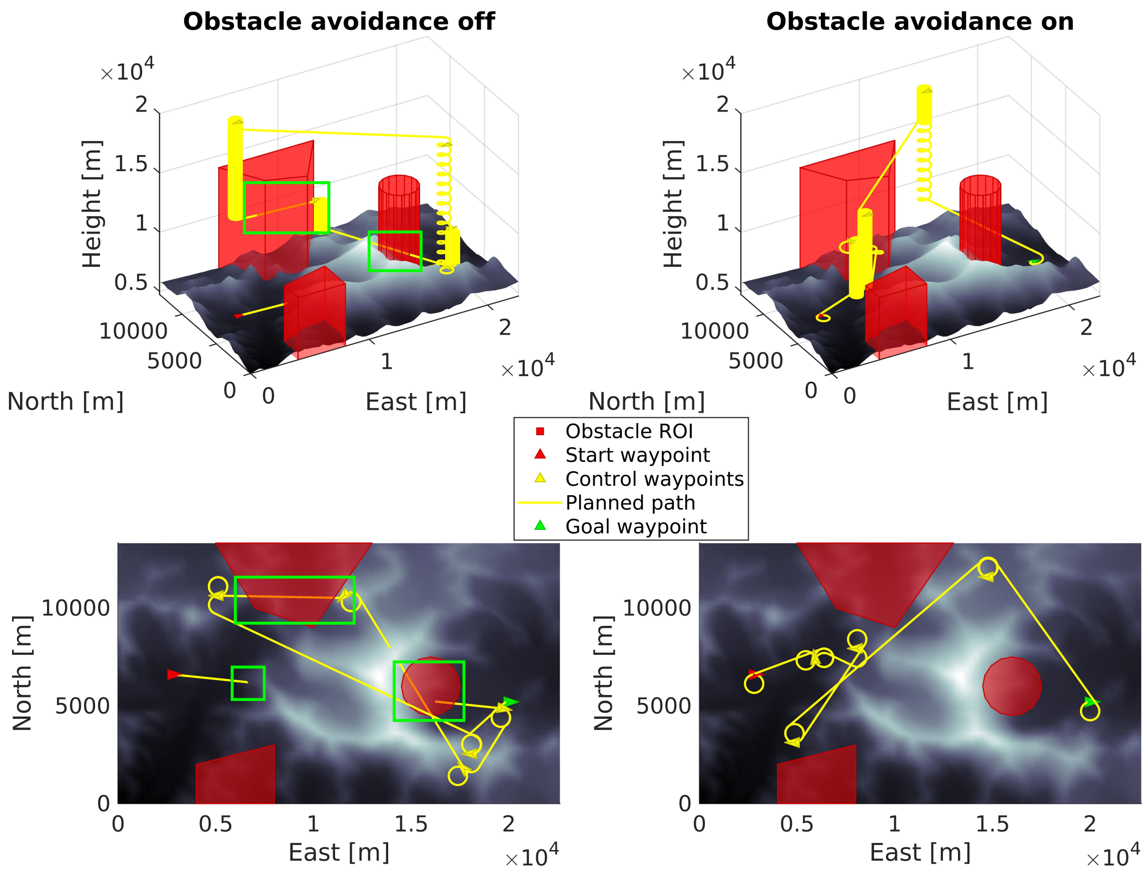

Fig. 5.9: Planned path with the obstacle avoidance criterion off (left) and on (right)

Minimal path length

The second criterion minimizes the total length of the resulting Dubins path. Minimal length contributes to reducing energy expenditure, mission duration and overall path complexity. This criterion was verified using an empty map, which required no additional maneuvers to avoid potential obstacles. The results are shown in Fig. 5.10. Again, figures on the left depict the reference path and the ones on the right show the path with the positive weight of the criterion. The upper figures show front view of the scene, while the bottom figures present top-down view. The meaning of the graphics remains the same as above.

It can be observed the criterion contributes to reduction of path length, especially minimizing changes in altitude. Seemingly unnecessary maneuvers shown in the right figures are used to gain height between the start and goal waypoints. Connecting the waypoints with a simpler path is not possible because of the kinematic constraints of the UAV[4].

[4] That is, the path would be too steep, thus violating the flight-path angle constraint. Hence, it is required to gain height over a greater distance, which explains additional maneuvers performed by the aircraft.

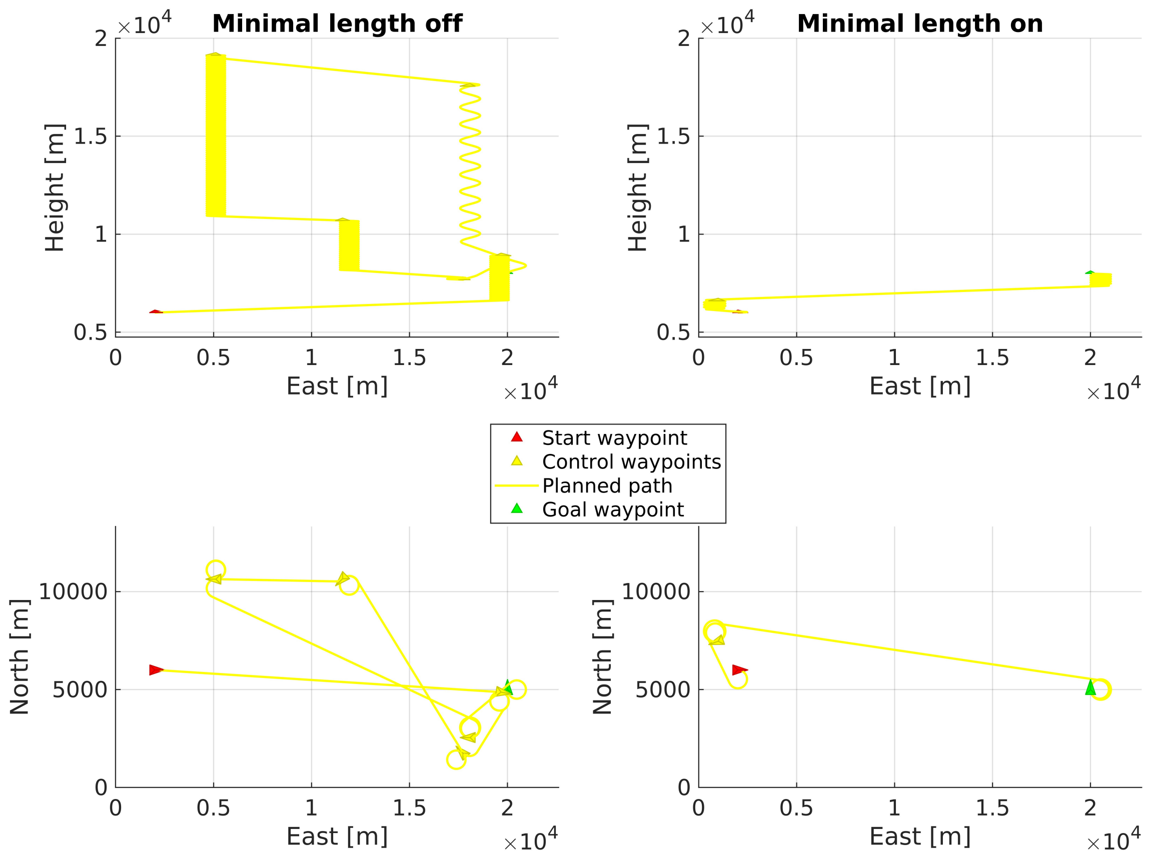

Fig. 5.10: Planned path with the minimal length criterion off (left) and on (right)

Wind influence

When considering wind, GPP optimizes the path to maximize benefits the current wind conditions has on the UAV, while reducing its negative implications. Fig. 5.11 shows the effects of the wind criterion on the planned path. The upper figure shows a reference path with the wind influence criterion turned off, while the lower one illustrates the wind-optimized path. Wind is shown as three-dimensional vector field, where the length of the vectors as well as their color indicate wind speed. Wind speed in meters per second can also be read on the colorbar.

Most of the time strong wind affects the UAV negatively by increasing the path cost (see Fig. 4.3 to 4.2). Hence, GPP optimizes the path to stretch at lower altitudes, where the resultant effect of wind is less severe.

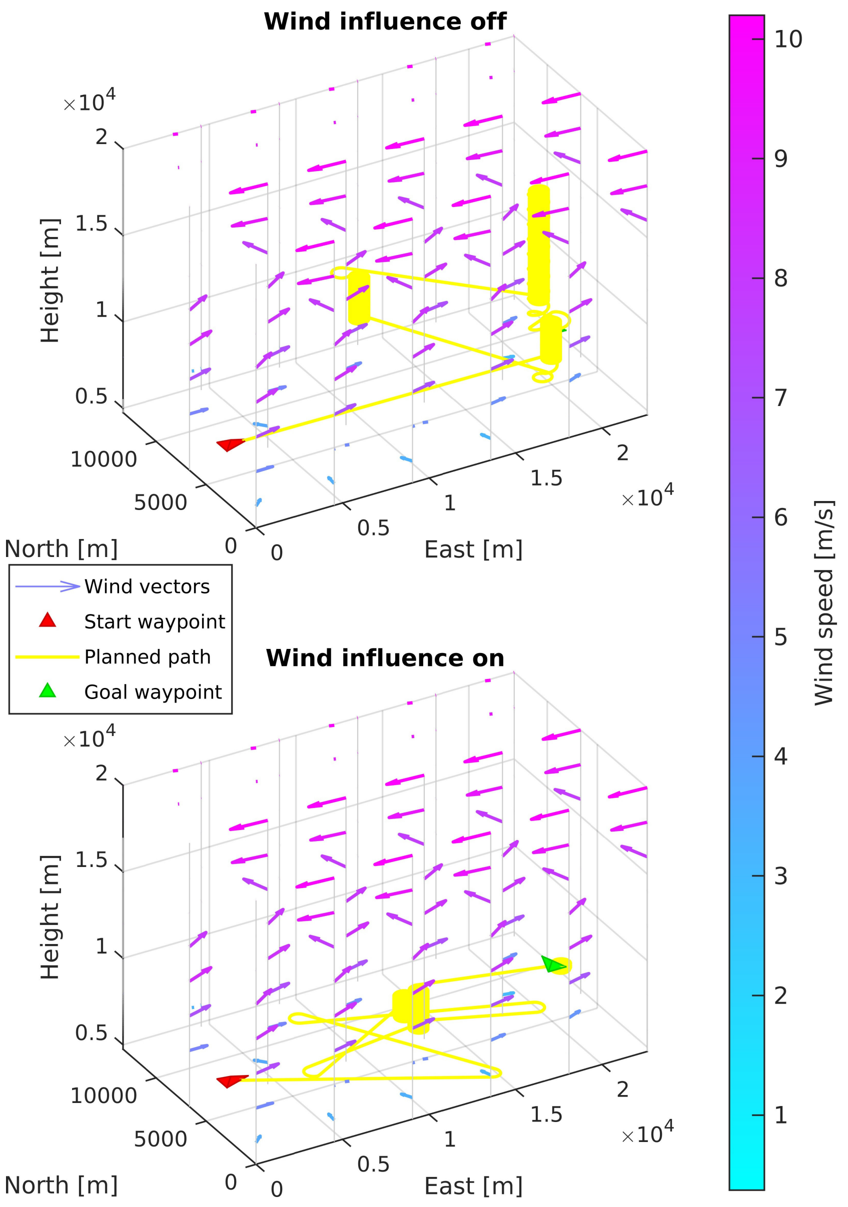

Fig. 5.11: Planned path with the wind influence criterion off (top) and on (bottom)

Energy expenditure

From the practical point of view, the minimal energy expenditure criterion acts similarly to the minimal path length criterion described above. The key difference is the energy criterion highlights the impact of altitude. In other words, the length criterion penalizes the planner evenly in each dimension. The energy criterion focuses on altitude changes and increases the cost significantly if the UAV climbs up following a steep trajectory, but the cost of maintaining or reducing height is relatively small. Moreover, this criterion detects if the path requires more energy than the UAV can afford. In that case, the cost value is severely increased. The minimal length criterion boasts significantly shorter computation time, though.

Fig. 5.12 presents an example paths with the criterion activated (right figures) and off (left figures). The minimal energy expenditure criterion visibly reduces altitude changes. Nevertheless, horizontal movement is less strict than in the case of the minimal length criterion.

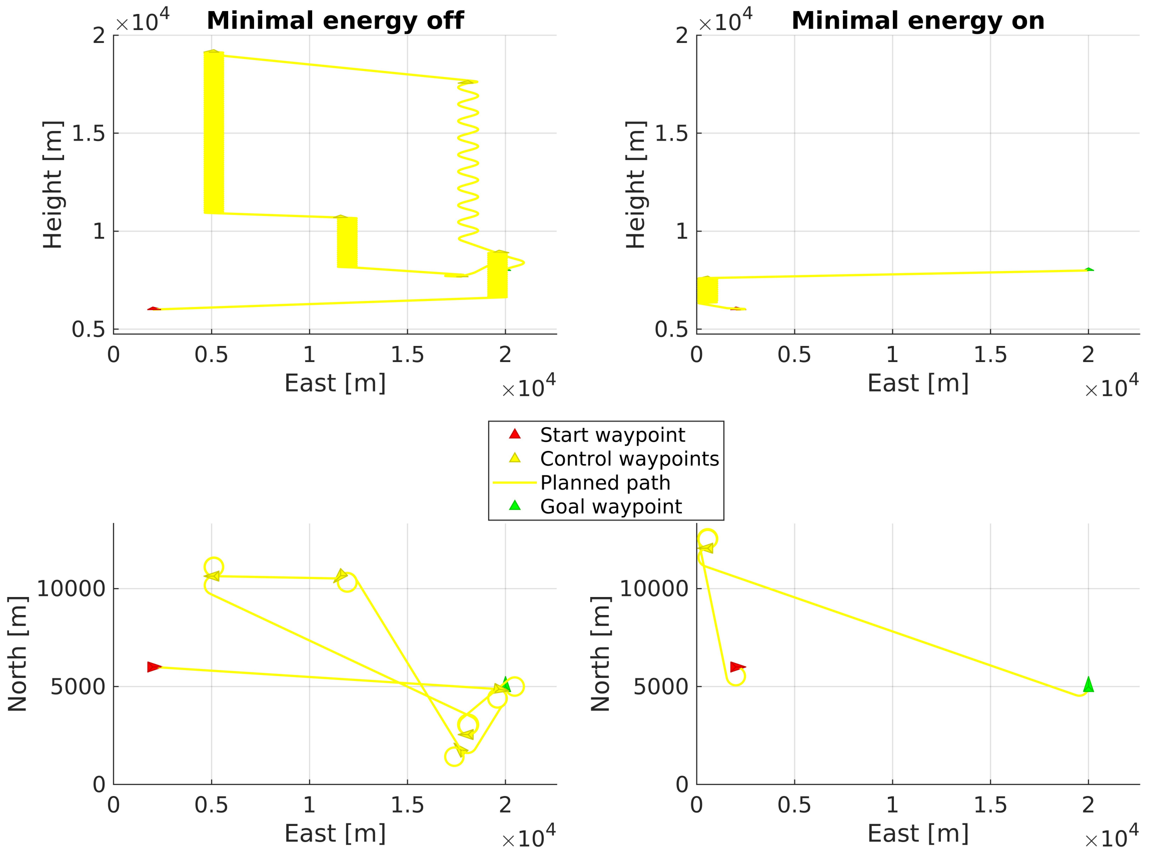

Fig. 5.12: Planned path with the minimal energy criterion off (left) and on (right)

Pollutant level

The last criterion optimizes the placement of the area, that will be covered by the measurement path. The geometric center of the measurement polygon should be placed at the point of highest pollutant concentration.

Fig. 5.13 illustrates an example of non-opitimized (left figures) and optimized (right figures) flight paths according to the pollutant level criterion. The center of the measurement region is placed roughly in the most polluted region on the map. The left figures present the alternative behavior of GPP, which does not generate a measurement path if this criterion is zeroed.

5.5.2 Comparison of the chosen single-objective optimization methods

The next series of experiment compares the performance of different flavors of single-objective optimization algorithms used for global planning. The mission involves solving the optimization problem (4.4) while flying in a relatively complex scene. Similarly as before, the map provides additional challenge for GPP not present in regular measurement missions. Tab. 5.6 presents the detailed overview of the mission.

Calibrating the optimization algorithms

Whenever possible, the algorithms used their default settings as provided by their corresponding literature: [54, 55] for I-GWO, [47, 56] for ACO$_{\mathbb{R}}$, [57] for PSO and [58] for GA. Some settings required modifications, which were done empirically by trial and error on a sample map until the algorithm started to converge to the optimal solution. The algorithms were tested using a balanced criteria weights, i.e., each of the 5 criteria has its weight set to 0.2. Some of the final

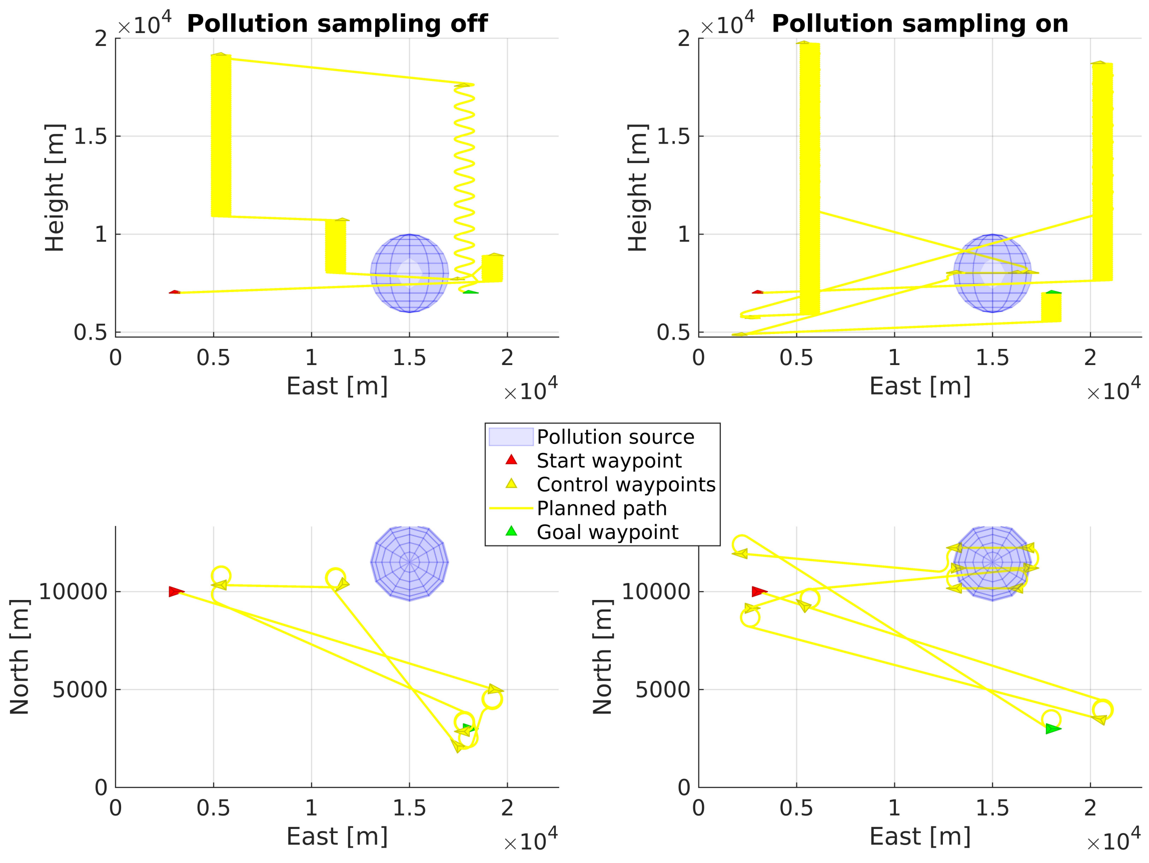

Fig. 5.13: Planned path with the pollutant level criterion off (left) and on (right)

settings of the algorithms are summarized in Tab. 5.7. The values that differ from their defaults are written in bold.

Comparative analysis

After calibrating each algorithm, the comparative study on the performance of the algorithms was conducted. They were evaluated on a series of 10 test cases, each repeated 5 times with different random seed. The analysis covered thus a total of 200 test cases with 50 evaluations of each algorithms. The cases shared the same terrain map, wind map and airspace map. The start and goal waypoints, as well as the placement of pollution source were adjusted manually and varied between the cases. These experiments employed the explicit pollution model (3.14). Pollution-induced cost increased linearly from the point of maximum pollution intensity. All five criteria as in Tab. 4.1 were active and weighed 0.2 each. The intensity of each criterion was adjusted empirically via their scaling factors s_f to provide intuitively-correct cost.

The results for the first run and the first test case are illustrated in Fig. 5.14 and described quantitatively in Tab. 5.8. Fig. 5.14 shows four complete global paths as colored lines. The red and green tetrahedral markers denote the start and goal waypoints, respectively. Wind is given as a colored vector field. The NFZs are illustrated as transparent red prisms. The blue sphere denotes the region with the highest pollution intensity. Measurement path (zigzag pattern) is visibly shifted south from the point of highest pollutant concentration. It is caused by applying a safety margin to the map boundaries and does not indicate improper behavior of the algorithm.

In this example all paths are feasible, i.e., no collisions occur. The measurement paths generated by the algorithms lie close to the maximally polluted region. Nevertheless, as noted in Tab. 5.8, the scores of the algorithms differ significantly. The best path (according to COST)

Tab. 5.6: Detailed scenario of a sample measurement mission

Scenario ID	AlgorithmsComparisonGPP
Description	Horizontal zigzag pollution profiling
Goal	GPP plans an optimized measurement path for a HALE UAV. The planner optimizes the placement of the measurement path to cover the horizontal profile with the highest concentration of the pollutant. The UAV flies from the preset start waypoint to the measurement area using a path optimized according to the minimal energy expenditure and wind conditions. After taking the measurements, the UAV flies a similarly planned path to the goal waypoint, which indicates its landing approach point.
Stages	1. The aircraft is airborne at the start waypoint, airspeed = 22 $\frac{\mathrm{m}}{\mathrm{s}}$. 2. The UAV flies to the region of the maximum pollutant concentration. 3. The UAV performs the measurement following a zigzag-shaped path. 4. The aircraft flies towards the goal waypoint, where the mission ends.
Map	Rectangle from (27.930000°N, 86.790000°E) to (28.050000°N, 87.020000°E), local origin $(0, 0, 0)$ at (27.930000°N, 86.790000°E, 0 m AMSL), wind forecast from t_0 to $t_0 + 12$ h
Algorithms	I-GWO, PSO, GA or $\mathrm{ACO}_{\mathbb{R}}$
Mission parameters	Kinematic constraints of the UAV as in Tab.5.5 Position of start and goal waypoints varies between the cases
Pollution model	Explicit linear as in eq. 3.14, cost growth rate = 0.005 $\mathrm{m}^{-}1$ from center Position of the center varies between the cases

was generated by $\mathrm{ACO}_{\mathbb{R}}$. This observation is supported by LEN, NPTS and EEE metrics designating this path as the shortest and the most energy-efficient[5]. Interestingly, $\mathrm{ACO}_{\mathbb{R}}$ achieved also the shortest CT, despite I-GWO and PSO having significantly less CFE. Compared to the other algorithms, $\mathrm{ACO}_{\mathbb{R}}$ converged faster to a relatively short path. Hence, the following computations of the cost function resulted in significant savings in CT, despite high CFE.

Tab. 5.9 displays basic statistical measures of the metrics computed for the first case and all runs of the algorithms. Fig. 5.15 provides a quick visual reference of path variability in each algorithm between consequent runs. Note that contrary to Fig. 5.14, colored lines indicate runs of the same algorithm, not different algorithms.

Most observations from the first run (Tab. 5.14) holds true for other algorithms runs. Paths generated by $\mathrm{ACO}_{\mathbb{R}}$ have unmatched quality (LEN, SMOO, NPTS and EEE). $\mathrm{ACO}_{\mathbb{R}}$ is inferior to I-GWO and PSO only in CFE. Nevertheless, it still achieves the lowest CT for the reason explained above. Moreover, the algorithm provides the most repeatable results with the lowest standard deviation for many metrics (COST, LEN, NPTS and EEE).

I-GWO features the lowest LEN and CFE. Nevertheless, it provides paths with the highest COST and requires the most CT to compute them. PSO and GA achieve comparable results with the former being faster (lower CT and CFE), while the latter being more reliable (lower standard deviations of most metrics).

The first test case is described in-detail as an example. To actually evaluate the general

[5] Simulation phase is skipped in this experiment. Therefore, EEE is calculated with zero wind assumption, as in the energy expenditure criterion ($v_g = v_a$). To provide more accurate results, EEE must be calculated for simulation data.

Tab. 5.7: Chosen settings of the algorithms

	Parameter name	**Value**	**Description**
I-GWO	Max iterations	**100**	Maximum number of iterations if not stalled
	Search agents	100	Number of potential solutions (wolves)
	Max stall iterations	**15**	Maximum number of consequent iterations without finding a better solution
	Function tolerance	**1**	The smallest change in cost that is not considered a stall
ACO$_\mathbb{R}$	Max iterations	**300**	Maximum number of iterations if not stalled
	Archive size	**50**	Number of the best solutions (ants) from the previous iteration
	Sample size	**300**	Number of solutions computed each iteration
	Intensification factor	0.001	The likelihood of chosing the best solution to generate new solutions during an update
	Deviation distance ratio	0.85	How far the new solutions can deviate from their source
	Max stall iterations	**15**	Maximum number of consequent iterations without finding a better solution
	Function tolerance	**1**	The smallest change in cost that is not considered a stall
PSO	Max iterations	**150**	Maximum number of iterations if not stalled
	Swarm size	**400**	Number of particles in the swarm
	Social adjustment weight	1.49	Weighting of the neighborhood's best position when adjusting velocity
	Self adjustment weight	1.49	Weighting of each particle's best position when adjusting velocity
	Max stall iterations	**25**	Maximum number of consequent iterations without finding a better solution
	Min neighbors fraction	**0.5**	Minimum adaptive neighborhood size
	Function tolerance	**1**	The smallest change in cost that is not considered a stall
GA	Max generations	3700	Maximum number of generations if not stalled
	Population size	**1500**	Number of population members
	Crossover fraction	0.8	How many non-elite children created at the next generation
	Crossover ratio	1.2	How far the child is from the better parent in the heuristic crossover function
	Stall gen limit	**20**	Maximum number of consequent generations without finding a better solution
	Function tolerance	**1**	The smallest change in cost that is not considered a stall

Tab. 5.8: Detailed metrics for the 1st test case and 1st run

Algorithm	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
I-GWO	189.205	9300	7036.044	124789	0.754	1251	1.064	0
ACO$_{\mathbb{R}}$	109.428	25850	648.173	93424	0.817	938	0.600	0
PSO	124.214	16800	3822.815	237134	0.717	2373	4.258	0
GA	258.184	31500	2607.089	195170	0.726	1955	2.997	0

Tab. 5.9: Basic statistical measures for the 1st test case and all 5 runs)

I-GWO								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	252.596	12740	5351.806	132764	0.754	1331	1.171	0
StDev	65.089	4047	2025.959	10265	0.004	103	0.159	0
Min	189.205	9300	1654.473	123761	0.748	1240	1.000	0
Max	366.800	20100	7036.044	147152	0.761	1474	1.367	0
Median	255.373	11100	6266.195	124789	0.754	1251	1.064	0
ACO$_{\mathbb{R}}$								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	98.591	21350	645.898	93431	0.805	938	0.600	0
StDev	8.851	3823	2.413	14	0.022	0	0.000	0
Min	91.010	16850	641.955	93422	0.770	937	0.600	0
Max	109.428	25850	648.173	93458	0.831	938	0.600	0
Median	91.678	20150	646.829	93424	0.817	938	0.600	0
PSO								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	130.876	18560	2646.573	189100	0.742	1893	2.885	0
StDev	6.761	1651	1161.778	50969	0.031	509	1.347	0
Min	124.214	16800	669.976	97770	0.717	981	0.606	0
Max	141.255	21200	3822.815	237134	0.801	2373	4.258	0
Median	128.627	18800	2503.187	190055	0.728	1904	2.655	0
GA								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	227.780	32070	2228.843	175115	0.725	1754	2.491	0
StDev	35.827	1140	360.917	18078	0.005	181	0.494	0
Min	171.256	31500	1638.002	147969	0.721	1483	1.663	0
Max	258.184	34350	2607.089	195170	0.733	1955	2.997	0
Median	251.762	31500	2180.302	170733	0.726	1710	2.453	0

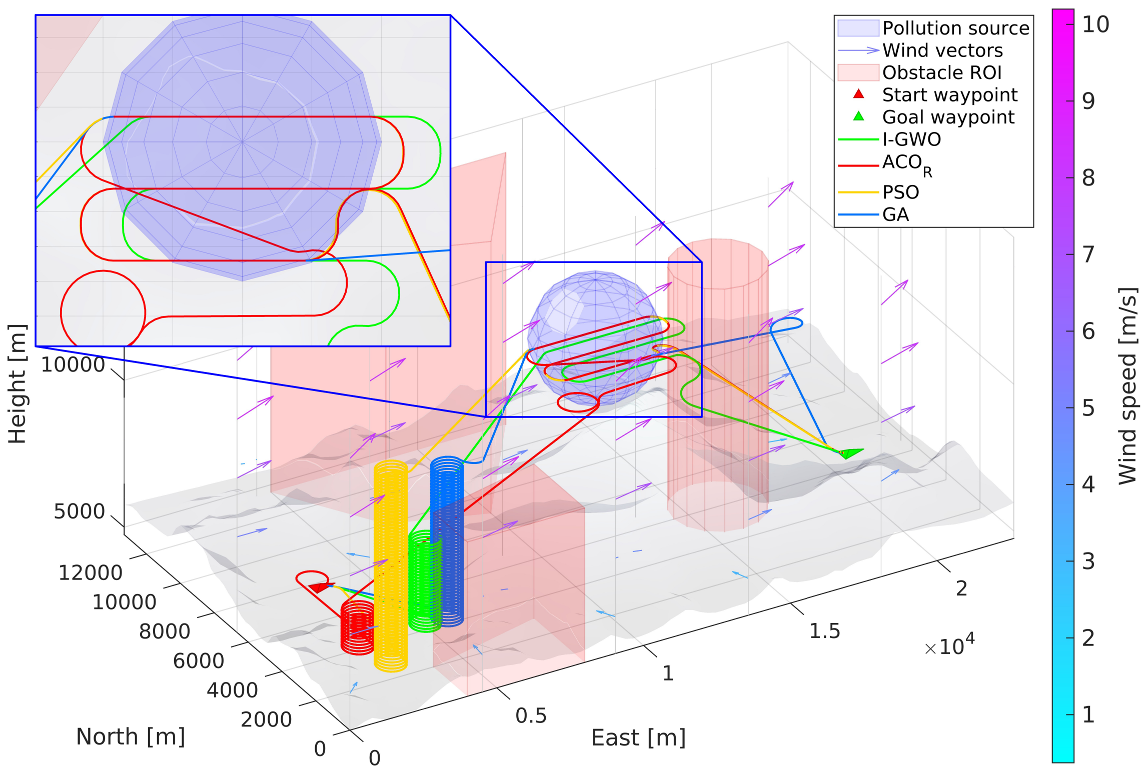

Fig. 5.14: Optimized paths generated by different algorithms in a single run (1st case)

performance of the algorithms, similar experiments were conducted for the rest of test cases. The results are summarized in Tab. 5.10.

Interestingly, while comparing the medians of all the cases, $ACO_{\mathbb{R}}$ still holds its leadership. Nevertheless, the algorithm is also the least consistent (highest standard deviation) in terms of CT, CFE, COST and SMOO. However, 75% of the paths still provide solid and dependable results. Based on repeatability, GA is the winner with the lowest standard deviation in 4 of 8 metrics, most notably CFE and COST. Despite that, it has the second-worst CT, LEN and EEE, surpassed only by I-GWO.

Choosing the most suitable algorithm, based on the metrics from Tab. 5.10, is a multi-criteria optimization problem. To solve the problem, a point-based optimization method was employed. The medians from Tab. 5.10 were evaluated by assigning weights to them according to their importance for GPP. The least important metric has weight of 1, the second-least important has 2 etc. NCOL hase been removed from the evaluation as no collisions occurred during the experiment. EEE is highly positively correlated to NPTS and LEN (0.990 for both[6]). Hence, NPTS and LEN were not considered in optimization.

COST correlates with many crucial mission parameters, so it is the most important metric. Then, EEE and SMOO were chosen. The first one positively correlates to small changes in altitude[7], while the second one also reflects the complexity of horizontal maneuvers. They were followed by computation performance metrics, specifically CT and CFE. The performance metrics were placed at the end because GPP is the highest-level planner and does not have to follow strict timing regime.

[6] NPTS is computed by sampling the resultant Dubins path. As the sampling step is constant, correlation of NPTS and LEN is 1, so both metrics convey the same information. They were left in comparison tables for convenience – to show the actual length and also the complexity of the path.

[7] The current implementation of energy expenditure does not consider energy expanded on horizontal maneuvers, see eq. 4.13.

Tab. 5.10: Basic statistical measures for the whole experiment, 50 samples per algorithm

I-GWO								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	239.818	11804	5332.872	178701	0.750	1791	2.517	0
StDev	95.159	5229	3404.326	67139	0.025	672	1.792	0
Min	84.167	3700	797.913	98243	0.705	986	0.607	0
Max	417.574	20100	13921.534	343875	0.800	3443	7.031	0
1st quartile	177.420	8900	3083.056	126182	0.728	1265	1.138	0
Median	232.707	11100	4695.423	159921	0.750	1604	1.771	0
3rd quartile	307.695	13900	7036.044	218643	0.762	2190	3.763	0

ACO$_{\mathbb{R}}$								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	171.881	33932	6271.053	120938	0.792	1213	1.121	0
StDev	139.482	23494	17800.854	52168	0.035	522	1.205	0
Min	47.071	11150	374.211	75409	0.717	757	0.286	0
Max	719.909	90050	111199.651	317829	0.861	3184	5.364	0
1st quartile	75.746	15950	546.130	90619	0.763	910	0.429	0
Median	118.626	24200	727.699	101784	0.792	1023	0.619	0
3rd quartile	220.260	45350	1530.986	129861	0.817	1301	1.313	0

PSO								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	136.278	21688	2481.475	119923	0.783	1203	1.073	0
StDev	35.723	4570	6914.403	40466	0.030	404	1.012	0
Min	59.430	10800	382.175	79731	0.714	801	0.273	0
Max	224.549	32000	36092.829	281648	0.849	2820	4.994	0
1st quartile	107.395	18800	546.170	95966	0.761	963	0.477	0
Median	138.544	21800	733.795	108130	0.788	1085	0.629	0
3rd quartile	157.436	24800	1220.731	123539	0.803	1239	1.313	0

GA								
Metric	CT [s]	CFE	COST	LEN [m]	SMOO	NPTS	EEE [kWh]	NCOL
Mean	201.779	32669	2095.801	164967	0.748	1653	2.187	0
StDev	30.020	1989	864.810	43910	0.023	439	1.077	0
Min	145.928	31500	439.135	76622	0.719	770	0.363	0
Max	273.989	38625	4374.112	272821	0.828	2733	4.825	0
1st quartile	181.028	31500	1459.975	132111	0.727	1324	1.353	0
Median	199.026	31500	1991.016	164131	0.744	1645	2.201	0
3rd quartile	217.777	32925	2607.089	190135	0.766	1903	2.991	0

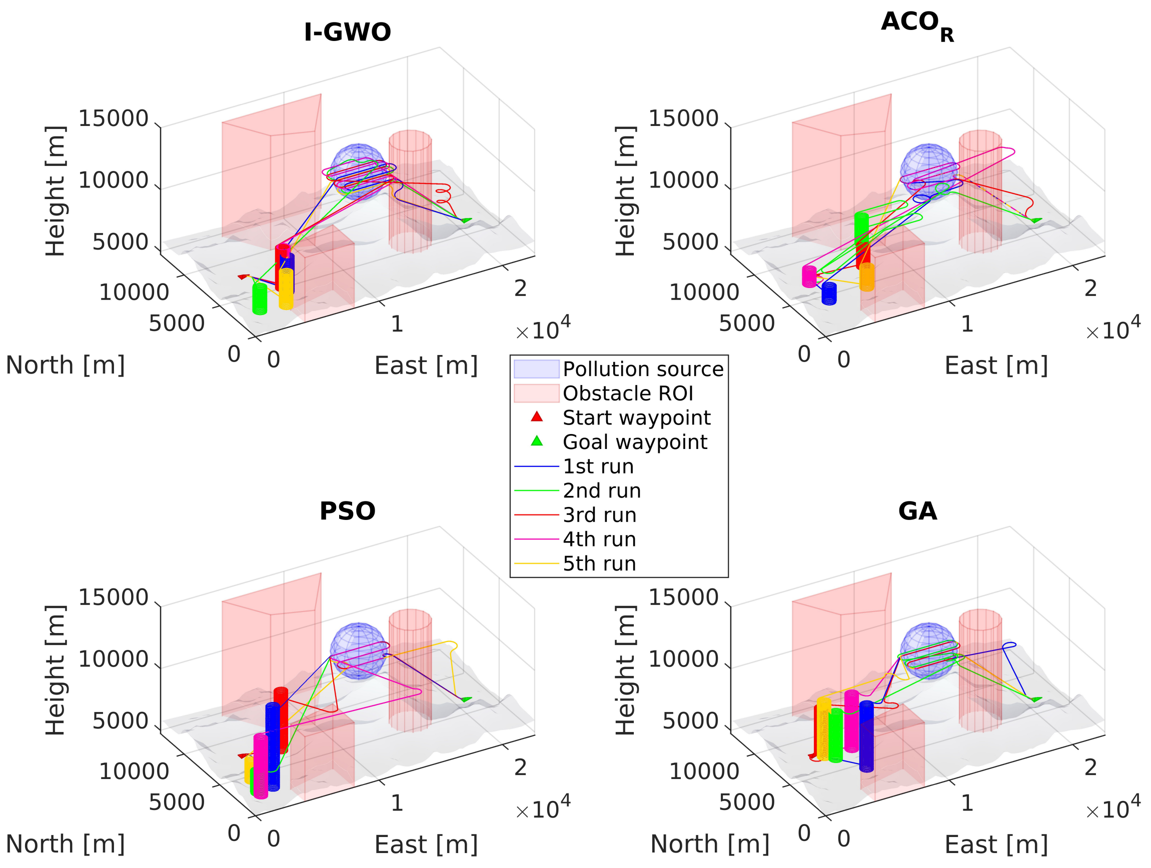

Fig. 5.15: Variability of the optimized paths in multiple runs (1st case)

It was decided that the variability of the results should be incorporated into the optimization. However, as the results of GPP are validated in simulation and then approved by a human operator, the optimization favored medians over standard deviations. The importance of metrics was kept as above, but standard deviations were assigned their weights after medians.

After deciding on the order of importance of metrics, each algorithm was evaluated by assigning a point-based grade to each of its metrics as below:

- 3 points for the first (best) mean result for a given metric[8].
- 2 point for the second.
- 1 point for the third.
- 0 points for the last (worst) result.

Next, computing the weighted sum of the metrics weights and their grades provided the final quantitative evaluation results of the comparison. For that, a simple weighted sum was employed, i.e.

$$G^f = \sum_{n=1}^{N} \omega_n G_n \tag{5.10}$$

where G^f is the final grade in points, ω_n is the weight of n-th metric and G_n is the grade of the n-th metric in points. N denotes the total number of the criteria (metrics). The results are presented in Tab. 5.11. According to the criteria defined above, the optimal algorithm for GPP

[8] It means the highest (e.g. SMOO) or the lowest (e.g. LEN) mean value for each algorithm, depending on the metric.

is $ACO_{\mathbb{R}}$ and PSO is the second-best. Interestingly, GA boasts the highest grades in terms of repeatability, so it would be the best algorithm if the evaluation phase is discarded.

Tab. 5.11: Point-based multi-criteria optimization table of GPP algorithms

	Metric	**Weight**	**Grade**				**Grade × Weight**			
			I-GWO	$ACO_{\mathbb{R}}$	PSO	GA	I-GWO	$ACO_{\mathbb{R}}$	PSO	GA
Median	CT	7	0	3	2	1	0	21	14	7
	CFE	6	3	1	2	0	18	6	12	0
	COST	10	0	3	2	1	0	30	20	10
	SMOO	8	1	3	2	0	8	24	16	0
	EEE	9	1	3	2	0	9	27	18	0
StDev	CT	2	0	1	2	3	0	2	4	6
	CFE	1	1	0	2	3	1	0	2	3
	COST	5	2	0	1	3	10	0	5	15
	SMOO	3	2	0	1	3	6	0	3	9
	EEE	4	0	1	3	2	0	4	12	8
			Weighted sum				52	114	106	58
			Max points				165	165	165	165
			Fitness of solution				31.52%	**69.09%**	64.24%	35.15%

5.5.3 Discussion

GPP was thoroughly tested first by verifying the effects of the optimization criteria, and then by comparing the performance of several optimization algorithms. The criteria successfully modified the generated path according to their definitions given in section 4.4.2.

Comparative analysis of the chosen optimization algorithms highlighted $ACO_{\mathbb{R}}$ and PSO as the potential winners. While $ACO_{\mathbb{R}}$ beats the others in terms of path quality, PSO gets consistently high grades for medians and standard deviations as well. Interestingly, if the optimization is based on standard deviations (they are rated higher than medians), GA emerges as the leader. GA is more robust and provides stable paths between consequent runs compared to the others. Depending on the actual goal, any of these can be chosen as the optimal one.

However, GPP in the thesis is employed primarily to support the human operator, not to entirely replace them. The results will be validated in simulation and must be manually approved by the operator prior to sending the path to the UAV. Hence, rejecting a small fraction of sub-optimal paths is not an issue, and it is worth the potential benefits in path quality. Therefore, $ACO_{\mathbb{R}}$ is the optimal solution for GPP acting as a mission planning tool and will be used for the final APP tests described in the thesis. Nevertheless, if GPP is run without human supervision, GA becomes the preferred choice due to its robustness. PSO, being an all-rounder, is also an interesting choice for the future research. I-GWO was found inferior to the competition in most of the aspects.

5.6 Local Path Planner

LPP was validated on a finite set of abstract simplified mission scenarios. The RRT algorithms used by LPP are stochastic and depend heavily on the random number generator (RNG) and its actual seed. Hence, the results are expected to have significant standard deviation between the experiments. To reduce the influence of randomness, each experiment was repeated several times using different initial conditions.

Randomizing test waypoints

The waypoints used for the elementary collision avoidance tests were randomized. The waypoint coordinates along x, y and z (NED), as well as ψ angle (defined in $\mathcal{F}^v$ frame) were randomly generated using uniform distribution. Uniform distribution was employed on per-axis basis. That is, the coordinates are uniformly distributed along each axis separately, not uniformly distributed in 3D space.

The ψ angle is required as the initial condition for LPP. Outside simulation, the current yaw of the UAV would be used. Next, the waypoints were grouped in pairs and optimized. The optimization problem was to keep approximately constant distance between the waypoints in a pair, that is

$$\begin{aligned}&\text{minimize } \mathbf{C}(\mathbf{W}^S, \mathbf{W}^G, d) = \sum_{n=1}^{N} \left| d - \left\| \overline{\mathbf{w}_n^S \mathbf{w}_n^G} \right\| \right|, \qquad \mathbf{w}_n^S \in \mathbf{W}^S, \qquad \mathbf{w}_n^G \in \mathbf{W}^G \\ &\text{subject to } \Omega(\mathbf{W}^S, \mathbf{W}^G)\end{aligned}$$

where $\mathbf{W}^S$ and $\mathbf{W}^G$ each contain N start and goal waypoints, respectively. Distance d is the expected constant distance between the waypoints in each pair, while $\Omega(\cdot)$ represents the constraints, which restrict the waypoints to be placed above terrain.

5.6.1 Test setting

The performance of LPP was first verified by supplying an environment map consisting of terrain and airspace data (Fig. 5.16). The continuous elevation model (3.3) with linear interpolation was used as the terrain map model. The terrain map used the elevation data of Mt. Everest and its surroundings. That setting was chosen for its diverse elevation levels over a relatively small area. The airspace map was filled with abstract obstacles meant to reflect law-enforced no-fly zones NFZs, or otherwise dangerous conditions (e.g., clouds). Note, it is only an abstract test case to provide a challenge for the planner. The UAV considered in the thesis is not meant to fly over such terrain.

For the experiment, 10 randomly distributed pairs of waypoints were generated, as seen in Fig. 5.16). Each waypoint is illustrated as a tetrahedral marker, indicating the initial and the final ψ orientation of the UAV. Red markers represent start waypoints, while green ones are goal waypoints. Each pair of waypoints is connected with a colored line. The lines are used only for presentation purposes and are ignored by LPP. The height of the waypoints was limited to stay below the highest point of the map, i.e, the peak of Mt. Everest. This was done to encouraged LPP to generate the paths around or over terrain obstacles. The distance criterion was set to 10 km.

The final parameters of the waypoint pairs are given in Tab. 5.12. The poses are defined in local ENU frame, where x, y and z axes are aligned with East, North and Up (Height), respectively. Yaw angle ψ is given in $\mathcal{F}^v$. For brevity, positions are rounded to integers. Variable d_f denotes the final distance between the waypoints in a pair (after optimization).

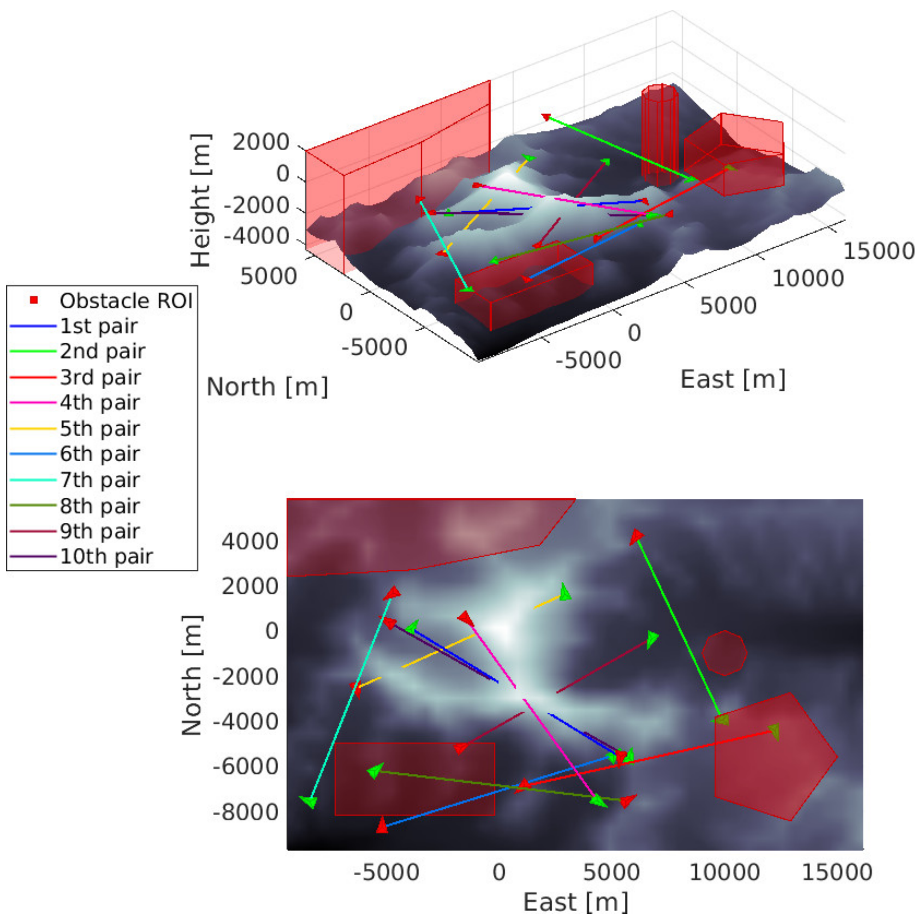

Fig. 5.16: Randomly placed and optimized waypoint pairs for LPP verification

5.6.2 Implementation of the LPP algorithm

LPP will be discussed step-by-step based on an example path computed for the first pair of waypoints. Fig. 5.17 presents the initial configuration of the scene. The UAV is assumed to be at the start waypoint, shown in red. The green waypoint indicates its current goal, i.e., the next global waypoint.

In the first phase (Fig. 5.18), LPP samples the state space (here, using RRT), looking for a feasible path to goal. The tree is shown in blue, where dots represent its vertices and lines are the edges. The shortest path from start to goal is shown in red. The placement of vertices is validated by connecting the subsequent vertices with a Dubins paths.

Paths generated by LPP tend to be suboptimal. To simplify them, the smoothing algorithm by Beard and McLain was used (see Algorithm 6). The algorithm removes redundant vertices, which are not required for collision-free flight, as in Fig. 5.19. It does not optimize the placement of the vertices, though. The smoothed path is seen in green.

Now, a Dubins path through the remaining vertices is computed, as in Fig. 5.20. The Dubins path shown in yellow respects the kinematic constraints of the modeled airplane, such as maximal climbing angle and airspeed. For example, the steep edge of the tree is replaced with an ascending helix. Note that even if an edge of the tree (green) crosses an obstacle, the resulting Dubins path (yellow) may still be feasible. Finally, the path is discretized into local waypoints (Fig. 5.21). Sampling step is a parameter of LPP.

Tab. 5.12: Detailed configuration of the randomly generated waypoint pairs

#	Start waypoint local poses				Goal waypoint local poses				d_f[m]
	x [m]	y [m]	z [m]	ψ [rad]	x [m]	y [m]	z [m]	ψ [rad]	
1	5379	-5615	-1153	-1.727	-3890	78	-1289	0.947	10878
2	6070	4173	-483	-2.190	9908	-3975	-2115	-3.038	9153
3	1078	-6906	-1427	-2.474	12227	-4429	-1991	-1.279	11436
4	-1488	550	-570	-0.780	4360	-7555	-790	-0.591	9997
5	-6337	-2552	-1810	-1.306	2900	1623	-847	1.763	10182
6	-5196	-8722	-1010	1.548	5073	-5536	-2539	0.750	10860
7	-4736	1646	-803	-2.540	-8409	-7574	-1082	2.742	9928
8	5643	-7552	-1196	0.535	-5520	-6222	-867	0.740	11247
9	-1699	-5186	-1671	0.507	6762	-347	-1529	-1.953	9748
10	-4895	367	-1033	2.590	5776	-5633	-2188	1.311	12297

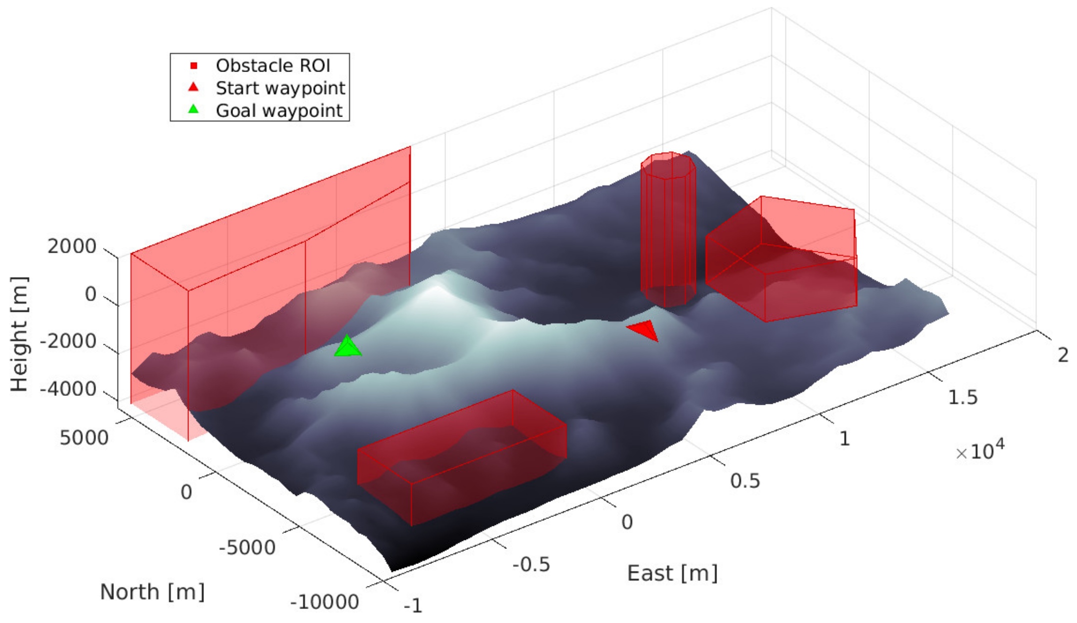

Fig. 5.17: Initial state of the scene

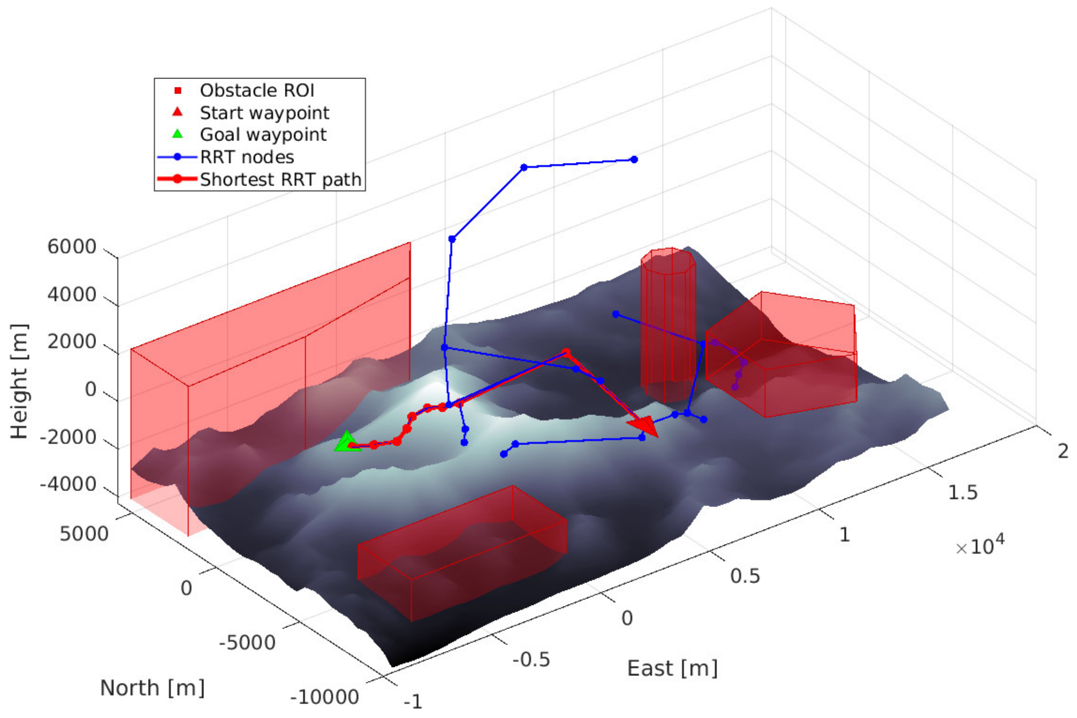

Fig. 5.18: Random tree computed by LPP and the shortest path to goal

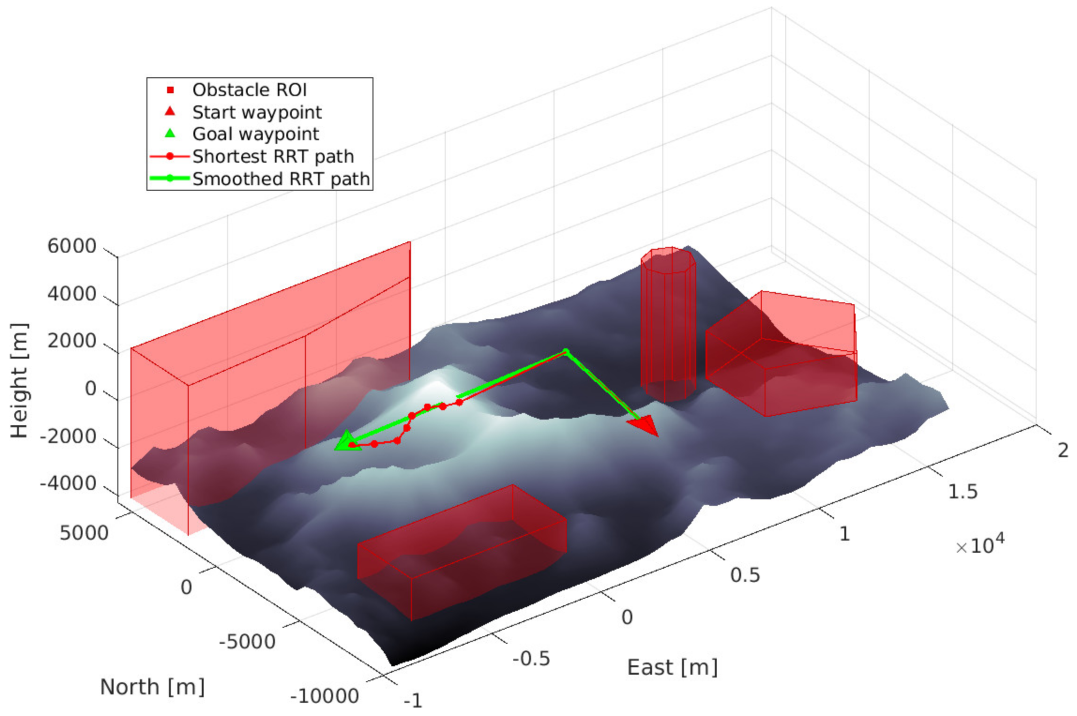

Fig. 5.19: The shortest path is simplified by removing redundant vertices

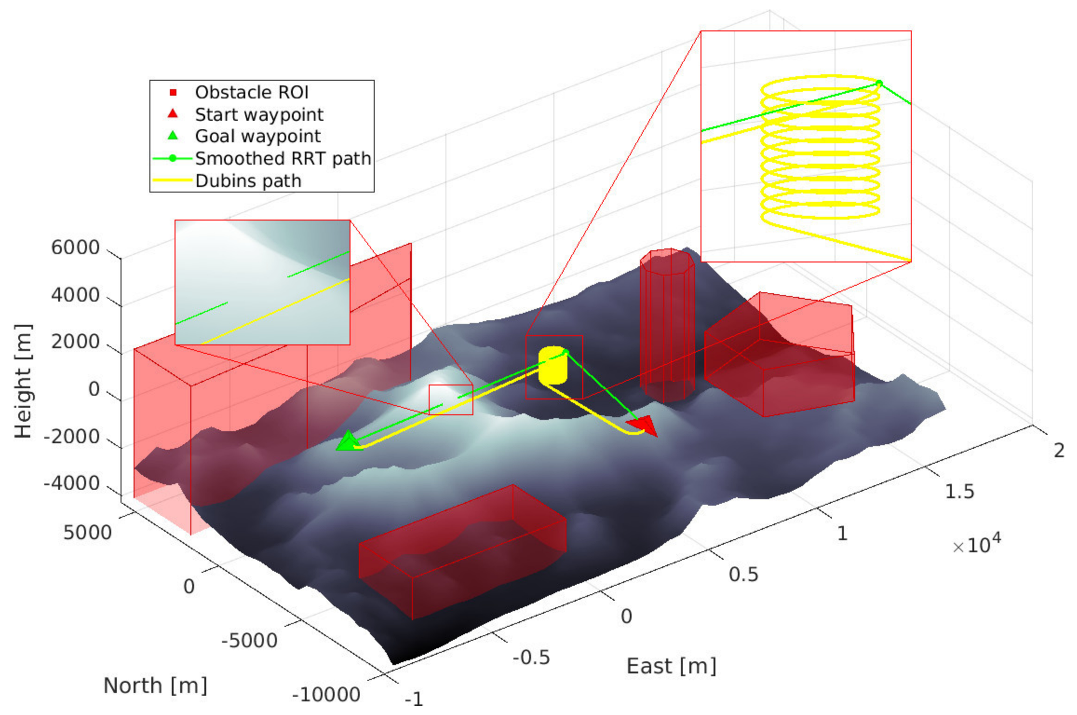

Fig. 5.20: Dubins path connects the start and goal via the smoothed RRT vertices

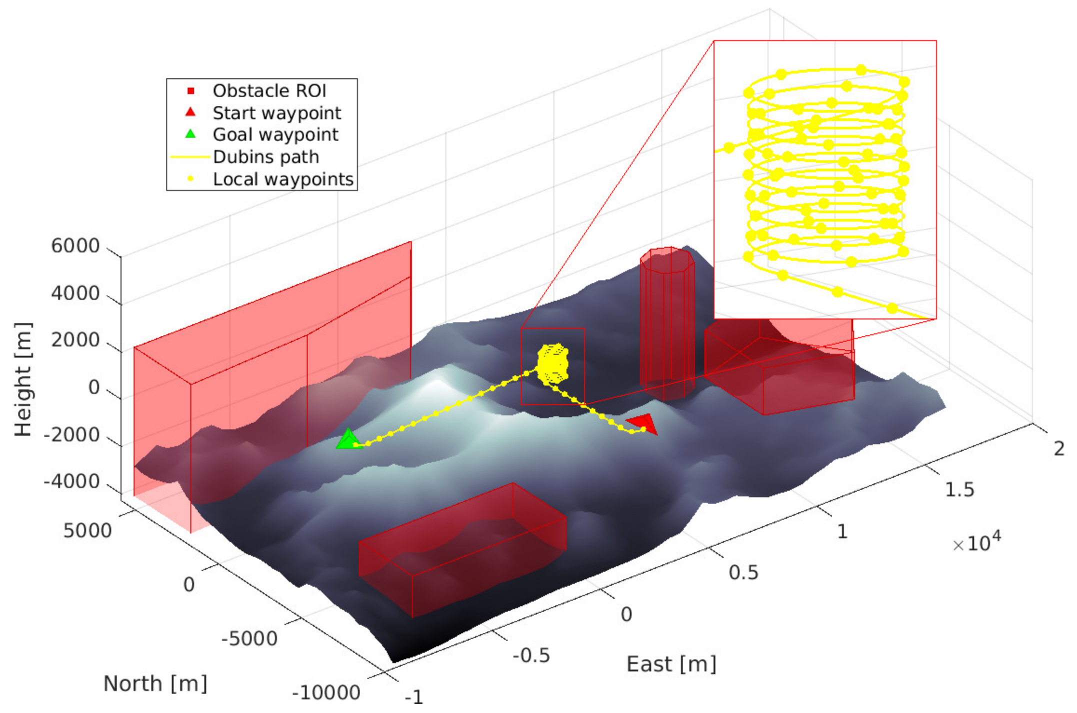

Fig. 5.21: Discretized Dubins path returned by LPP

5.6.3 Comparison of chosen RRT algorithms

The experiment employed three different RRT-based algorithms: RRT, RRT* and BiRRT. The test was repeated 30 times for each algorithm and each pair of the waypoints described in Tab. 5.12. Parameter settings used throughout the test are summarized in Tab 5.13. The parameters not mentioned were set to their default values, as implemented in MATLAB.

The first series of experiments used RRT as the path planning algorithm with and without the smoothing Algorithm 6. Fig. 5.22 illustrates a close-up comparison of the first 5 of 30 RRT paths without (top) and with (bottom) the smoothing algorithm in the case of the first random waypoint pair. The smoothing algorithm simplified the RRT path by removing redundant waypoints. In this case, it resulted in the same shape of the second and third local paths (see the overlapping lines in Fig. 5.22). While the figure is not meant to clearly display the details of the paths, it shows high variability in subsequent algorithm runs. Note, the algorithm parameters have not changed between generating each path (except for random seed).

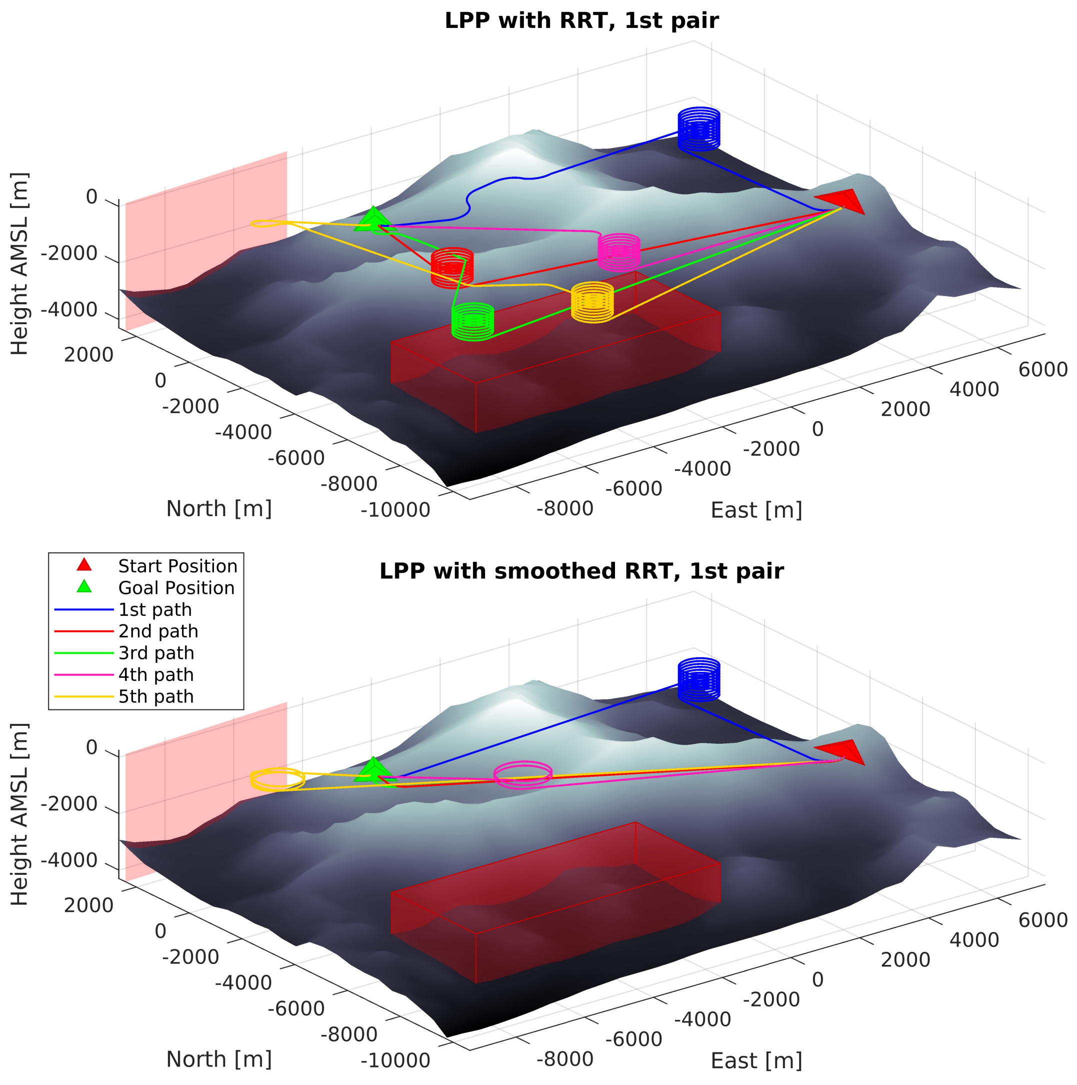

Fig. 5.22: Random RRT paths: raw (top) and smoothed (bottom)

Next, the same experiment was repeated for RRT* (Fig. 5.23) and BiRRT (Fig. 5.24). Again, smoothing the paths resulted in less variability and more visually consistent results. Detailed quantitative results summarized for the first waypoint pair and different planning algorithms are given in Tab. 5.14, where the algorithms are compared using the basic statistical measures

Tab. 5.13: Parameters of LPP during the experiment

#	Parameter	Value	Unit	Description
1	Airspeed	22	$\frac{\mathrm{m}}{\mathrm{s}}$	Constant UAV airspeed for defining Dubins paths
2	Ball radius	1000	m	Constant, that determines the diameter of an imaginary ball, which limits the range of RRT* optimization
3	Distance to waypoint	500	m	Constant step between the neighboring local waypoints in the final path
4	Flight-path angle limits	$\langle -\frac{\pi}{90}, \frac{\pi}{12} \rangle$	rad	Flight-path angle limit when ascending and descending, respectively
5	Goal bias	0.3	-	The probability of choosing the actual goal state during the process of randomly selecting states from the state space
6	Linear goal tolerance	10	m	Max linear distance to goal to consider it as reached
7	Max connection distance	5000	m	Maximal Euclidean distance between the current RRT vertex and the next randomly generated vertex
8	Max iterations	500	-	Reaching this value restarts RRT with different random seed
9	Max number of vertices	10000	-	Limits the number of vertices of the random tree
10	Max roll angle	$\frac{\pi}{30}$	rad	Symmetrical roll angle limit
11	Safety margin	100	m	Minimal distance to obstacles (terrain and airspace NFZs)

and metrics described above. Sample size for each case is 30.

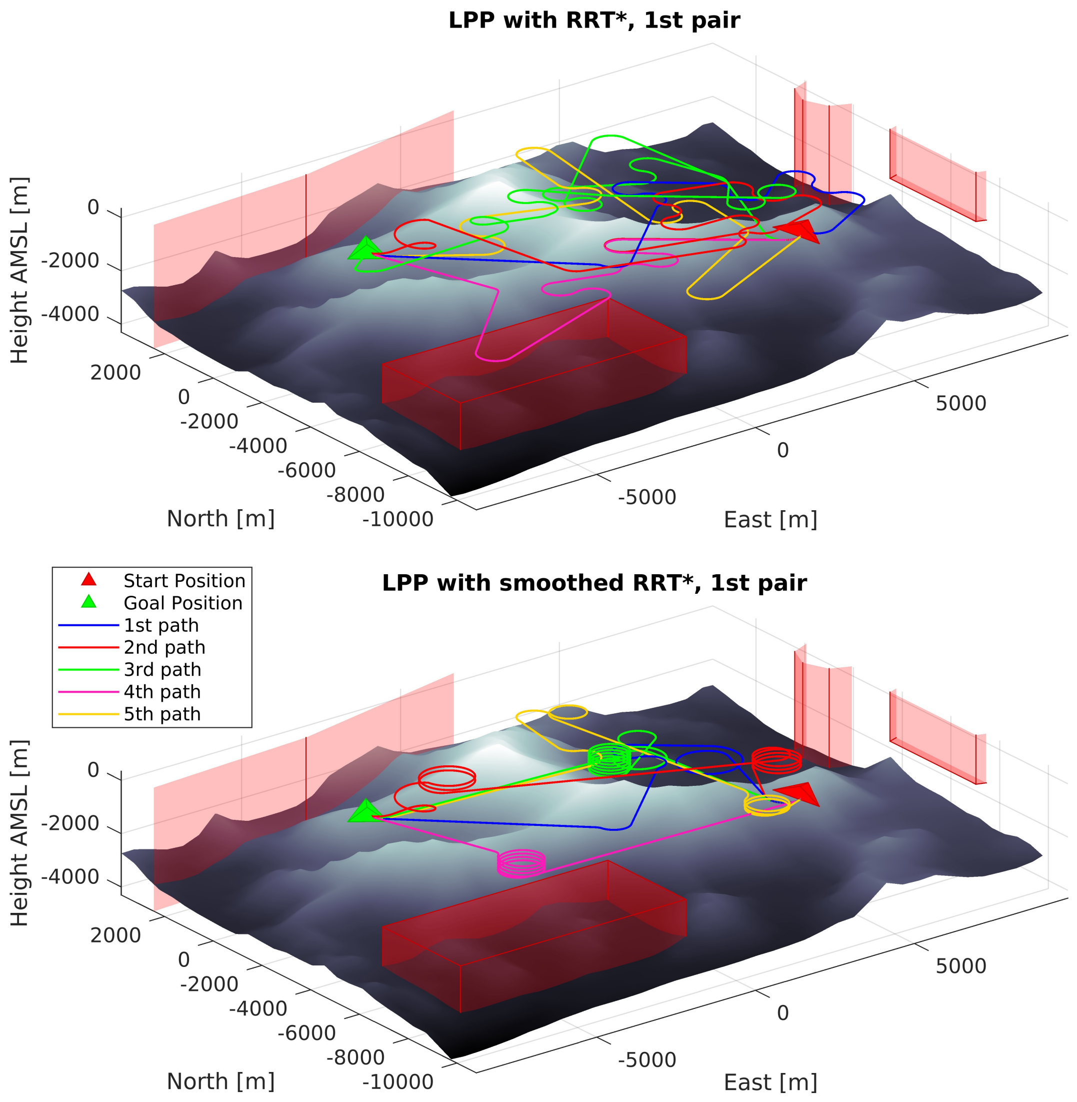

Fig. 5.23: Random RRT* paths: raw (top) and smoothed (bottom)

BiRRT resulted in the smallest and the most repeatable CT before smoothing (Mean = 0.213, StDev = 0.224) and after (Mean = 0.276, StDev = 0.237). While genuine RRT resulted in the shortest mean LEN of 25802 m after smoothing, the paths had the highest standard deviation of 20538 m. Both RRT* and BiRRT provided somewhat longer, but more repeatable results with BiRRT performing slightly better (mean LEN lower by 2866 m with standard deviation lower by 3639 m).

Before smoothing, RRT had the lowest SMOO rating, while after applying the smoothing algorithm SMOO for all three algorithms was comparable. Interestingly, smoothed RRT had the highest SMOO, despite having initially the most complicated tree structure indicated by significantly higher NTREE. After smoothing, the paths had comparable NRRT, which justifies the similar SMOO values. All three algorithms resulted in collision-free paths (NCOL = 0).

To provide statistically relevant results, the experiment above was repeated for all 10 different waypoint configuration for a total of 900 test cases. The results were summarized for each algorithm. Then, basic statistical metrics were computed from all test cases for each algorithm (sample size = 300). The results are shown in Tab. 5.15.

However, even if the distances between the waypoints in each pair are comparable, some

Tab. 5.14: Detailed metrics for the 1st waypoint pair, sample size is 30 for each algorithm

RRT										
Metric	**Path before smoothing**					**Path after smoothing**				
	Mean	StDev	Min	Max	Median	Mean	StDev	Min	Max	Median
CT [s]	1.115	1.853	0.049	7.906	0.343	1.190	1.865	0.077	8.046	0.428
LEN [m]	44914	20837	11964	93533	42863	25802	20538	11717	78903	16097
SMOO	0.796	0.069	0.723	0.966	0.769	0.888	0.090	0.709	0.965	0.937
NTREE	57	67	4	277	32	57	67	4	277	32
NRRT	11	4	4	22	11	3	0	3	4	3
NPTS	93	44	24	195	89	53	43	24	164	33
NCOL	0	0	0	0	0	0	0	0	0	0

RRT*										
Metric	**Path before smoothing**					**Path after smoothing**				
	Mean	StDev	Min	Max	Median	Mean	StDev	Min	Max	Median
CT [s]	0.624	1.333	0.063	7.092	0.199	0.685	1.345	0.087	7.168	0.243
LEN [m]	38128	24271	12219	114034	33368	32915	17823	11639	81622	31114
SMOO	0.898	0.039	0.830	0.971	0.889	0.844	0.076	0.733	0.969	0.826
NTREE	24	35	5	174	14	24	35	5	174	14
NRRT	9	5	4	25	8	4	1	3	7	4
NPTS	78	50	25	232	68	68	37	23	168	64
NCOL	0	0	0	0	0	0	0	0	0	0

BiRRT										
Metric	**Path before smoothing**					**Path after smoothing**				
	Mean	StDev	Min	Max	Median	Mean	StDev	Min	Max	Median
CT [s]	0.213	0.224	0.051	1.334	0.189	0.276	0.237	0.083	1.427	0.239
LEN [m]	39783	17001	12151	78112	41391	30049	14184	11630	62654	30081
SMOO	0.872	0.033	0.796	0.94	0.878	0.841	0.068	0.720	0.964	0.829
NTREE	16	10	5	59	15	16	10	5	59	15
NRRT	9	3	4	17	10	4	1	3	6	4
NPTS	81	35	24	160	84	62	30	23	130	62
NCOL	0	0	0	0	0	0	0	0	0	0

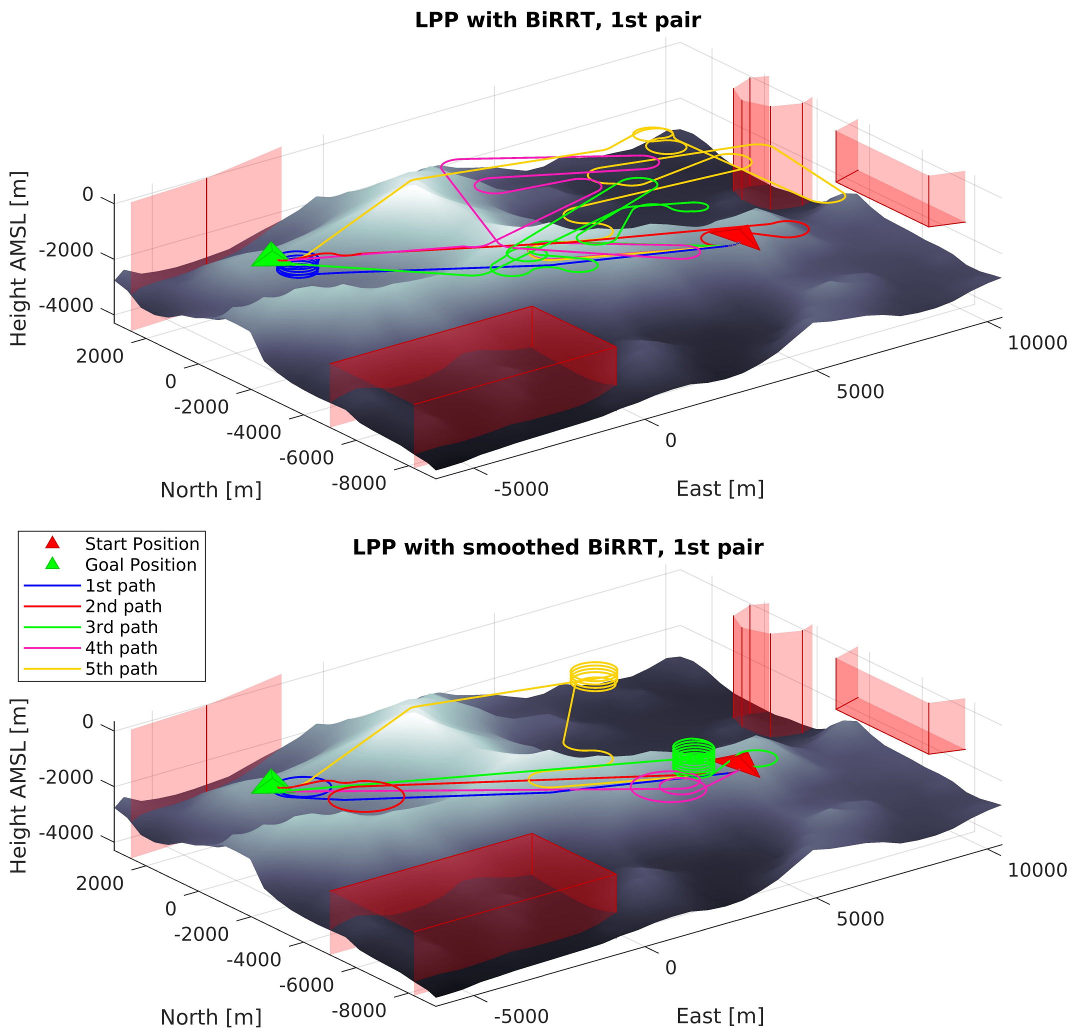

Fig. 5.24: Random BiRRT paths: raw (top) and smoothed (bottom)

pairs are placed with an obstacle between them, while there is a clean line-of-sight between the others. This fact impacts the initial path planning difficulty of each pair, making the results less comparable between different pairs of waypoints. Nonetheless, the same case is true across all the algorithms, which reduces the risk of biasing the results.

Similarly to Tab. 5.14, BiRRT resulted in the smallest CT (Mean = 0.271, StDev = 0.611), followed by RRT*, and then RRT. This difference was mitigated by smoothing the paths but still persists. Nevertheless, RRT* produced the shortest mean LEN with the second-lowest standard deviation. Difference in length of the paths is also reflected by NPTS. RRT* produced the smoothest paths (SMOO : Mean = 0.915, StDev = 0.043). The difference, however, was more severe before applying the smoothing algorithm.

RRT produced the most complex random trees indicated by the highest NTREE. NTREE for RRT* and BiRRT was comparable (Mean = 17 for both). The complexity of BiRRT trees was more consistent, though (StDev = 19 for BiRRT vs StDev = 30 for RRT*). Also the results for NRRT supported this observation. As in the previous experiment, the algorithms provided valid paths with NCOL = 0 for all of them.

As in the case of GPP, the same point-based optimization method was used. The mean results from Tab. 5.15 were evaluated by assigning weights to them according to their importance for LPP. As the smoothing algorithm significantly contributes to the path quality with

Tab. 5.15: Summary for all the test cases, sample size is 300 for each algorithm

RRT										
Metric	**Path before smoothing**					**Path after smoothing**				
	Mean	StDev	Min	Max	Median	Mean	StDev	Min	Max	Median
CT [s]	0.677	1.166	0.029	7.906	0.292	0.741	1.182	0.047	8.046	0.354
LEN [m]	38758	29532	9868	143730	29257	24382	21123	9660	118244	12861
SMOO	0.835	0.082	0.706	0.992	0.815	0.901	0.089	0.705	0.994	0.947
NTREE	39	45	3	277	25	39	45	3	277	25
NRRT	11	5	4	29	11	3	1	2	6	3
NPTS	80	62	20	299	61	50	44	19	245	26
NCOL	0	0	0	0	0	0	0	0	0	0

RRT*										
Metric	**Path before smoothing**					**Path after smoothing**				
	Mean	StDev	Min	Max	Median	Mean	StDev	Min	Max	Median
CT [s]	0.462	1.065	0.029	7.114	0.119	0.503	1.074	0.047	7.168	0.159
LEN [m]	25694	18070	10159	114034	18063	21760	14587	9763	81622	13195
SMOO	0.915	0.043	0.802	0.988	0.912	0.907	0.074	0.733	0.991	0.946
NTREE	17	30	3	198	8	17	30	3	198	8
NRRT	7	4	4	25	5	3	1	2	7	3
NPTS	52	37	20	232	37	44	30	20	168	27
NCOL	0	0	0	0	0	0	0	0	0	0

BiRRT										
Metric	**Path before smoothing**					**Path after smoothing**				
	Mean	StDev	Min	Max	Median	Mean	StDev	Min	Max	Median
CT [s]	0.271	0.611	0.035	5.625	0.116	0.322	0.62	0.053	5.704	0.166
LEN [m]	34904	16850	11405	90532	29890	23387	14631	9708	73654	16730
SMOO	0.869	0.04	0.768	0.974	0.871	0.895	0.072	0.720	0.996	0.906
NTREE	17	19	4	162	12	17	19	4	162	12
NRRT	8	3	4	20	7	3	1	2	8	3
NPTS	71	34	23	186	61	48	30	20	152	34
NCOL	0	0	0	0	0	0	0	0	0	0

relatively small computation cost, only smoothed paths were considered in the evaluation (the rightmost columns of Tab. 5.15). NCOL and NPTS have been removed from the evaluation as no collisions occurred during the experiment and NPTS is highly positively correlated to LEN. Mean correlation coefficient across all the algorithms (unsmoothed and smoothed) between the two metrics was 0.815.

LEN was chosen as the most important metric, as it correlates with many crucial mission parameters, such as energy expended by the UAV and time of the mission. Then, SMOO determines the easiness of following the path, intensity of control maneuvers and thus also energy expenditure. The UAV may fail to closely follow a path with very low SMOO. The next metric is CT. It was chosen after LEN and SMOO because LPP acts as a middle-level planner, so time regime is not as strict as for real-time capable low level controller. Nonetheless, it is still required to keep CT as low as possible to grant reactivity of LPP. NRRT relates to the quality of internal path returned by RRT-class algorithms. Its impact on the final path quality is reduced by introducing Dubins paths. For example, a result with more RRT vertices may provide simpler Dubins path, if height difference between the vertices is relatively small, so helix-based ascending maneuvers are not necessary. NTREE comes at the end as it does not influence the complexity of the final shortest path. For example, a large tree with high NTREE can have many complex branches, but the shortest one may still consist just of a few vertices.

The metrics were given 2 (best), 1 or 0 (worst) points, similarly as for GPP. Then, weighted sums as in eq. 5.10 were computed. Tab. 5.16 presents the evaluation results.

Tab. 5.16: Point-based multi-criteria optimization table

Metric	**Weight**	**Grade**			**Grade × Weight**		
		RRT	RRT*	BiRRT	RRT	RRT*	BiRRT
CT	3	0	1	2	0	3	6
LEN	5	0	2	1	0	10	5
SMOO	4	1	2	0	4	8	0
NTREE	1	0	1	2	0	1	2
NRRT	2	0	2	1	0	4	2
		Weighted sum			4	26	15
		Max points			30	30	30
		Fitness of solution			13.33%	**86.67%**	50.00%

RRT* turned out to be the optimal LPP algorithm with BiRRT being the alternative solution. Genuine RRT got the worst grades for all the metrics but SMOO, so it became the least suitable algorithm for LPP. Hence, RRT* will be used in the further research.

5.6.4 Tuning the chosen algorithm

The RRT* algorithm was further calibrated to tune its capabilities to the requirements of LPP. Two most important parameters were chosen: maximum connection distance (MCD) and goal bias (GB). MCD is the maximum length of an edge of the random tree. In other words, it is a maximal length of motion given as Euclidean distance allowed between two connected tree vertices. GB is the probability of choosing the actual goal state during the process of randomly selecting states from the state space [59]. That is, GB determines how many waypoints will be drawn before the current sampling phase ends. Hence, lower GB provides more variability in a tree, but increases computation time.

The experiment as above was repeated 30 times for each of 10 pairs of waypoints and for 15 different combinations of the parameter values. This resulted in a total of 4500 test cases. The minimal value of MCD is limited by the maneuverability of the aircraft. To produce a valid set of Dubins paths, MCD must allow at least 2 full turns with radius equal to the minimum turning radius for the aircraft (470 m in this case). Hence, the minimum value of MCD rounded up to full thousands is 3000 m.

The results were then evaluated by calculating the basic statistical measures of the path metrics for each pair of the parameters. For the same reasons as described in section 5.6.3, the mean values of five metrics were chosen for the evaluation of the results. The metrics included: CT , LEN , SMOO , NTREE and NRRT .

The mean values of each metric for a parameter pair were sorted from worst to best. The worst value got 1 point, the second-worst got 2 points etc. Hence, the best value for a given metric got 15 points. Then, a similar point-based multi-criteria optimization approach as in section 5.6.3 was employed using eq. 5.10. It resulted in a 15-cell Tab. 5.17. Each cell contains a point grade for each combination of the chosen parameters. Therefore, the optimal pair of parameter values, which scored 201 points, is MCD = 10000 m, and GB = 0.3.

Tab. 5.17: Evaluation of the results for 15 pairs of chosen RRT* parameters

GB / **MCD** [m]	0.05	0.1	0.3
3000	34	66	166
4000	41	92	143
5000	37	104	191
7000	84	92	187
10000	118	124	**201**

5.6.5 Validation of adaptive re-planning in simulation

The last part of the LPP verification study was to test the planner in dynamic scenarios described below. To examine the adaptability of LPP, three different scenarios are considered:

1. **Adaptive mid-flight local re-planning** addresses the problem of finding a feasible fallback path to an alternative waypoint (emergency landing, for example) if the initial waypoint cannot be reached.
2. **Sequential path planning** simulates the scenario, when the fallback local path to the same goal waypoint is periodically recalculated, for example, due to frequent gusts of strong wind.
3. **Emerging obstacle** covers a scenario, when the environment map is updated with an obstacle without updating global path. The path must be recalculated using onboard LPP.

Adaptive mid-flight local re-planning

In the first scenario, the position of the goal waypoint changes mid-flight, e.g., a simulation of emergency landing request. The start waypoint is shown in Fig. 5.25 as a red polyhedron indicating the initial pose of the aircraft. Two other polyhedrons denote the preliminary (blue) and the final (green) goal of the UAV. Initially, LPP is not aware of the second goal, thus it plans the path directly to the first goal. Then, after given time, the destination changes and the planner has to adapt the path to guide the UAV to its new goal. The exact mission scenario is presented in Tab. 5.18. All local coordinates are given in ENU frame in meters relative to the origin of the map.

Tab. 5.18: Details of the scenario of the first dynamic LPP mission

Scenario ID	AdaptiveLPP1
Description	Goal changes during flight
Goal	The UAV travels from the preset start waypoint to the preset goal waypoint. Then, after a given amount of time, the mission goal changes. LPP must adapt by planning a new path and reaching a new goal waypoint.
Stages	1. UAV is airborne at the start waypoint, airspeed = 22 $\frac{\mathrm{m}}{\mathrm{s}}$. 2. LPP plans a local path from the start waypoint to the goal waypoint. 3. Simulated UAV flies along the path for 5 minutes of simulation time. 4. The goal waypoint changes and LPP re-plans the local path. 5. Simulated UAV flies along the new path until it reaches the goal.
Map	Rectangle from (49.160000°N, 20.000000°E) to (49.220000°N, 20.120000°E), local origin $(0, 0, 0)$ at (49.160000°N, 20.000000°E, 0 m AMSL), no wind
UAV model	Fixed-wing kinematic guidance model calibrated as in section 5.4.3
Algorithms	RRT* calibrated as in section 5.6.4
Mission parameters	Kinematic constraints of the UAV as in Tab.5.13 Max connection distance = 10000 m Airspeed = 22 $\frac{\mathrm{m}}{\mathrm{s}}$ Start waypoint = $(12000, 2500, 2100, \pi)$ Initial goal waypoint = $(770, 3300, 1900, \pi)$ Final goal waypoint = $(8700, 5700, 1600, \frac{\pi}{4})$
Criteria of success	FT $\leqslant$ 1024 s NCOL $= 0$ max(CT) $\leqslant$ 4.55 s

The mission is considered successful if all criteria of success are met. The criterion of maximum time of flight is computed as

$$\mathrm{FT} \leqslant \frac{s_m \left\|\overline{\mathbf{w}_s \mathbf{w}_g}\right\|}{v_a}$$

where w_s and w_g denote the positions of start and goal waypoints, and v_a is a constant airspeed of 22 $\frac{\mathrm{m}}{\mathrm{s}}$, as in Tab. 5.13. Variable s_m is a scenario-dependent safety multiplier empirically, here set to 2. FT refers to the time of simulated flight as a whole – from start to the final goal.

The value of maximum CT is given as safety margin (100 m) divided by airspeed (22 $\frac{\mathrm{m}}{\mathrm{s}}$), rounded to the second decimal point. Therefore, for values from Tab. 5.13 $\max(\mathrm{CT}) \leqslant 4.55$ s. Note, $\max(\mathrm{CT})$ means CT of the longest computation in a given test case.

The results for the scenario are shown in Fig. 5.25. Upper figure presents the initial path planned (in yellow) and partially simulated (in red). Lower figure shows the second part of the flight. Green square indicates the point where the re-planning takes place. The UAV follows then the new planned path, smoothly transitioning from the previous one. The mission completed with total $\mathrm{FT} = 674.155$ s, $\mathrm{NCOL} = 0$ and $\mathrm{CT} = \{0.161\ \mathrm{s}, 0.067\ \mathrm{s}\}$, thus meeting the criteria of success.

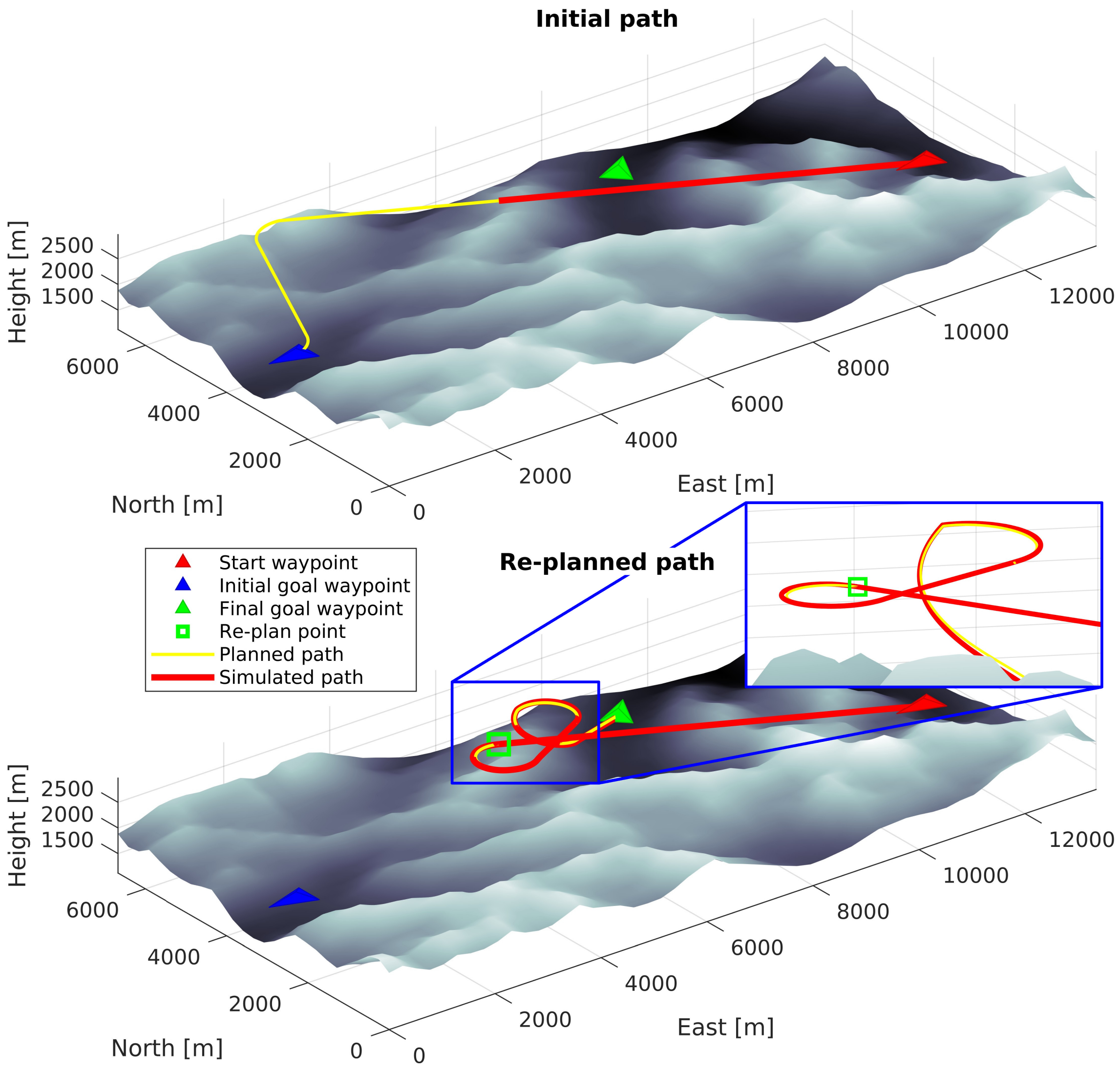

Fig. 5.25: Planned and simulated paths generated in AdaptiveLPP1 (1st case)

After running the experiment, it was found that the controller of the simulated aircraft reacts to slowly and it is unable to closely follow the path. The issue was caused by low sampling density of the resulting path returned by LPP. Decreasing “Distance to waypoint” parameter of LPP (see Tab. 5.13) from 500 m to 100 m solved the problem. Note, this issue was not present in the model calibration (section 5.4.3) because the helical path used for calibration was generated manually, i.e., not using parametrized LPP.

As before, to provide statistically relevant results, the experiment was repeated 30 times with different random seed. The results are summarized in Tab. 5.19. Note that FT, LEN, SMOO

and NCOL were computed using the concatenated simulated path. For the other metrics a plus notation is used, where each term denotes a path fragment. For example, the CT [s] column in the first row shows 0.13 + 0.06. It means that for the first fragment CT = 0.13 s and for the second one CT = 0.06 s. This notation is extendable to an arbitrary number of path fragments.

Due to relatively small sampling step used in simulation, SMOO was calculated by taking every 500th simulated waypoint. The other metrics show values for each of the planning steps.

None of the test cases resulted in collisions, i.e., NCOL > 0. In the 8th, 15th and 23rd case, however, the path returned by LPP was highly non-optimal and lengthy, thus violating the FT $\leqslant$ 1024 s constraint (Fig. 5.26). Also in the 27th case the max(CT) $\leqslant$ 5 s constraint was violated with CT = 6.59 s. The paths in other cases met all the criteria. Hence, the success rate of LPP in the experiment was 24/30 = 80.00%.

Tab. 5.19: Metrics computed for the results of AdaptiveLPP1

#	CT [s]	FT [s]	LEN [m]	SMOO	NTREE	NRRT	NPTS	NCOL
1	0.13 + 0.06	674.155	14831	0.766	3 + 3	3 + 3	140 + 81	0
2	0.06 + 0.02	455.745	10026	0.803	3 + 1	4 + 2	246 + 35	0
3	0.12 + 0.02	445.514	9801	0.714	3 + 1	4 + 2	243 + 32	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
8	0.29 + 0.21	1999.882	43997	0.551	28 + 21	4 + 3	368 + 369	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	0.09 + 0.21	1147.327	25241	0.782	6 + 159	4 + 4	348 + 185	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
23	0.06 + 2.24	1118.016	24596	0.786	2 + 113	3 + 4	146 + 177	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	6.59 + 0.04	437.69	9629	0.784	218 + 6	4 + 2	689 + 31	0
28	2.99 + 0.02	432.615	9518	0.885	139 + 2	4 + 2	638 + 30	0
29	0.06 + 0.48	720.255	15846	0.832	4 + 38	3 + 3	150 + 91	0
30	0.08 + 0.05	960.416	21129	0.589	7 + 4	4 + 3	239 + 142	0

Suboptimality in some test cases, e.g., the 8th case, was caused by "Max connection distance" set to relatively high value of 10000 m. Decreasing this parameter increases the initial path complexity, but helps avoid local minima, i.e., collision-free, but unnecessary long paths. Thus, it was decided to reduce the parameter in further experiments.

Sequential path planning

In the second dynamic test case, several updates of the local path occur. The scenario simulates a flight, which is often disrupted, e.g., by wind gusts causing the UAV to deviate from its initial path. Nevertheless, no change in position of the UAV due to wind is actually modeled in this case.

At the beginning fo the mission, LPP plans an initial path to goal. Then, simulation begins. After given time, simulation pauses and LPP is ordered to plan a new path from the current pose of the UAV to the unchanged goal waypoint. Simulation resumes and the aircraft flies toward its destination until next re-planning request occurs. Re-planning and simulation phases alternate

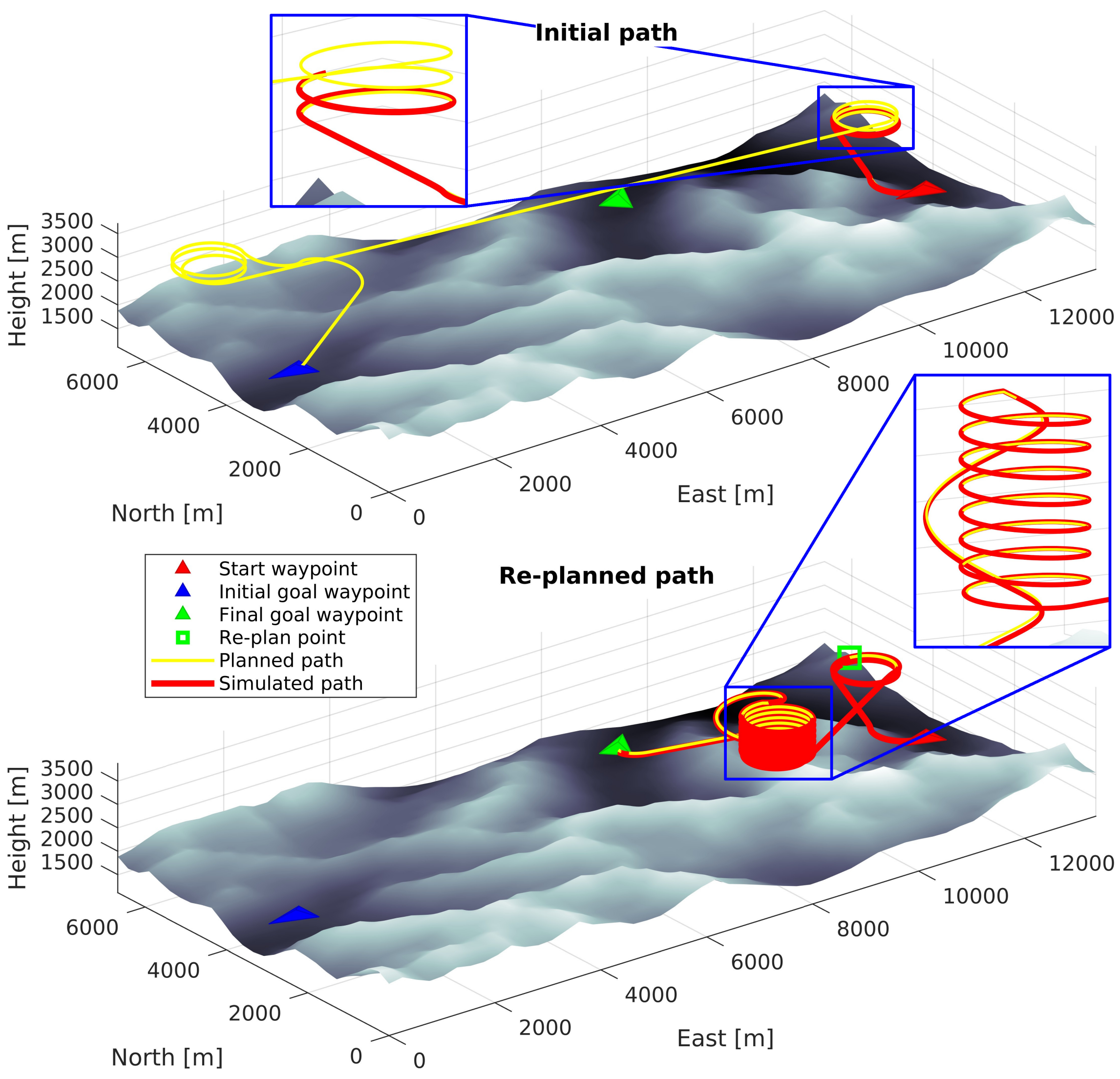

Fig. 5.26: Suboptimal path generated by LPP during AdaptiveLPP1 (8th case)

until the aircraft reaches its destination. The exact mission scenario is presented in Tab. 5.20. The criteria of success were defined as before. However, due to high flight inconsistency due to frequent path re-computations, s_m was increased to 6, giving $\text{FT} \leqslant 5185$ s.

The experiment was repeated for the same start and goal, but with different random seed for a total of 30 test cases. The result of the first case is visualized step-by-step in Fig. 5.27. During the flight a total of six re-planning events take place, resulting in 7 paths (including the initial one). The current path planned for a phase is shown in yellow, while the thick red line depicts the path flied by the UAV until the phase ends. Green squares denote the positions, where LPP re-plans the path. Note each iteration the new path visibly deviates from the previous one, which is caused by the randomness of RRT*.

The results of the other test cases are shown in Tab. 5.21. The metrics computed independently for each phase (NTREE , NPTS , NRRT and NCOL) were summed for brevity. For CT , maximum values for each test case were used instead. FT , LEN and SMOO are computed for the full simulated path. For SMOO , the path was re-sampled with sampling step of 500, as before.

The mission succeeded in 20 of 30 test cases, giving the success rate of 66.67%. For cases 1, 5, 9, 10, 14, 19, 20, 21, 22 and 23 the $\text{FT} \leqslant 5185$ s constraint has been violated. Additionally,

Tab. 5.20: Details of the scenario of the second dynamic LPP mission

Scenario ID	AdaptiveLPP2
Description	Cyclic re-planning attempts
Goal	The UAV travels from the preset start waypoint to the preset goal waypoint. At constant period of time the local path must be re-planned. LPP must adapt by planning a new path and finally reaching the goal waypoint.
Stages	1. UAV is airborne at the start waypoint, airspeed = 22 $\frac{\text{m}}{\text{s}}$. 2. LPP plans a local path from the start waypoint to the goal waypoint. 3. Simulated UAV flies along the path for 15 minutes of simulation time. 4. LPP plans a new path, but the goal remains the same. 5. Steps 3 and 4 repeat until the UAV reaches the goal waypoint.
Map	Rectangle from (27.930000°N, 86.790000°E) to (28.050000°N, 87.020000°E), local origin $(0,0,0)$ at (27.930000°N, 86.790000°E, 0 m AMSL), no wind
UAV model	Fixed-wing kinematic guidance model calibrated as in section 5.4.3
Algorithms	RRT* calibrated as in section 5.6.4
Mission parameters	Kinematic constraints of the UAV as in Tab.5.13 Max connection distance = 5000 m Airspeed = 22 $\frac{\text{m}}{\text{s}}$ Start waypoint = $(3850, 4800, 5300, \frac{\pi}{4})$ Goal waypoint = $(22750, 6850, 5500, -\frac{3}{4}\pi)$
Criteria of success	$\text{FT} \leqslant 5185$ s $\text{NCOL} = 0$ $\max(\text{CT}) \leqslant 4.55$ s

the 9th and 10th test cases have CT higher than 4.55 s, that is, 8.417 s and 6.029 s, respectively. Most importantly, in the 2nd case the "hard" constraint of $\text{NCOL} > 0$ has been violated by the simulated UAV deviating from the planned path, as seen in Fig. 5.28. The paths generated during this experiment consist of several sub-paths, which contributes to lower SMOO of the final simulated paths.

Emerging obstacle

In the last dynamic test of LPP, the environment map is updated during the mission. The UAV starts by flying along its initial path computed by LPP at the beginning of the mission. Then, it is assumed a new obstacle is detected, which simulates a dangerous area, which must be avoided, e.g., a storm cloud. LPP has to re-plan the path on-the-fly to reach its destination, while avoiding collisions with the new obstacle. The obstacle is placed in such way, it obscures the global waypoint. Scenario details are shown in Tab. 5.22. Safety margin s_m was set to 3 to account for the added obstacle, which results in $\text{FT} \leqslant 1535$ s.

The experiment was repeated for the same start and goal, but with different random seed for

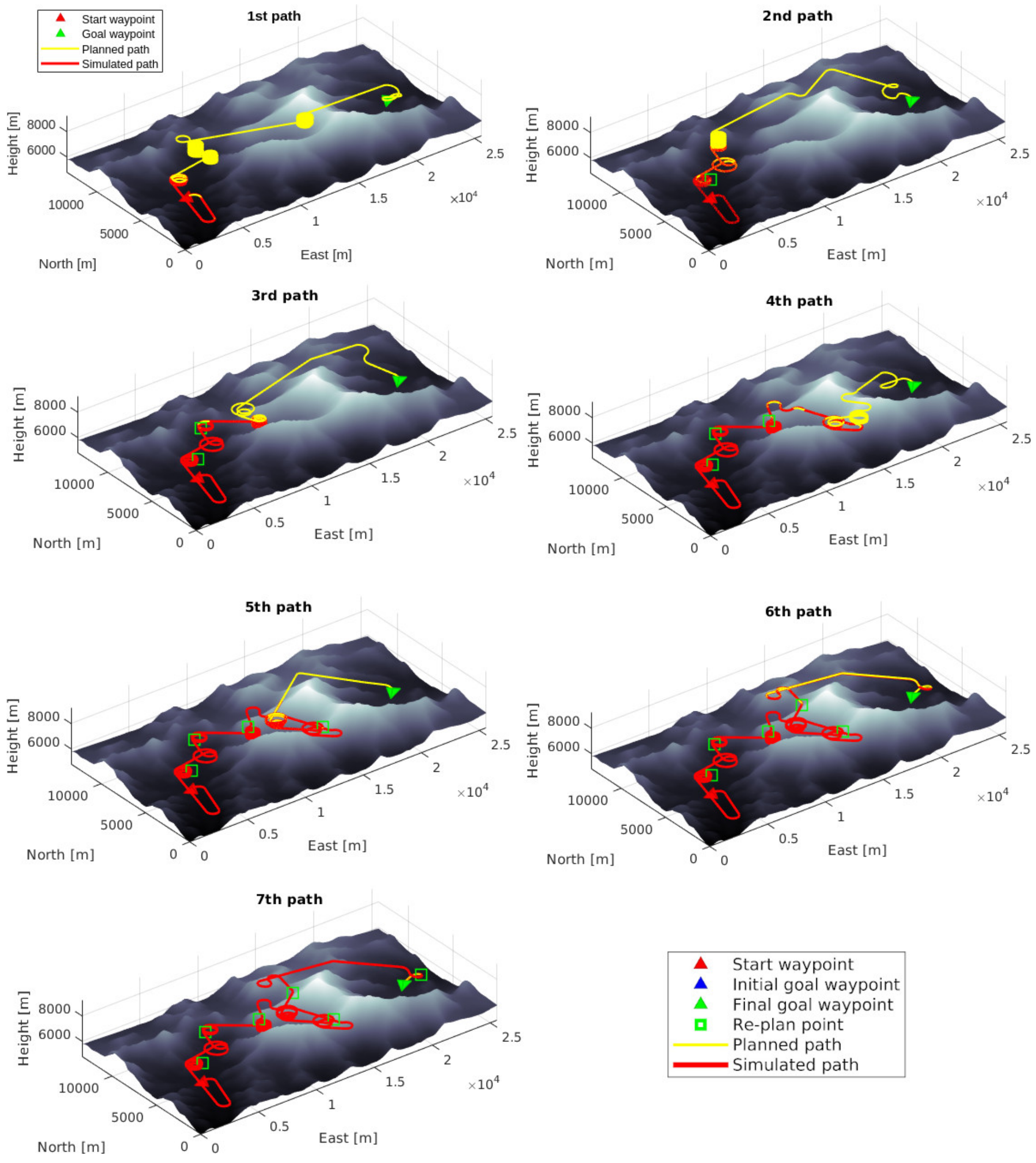

Fig. 5.27: Planned and simulated paths generated in AdaptiveLPP2 (1st case)

a total of 30 test cases. Fig. 5.29 shows the result for the first case. The upper figure illustrates the initial path without the obstacle. The lower one depicts a new path generated after updating the map with the obstacle shown as a transparent red prism. The current path planned for a phase is shown in yellow, while the thick red line depicts the simulated path. Green square denotes the position of the UAV, at which the update occurred.

Tab. 5.23 summarizes the results for all test cases. The metrics were computed as in the first scenario. The mission succeeded in 25 of 30 test cases, giving the success rate of 83.33%. Cases 2, 4, 13, 24 and 25 exceeded the $\mathrm{FT} \leqslant 1535$ s constraint. Maximum $\mathrm{FT} = 2042.083$ s for the 24th case, while case 14 presented the smallest $\mathrm{FT} = 535.885$ s. No case resulted in collisions,

Tab. 5.21: Metrics computed for the results of AdaptiveLPP2

#	max(CT) [s]	FT [s]	LEN [m]	SMOO	NTREE	NRRT	NPTS	NCOL
1	2.398	5475.691	120465	0.660	230	7	3383	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
5	2.136	5209.701	114613	0.704	252	6	3765	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
9	8.418	5494.419	120877	0.637	516	7	5035	0
10	6.029	5776.766	127089	0.674	535	7	4006	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	2.600	5611.571	123455	0.677	394	7	4444	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
19	2.399	5202.89	114464	0.621	315	6	4512	0
20	1.466	5591.851	123021	0.622	258	7	4431	0
21	3.303	5550.232	122105	0.627	396	7	5568	0
22	1.233	5215.345	114738	0.593	157	6	3001	0
23	0.852	6084.96	133869	0.627	265	7	4781	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
29	2.040	4556.773	100249	0.749	167	6	2800	0
30	4.362	5031.91	110702	0.670	334	6	4358	0

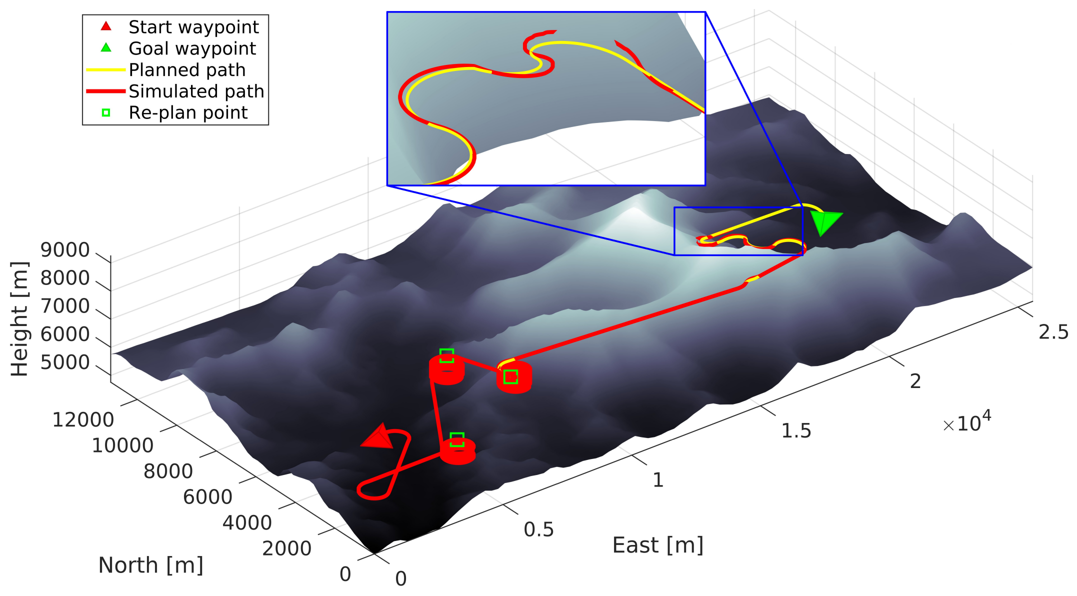

Fig. 5.28: Simulated path collides with terrain – 2nd test case of AdaptiveLPP2

i.e., NCOL > 0. For all test cases CT was lower than 4.55-second limit criterion. The worst CT $= 0.92$ s was measured for the initial path of the 9th case.

Tab. 5.22: Details of the scenario of the third dynamic LPP mission

Scenario ID	AdaptiveLPP3
Description	A new obstacle appears
Goal	The UAV travels from the preset start waypoint to the preset goal waypoint. Then, after a given amount of time, the environment map is updated with a new obstacle, which obscures the goal waypoint. LPP must adapt by planning a new path, avoiding the obstacle and reaching the goal.
Stages	1. UAV is airborne at the start waypoint, airspeed = 22 $\frac{\mathrm{m}}{\mathrm{s}}$. 2. LPP plans a local path from the start waypoint to the goal waypoint. 3. Simulated UAV flies along the path for 150 seconds of simulation time. 4. The environment map updates and a new obstacle is placed. 5. LPP re-plans the local path from the current position of the aircraft to the goal waypoint, avoiding the obstacle. 6. Simulation resumes and the UAV flies along the new path to its goal.
Map	Rectangle from (49.160000°N, 20.000000°E) to (49.220000°N, 20.120000°E), local origin $(0, 0, 0)$ at (49.160000°N, 20.000000°E, 0 m AMSL), no wind
UAV model	Fixed-wing kinematic guidance model calibrated as in section 5.4.3
Algorithms	RRT* calibrated as in section 5.6.4
Mission parameters	Kinematic constraints of the UAV as in Tab.5.13 Max connection distance = 5000 m Airspeed = 22 $\frac{\mathrm{m}}{\mathrm{s}}$ Start waypoint = $(12000, 2500, 2100, \pi)$ Final goal waypoint = $(770, 3300, 1900, \frac{\pi}{4})$
Criteria of success	FT $\leqslant$ 1535 s NCOL $= 0$ max(CT) $\leqslant$ 4.55 s

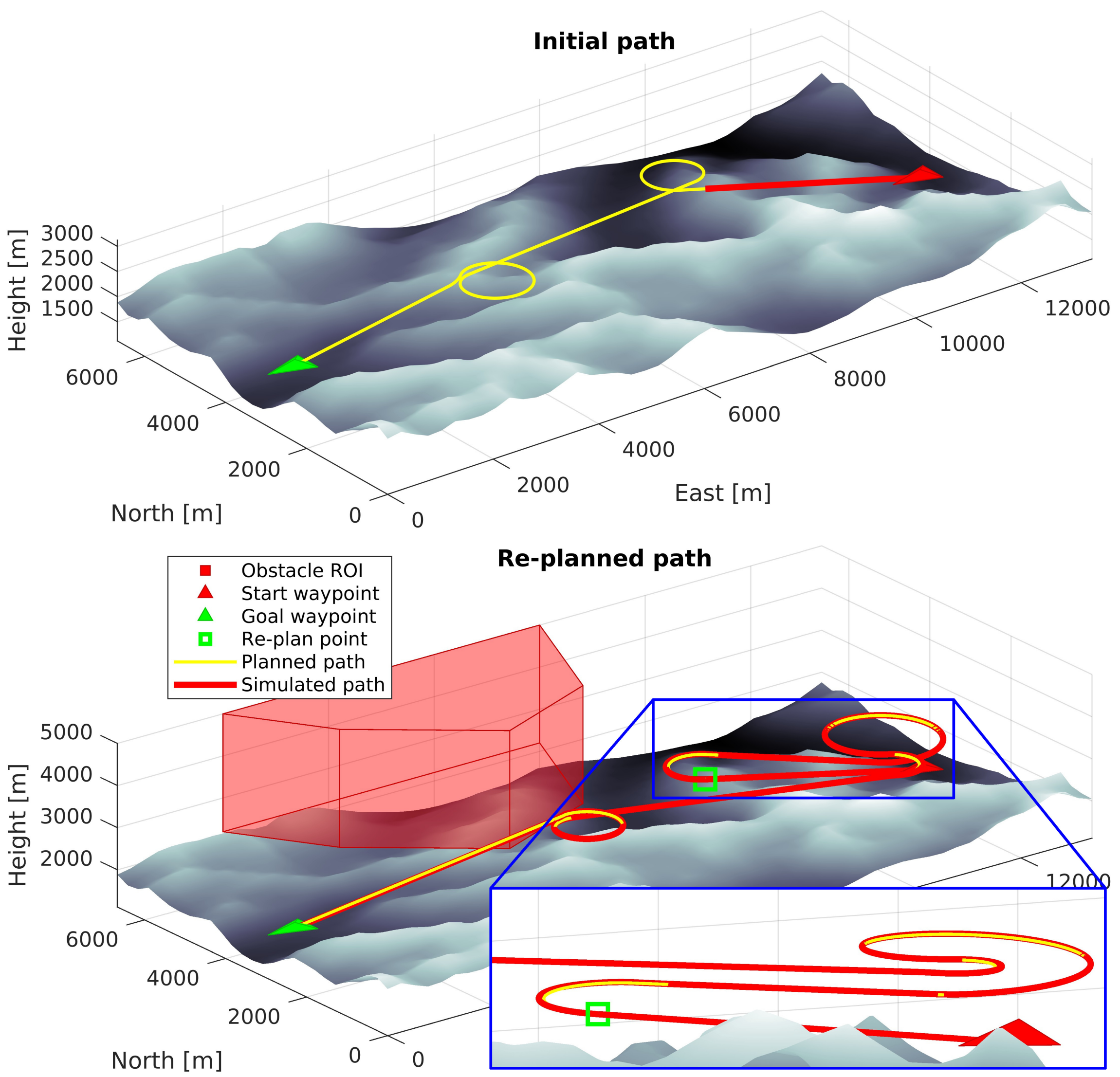

Fig. 5.29: Planned and simulated paths generated in AdaptiveLPP3 (1st case)

5.6.6 Discussion

Even though RRT* was found the optimal path planning algorithm for LPP, its major advantage has not been addressed by this research. RRT* can continue execution to further optimize the path after finding a valid path to goal. That way, it is possible to provide a sub-optimal solution quickly or, alternatively, lengthen the computation time to potentially find an even better solution. For example, setting a fixed computation time (or random tree complexity) can reduce the variability of LEN and SMOO observed between random paths generated for the same configuration of waypoints. It will also reduce the standard deviation of CT. This issue will be addressed by further research.

Paths dynamically calculated by LPP significantly deviate from its initial form due to the randomness of RRT*. However, it will be less severe in the final application when integrated with GPP, which will place the neighboring global waypoints closer to each other. Moreover, optimality of RRT* paths can be improved by allowing the algorithm to continue after a valid path was found. This way, RRT* generates new vertices of the random tree, possibly optimizing the path until a given condition, e.g., the number of RRT iterations, is met. This, however, would increase CT. Nevertheless, it may be included as an alternative to the first valid path

Tab. 5.23: Metrics computed for the results of AdaptiveLPP3

#	CT [s]	FT [s]	LEN [m]	SMOO	NTREE	NRRT	NPTS	NCOL
1	0.13 + 0.24	1528.079	33618	0.690	3 + 11	4 + 4	148 + 299	0
2	0.10 + 0.20	1591.295	35008	0.676	7 + 7	5 + 4	348 + 311	0
3	0.07 + 0.08	1041.828	22920	0.740	6 + 4	5 + 4	291 + 195	0
4	0.10 + 0.40	1610.016	19739	0.677	8 + 21	4 + 4	207 + 319	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
9	0.92 + 0.13	870.562	19152	0.791	61 + 5	4 + 4	245 + 157	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
13	0.20 + 0.13	1629.665	35853	0.701	11 + 7	4 + 4	307 + 323	0
14	0.14 + 0.08	535.885	11789	0.961	12 + 6	4 + 3	358 + 86	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
19	0.10 + 0.08	1020.618	22454	0.739	3 + 5	4 + 3	149 + 190	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
24	0.09 + 0.21	2042.083	44926	0.601	5 + 10	3 + 4	152 + 409	0
25	0.04 + 0.18	1577.559	34706	0.666	3 + 14	4 + 4	146 + 308	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
30	0.12 + 0.11	915.155	20133	0.804	8 + 7	5 + 4	215 + 167	0

approach used in this research.

None of the planned paths resulted in collision. A collision occurred, however, during simulation (see Fig. 5.28). The UAV simulated using the kinematic guidance model deviated from the planned path while performing relatively tight turn, which resulted in a collision with terrain. To avoid such events in the future:

- Safety margin of LPP can be increased to provide wider tolerance and take into account the position errors caused by the model.
- The current model should be calibrated using diverse test paths, which include tight turning maneuvers, such as sine-shaped path.
- Kinematic constraints of LPP should be tighter than the constraints of the model.
- A more feasible model may be used.

5.7 Adaptive Path Planning for pollution sampling

The last part of the verification study is devoted to APP working as a whole. Both GPP and LPP are verified in two compound missions, which mimic the real use cases of APP as the path planner for a meteorological UAV:

- **Smog profiling over Żywiec** addresses a low-emission measurement scenario over the second most polluted city in Poland.
- **Black carbon concentration over Kongsvegen** describes a mission in Svalbard, which scope is to measure the BC concentration over the Kongsvegen glacier.

The cases base on the rough measurement scenarios described in sections 1.1.1 and 1.1.2.

5.7.1 Modifications of criteria

While conducting preliminary research on APP, it was found that the heuristic wind influence criterion (see section 4.4.2) has a little effect on the quality of the final path. Moreover, the energy expenditure criterion (see section 16), which neglected wind influence entirely could benefit from wind data. Hence, the criteria were mixed together.

The new energy expenditure criterion use precomputed energy data as in eq. 4.13. However, it computes the estimated flight time using v_g instead of v_a, where v_g is computed from the wind triangle (2.2) using wind velocity from a wind map.

It is still a heuristic approach, though. In this criterion wind has no influence on the direction of $\vec{v}_g$, so its influence is modeled only by modifying the value of v_g (to speed up or slow down the UAV, thus modifying energy expenditure). To prevent infinite estimated energy when v_g reaches 0, the value of v_g is clamped to a small positive value.

5.7.2 Smog profiling over Żywiec

The first compound use case addresses smog measurements performed by flying in zigzag pattern over a city. According to the WHO air pollution database, in 2018 the city of Żywiec was rated the 2nd most polluted city in Poland with the PM2.5 level of over $40\mu\text{g/m}^3$. At the same time, Żywiec was the 4th most polluted city in the EU [60]. High air pollution, mountainous terrain and close proximity to an airport make Żywiec an interesting use case to plan a low emission (smog) profiling mission with APP.

The map, which depicts the mission is shown in Fig. 5.30 – the satellite image was acquired from Google Earth (`https://earth.google.com/web/`). The mission starts and ends over the nearby Żar Airport. The airport is located on the slope of Żar mountain (Little Beskids in southern Poland), north of the city. The volume just above the city[9] is considered an NFZ. The UAV must fly to the point of the highest predicted pollution concentration and then fly along a zigzag-shaped path over a predefined measurement polygon. While flying back, an unexpected error occurs and communication with the GCS as well as the current path are lost. Now, the UAV has to recompute the returning path to the airport locally. The detailed scenario of this mission is presented in Tab. 5.24.

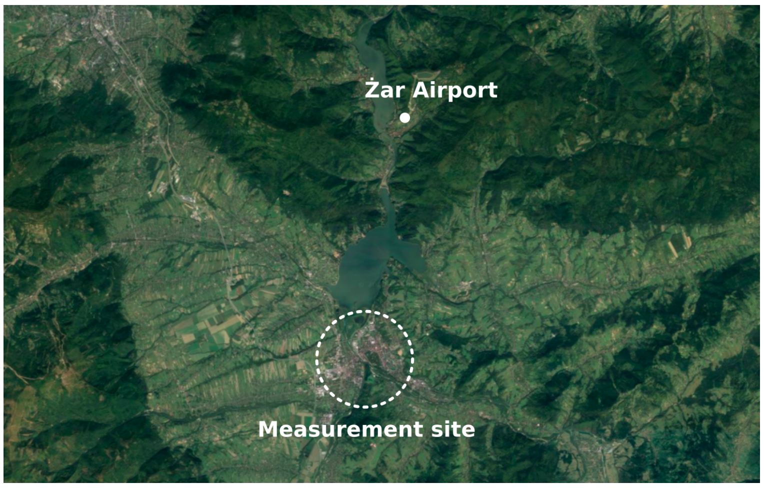

Fig. 5.30: Measurement mission over Żywiec

The success criteria were computed as in the test scenarios of LPP (see Tabs. 5.18, 5.20 and

[9] That is, counting from the highest ground elevation measured at the area of the city. The infrastructure is avoided by applying vertical safety margin.

5.22). However, to account for changes in altitude present in this case and slow climb rate of the UAV, FT was estimated using equation

$$\text{FT} \leqslant \frac{s_m}{v_a}\sum_{n=1}^{N-1}\left(\sqrt{(w_n^x - w_{n+1}^x)^2 + (w_n^y - w_{n+1}^y)^2} + \frac{1}{\sin(\gamma_n)}(w_n^z - w_{n+1}^z)\right)$$
$$\gamma_n = \begin{cases}\gamma_{min} \text{ if } w_n^z - w_{n+1}^z \\ \gamma_{max} \text{ otherwise}\end{cases} \tag{5.11}$$

where γ_{min} and γ_{max} are the minimum and maximum inertial-referenced flight-path angle as in Tab. 5.13. Variable s_m is safety margin. The base value of s_m is 2.0 to account for wind. It is then increased by 0.5 for each measurement site during the flight. Hence, in this case $s_m = 2.5$.

Simplified static case

To compare paths planned by APP with a reference path designed by a human expert, the scenario from Tab. 5.24 was simplified by removing dynamic re-planning. Thus, the simplified scenario uses only steps 1 to 4. Fig. 5.31 shows the reference human-designed path (top figure) and the worst (longest in terms of FT) path planned by GPP (bottom figure). Thin yellow line denotes the planned (theoretical) path, while the red thick line is the simulated path. Red and green tetrahedrons depict start and goal, respectively. The blue sphere indicates the region with the highest predicted pollutant concentration and the red prism depicts the volume above Żywiec, that counts as an obstacle.

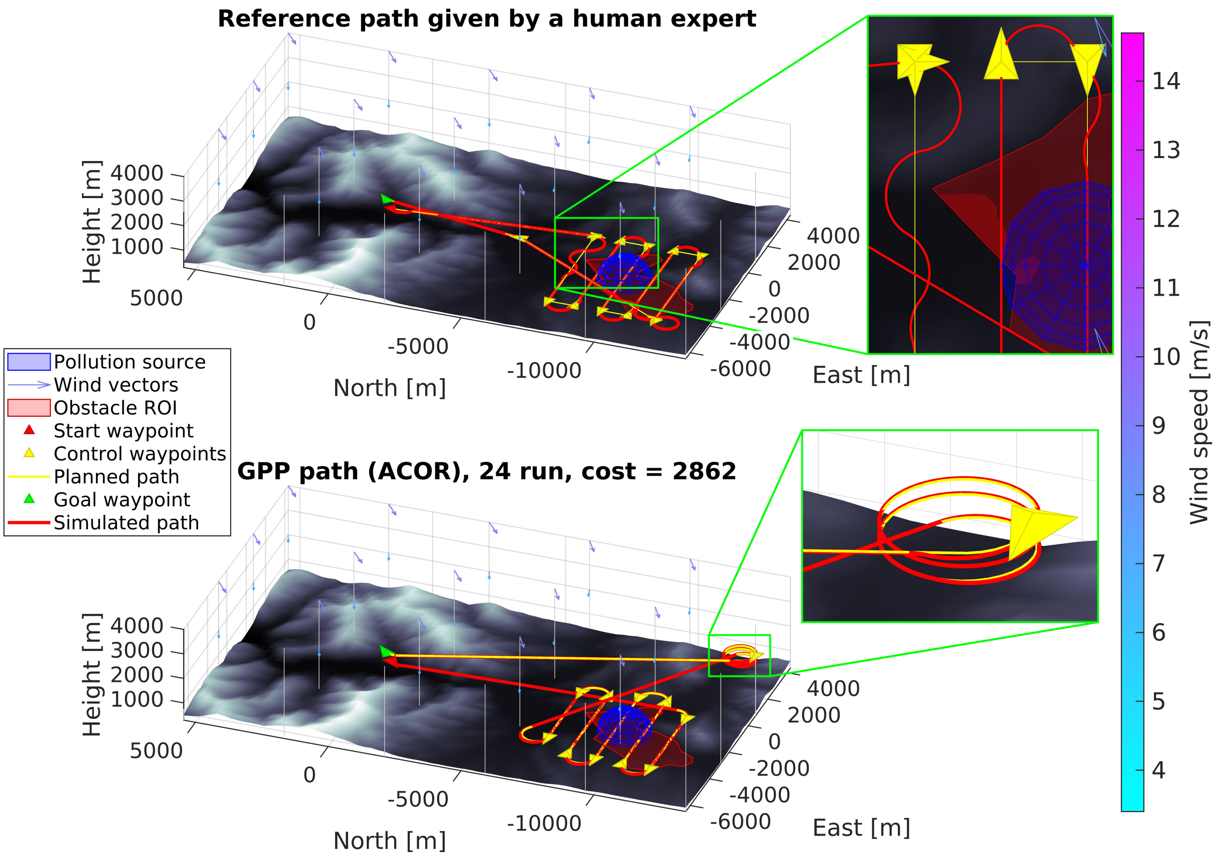

Fig. 5.31: The expert-provided path (top) and the best GPP-generated path (bottom)

The experiment was repeated 30 times with different seed of the random generator. The results are summarized in Tab. 5.25. FT , LEN , SMOO , EEE and NCOL were computed

Tab. 5.24: Detailed scenario of the smog profiling mission over Żywiec

Scenario ID	APPLowEmissionZywiec
Description	Smog profiling over Żywiec
Goal	UAV starts over Żar Airport (ICAO:EPZR). GPP optimizes the path from start to the measurement site and back to goal. Then, the flight along the path is simulated. After 1800 s of simulated flight time the UAV must plan a new fallback path (an RTH maneuver) using onboard LPP. Then, the simulation resumes and the aircraft flies back to the Żar Airport uninterrupted.
Stages	1. The aircraft is airborne at the start waypoint, airspeed = 22 $\frac{\mathrm{m}}{\mathrm{s}}$. 2. The UAV flies to the region of the maximum pollutant concentration. 3. The UAV performs the measurement following a zigzag-shaped path. 4. The aircraft flies towards the goal waypoint. 5. 1800 s after the start, error occurs and communication is lost. 6. The path is re-computed locally and the UAV performs an RTH. 7. The aircraft continues along a new path towards the goal waypoint.
Map	Rectangle from (49.650000°N, 19.130000°E) to (49.820000°N, 19.280000°E), local origin $(0, 0, 0)$ at (49.685000°N, 19.190000°E, 348 m AMSL†), wind forecast from t_0 to $t_0 + 12$ h
UAV model	Fixed-wing kinematic guidance model calibrated as in section 5.4.3
Algorithms	$\mathrm{ACO}_{\mathbb{R}}$ (GPP), RRT* calibrated as in section 5.6.4 (LPP)
Mission parameters	Kinematic constraints of the UAV as in Tab.5.5 Max connection distance = 3000 m Start waypoint = $(-400, -200, 518, -\frac{3}{4}\pi)$ Goal waypoint = $(100, 200, 544, \frac{\pi}{4})$ Safety margin = 200 m (horizontal), 100 m (vertical)
Pollution model	Explicit linear as in eq. 3.14, cost growth rate = 0.005 m^{-1} from the center Center at (49.685000°N, 19.190000°E, 348 m AMSL)

Criteria of success	Total FT $\leqslant$ 2964 s
	NCOL = 0 (GPP and LPP)
	max(CT) $\leqslant$ 4.55 s (only for LPP)

† Actually, the airport is located at 1291 ft (394.5 m) AMSL [61]. However, zero altitude was used to keep the vertical scale relative to sea level.

from simulation data. CT , CFE and COST used the output of GPP. Due to low simulation time step and high density of the waypoints, SMOO was calculated by taking every 500th waypoint. The last row (**Ref**) shows metrics of the reference path. CT uses the plus notation introduced in Tab.5.19. Here, the first term refers to the global path and the second to the local one.

25 of 30 paths met the success criteria (83.33% of runs). For all paths NCOL > 0. Paths 4, 5, 10, 22 and 24 exceeded the FT limit with the highest FT $= 3236.482$ s for the 24th path. It was caused by an unnecessary control waypoint, which resulted in a helical ascend (see Fig. 5.31). CT limit is not considered as LPP was not used.

The 24th path is the longest (LEN $= 71025$ m) and least energy-efficient (EEE $= 93.491$ Wh). Path 15 has the smallest FT of 2135.013 s and also the lowest LEN (46868 m) and EEE (61.633 Wh). The 25th path is the smoothest one (SMOO $= 0.887$), while the 4th path is the least smooth (SMOO $= 0.801$).

Compared to the reference path supplied by a human expert, 19 of 30 paths (63.33%) have lower FT, LEN and EEE. Additionally, 28 paths provided smoother paths than the reference (based on simulation data). Medians of the applicable metrics of GPP-generated paths (FT $=$ 2233.620 s, LEN $= 49005$ m, EEE $= 64.467$ Wh and SMOO $= 0.872$) are better than the corresponding metrics of the reference path. Paths with low number of waypoints, such as human-provided paths, are harder for the simulated controller (as in section 5.4.3) to follow, which resulted in oscillations noticeable especially in measurement path (see top right detail in Fig. 5.31).

Tab. 5.25: Metrics computed for the simplified APPLowEmissionZywiec scenario

#	CT [s]	FT [s]	CFE	COST	LEN [m]	SMOO	EEE [Wh]	NCOL
1	30.554	2151.837	15350	3020	47257	0.868	62.116	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
4	95.526	2993.788	35750	2742	65447	0.801	86.473	0
5	87.318	3021.026	40850	2844	66236	0.850	87.191	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	34.790	3024.412	17450	3071	66391	0.825	87.320	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	23.032	2135.013	12050	3232	46867	0.883	61.633	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
22	44.314	2982.508	22850	2847	65386	0.855	86.106	0
23	18.425	2135.032	10250	3233	46867	0.884	61.633	0
24	47.061	3236.482	22850	2862	71025	0.841	93.491	0
25	38.915	2202.007	19550	3023	48372	0.887	63.551	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
30	39.131	2272.361	20150	2805	49857	0.871	65.579	0
Ref	N/A	2488.966	N/A	N/A	54596	0.826	71.857	0

Complete case with local re-planning

Next, the experiment was repeated, this time including stages 5 to 7 from Tab. 5.24. As before, 30 independent runs were computed. Tab. 5.26 summarizes the results.

Paths 14 and 21 violate the FT success criterion with values of 3789.433 s and 2995.963 s, respectively. These paths also are the longest and least energy-efficient. Interestingly, the shortest and most energy-efficient is path 4, which actually has the highest COST = 3233. It was caused by the energy criterion, which does not model wind influence exactly. Thus, EEE during optimization differs from EEE computed for simulation data. This suggests, the energy criterion could be further improved.

Path 28 is the smoothest one, with the 1st path being the opposite. As expected, SMOO shows high negative correlation with EEE (-0.705). FT has the highest absolute correlation to EEE (0.999). The shortest local CT = 0.061 s is noted for the 20th path and the longest (1.030 s) for the 8th one. In the case of global paths, path 25 has the smallest CT = 17.181 s, while for the 2nd path CT is the largest. Hence, no path exceeded the CT limit. Moreover, all paths are collision-free. Finally, 28 of 30 paths met the criteria of success, giving success ratio of 93.33% for the scenario.

Tab. 5.26: Metrics computed for the dynamic APPLowEmissionZywiec scenario

#	CT [s]	FT [s]	CFE	COST	LEN [m]	SMOO	EEE [Wh]	NCOL
1	32.446 + 0.171	2284.034	13550	3183	49678	0.687	67.191	0
2	80.452 + 0.650	2623.316	34250	2703	57491	0.812	75.720	0
3	21.215 + 0.071	2137.449	11150	3232	46923	0.884	61.704	0
4	19.994 + 0.082	2137.448	10550	3233	46923	0.884	61.704	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
8	53.150 + 1.030	2873.110	25850	2805	62944	0.814	82.951	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	50.291 + 0.332	3789.433	24650	2703	82716	0.717	110.512	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
20	38.989 + 0.061	2316.751	17750	3021	50843	0.836	66.877	0
21	28.481 + 0.153	2995.963	13850	3043	65578	0.722	86.647	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
25	17.181 + 0.108	2137.571	9050	3232	46926	0.884	61.708	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
28	20.707 + 0.069	2137.687	11150	3233	46928	0.885	61.711	0
29	24.547 + 0.184	2296.753	12950	3029	50458	0.841	66.312	0
30	21.603 + 0.100	2137.597	12050	3232	46926	0.884	61.708	0

5.7.3 Black carbon concentration over Kongsvegen

The second compound use case considers measuring BC concentration over the Kongsvegen glacier in Svalbard. Two variants of the mission were considered. In the first one, GPP is used once to plan a single measurement mission without re-planning to compare the results with a reference path provided by a human expert. The second one expands the former to a dynamic measurement mission with GPP- and LPP-based re-planning attempts.

The mission is visualized on the map in Fig. 5.32 – the satellite image was provided by `https://www.bing.com/maps/`. The flight starts and ends over the airport in Ny-Ålesund. The UAV must fly to the point of highest predicted pollution concentration located above the glacier and then fly along a zigzag-shaped path inside a predefined measurement polygon. However, after taking the measurements, the personnel in GCS decides to reattempt the measurements in a different place. The mission plan is updated, while the UAV is still airborne. After finishing the second measurement, the UAV flies back to the airport, but the communication with GCS is lost. The aircraft looses track of its previous path and locally plans a path back to the airport. The UAV reaches the goal without further complications.

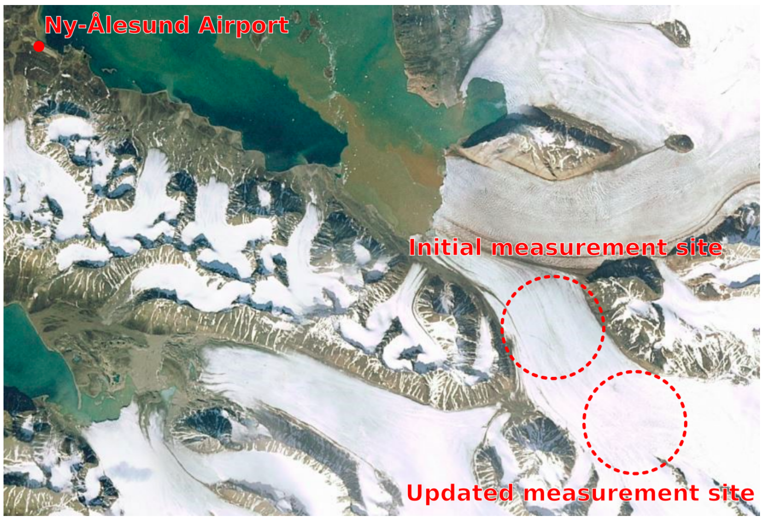

Fig. 5.32: Measurement mission over the Kongsvegen glacier

Details of the scenario are summarized in Tab. 5.27. The success criteria were computed as in previous case. Estimated FT was computed using eq. (5.11). The simplified variant used $s_m = 2.5$. For the complete mission s_m was increased to 3.0 because of the second measurement site. Horizontal safety margin was increased to 500 m to account for stronger wind than in the previous case.

Simplified static case

As in the previous case, before attempting the mission as in Tab. 5.27, a simplified static case was defined to compare the results with a manually designed flight path. The mission was reduced to steps 1, 2, 3 and 7. Hence, only the initial measurement plan was realized with no additional complications. Fig. 5.33 shows the reference path provided by a human expert (top figure) and the best-cost path planned by GPP (bottom figure). Thin yellow line denotes the planned (theoretical) path, while the red thick line is the simulated path.

The human-provided path consists of just a few control waypoints. Exact way of reaching them is left to the simulated onboard low-level controller (see section 5.4.3). Therefore, the reference path is not checked for feasibility. Conversely, GPP-generated path provides densely-sampled waypoints to closely match the shape of the optimized path and handle the avoidance of static obstacles.

Detailed metrics are summarized in Tab. 5.28. All the paths met the success criteria. All generated paths are longer than the reference one, however. Paths 7, 11, 26 and 27 resulted in more than 10% increase in EEE, while the 25th path provides about 5% savings in EEE compared to the reference path. CT ranges from 19.359 s for the 20th path to 70.899 s for path 5 with median of 28.119 s. FT follows the trend of EEE (correlation coefficient is 0.944) with maximum FT = 20360.692 s for the 26th path.

Tab. 5.27: Detailed scenario of the BC measurement mission over Kongsvegen

Scenario ID	APPBlackCarbonKongsvegen
Description	BC measurements over Kongsvegen
Goal	UAV starts over Ny-Ålesund Airport, Hamnerabben (ICAO:ENAS). GPP optimizes the placement of the path from start to the measurement site and back to goal. Then, the flight along the path is simulated. After the first measurements, pollution data is updated and GPP is run again. Then, the UAV follows the new global. After the second measurement series, LPP must plan a new fallback path using onboard LPP. The simulation resumes and the aircraft reaches goal undisturbed.
Stages	1. The aircraft is airborne at the start waypoint, airspeed = 22 $\frac{\text{m}}{\text{s}}$. 2. The UAV flies to the region of the maximum BC concentration. 3. The UAV performs the measurement following a zigzag-shaped path. 4. The global path is updated and the aircraft resumes the mission. 5. The UAV performs the measurement and flies back to the airport. 6. Communication is lost – the UAV locally plans a new path to goal. 7. The aircraft continues uninterrupted to the goal waypoint.
Map	Rectangle from (78.750000°N, 11.800000°E) to (78.940000°N, 13.200000°E), local origin $(0, 0, 0)$ at the Ny-Ålesund Airport (78.927778°N, 11.874722°E, 0 m AMSL), wind forecast from t_0 to $t_0 + 12$ h
UAV model	Fixed-wing kinematic guidance model calibrated as in section 5.4.3
Algorithms	ACO$_{\mathbb{R}}$ (GPP), RRT* calibrated as in section 5.6.4 (LPP)
Mission parameters	Kinematic constraints of the UAV as in Tab.5.5 Max connection distance = 5000 m Start waypoint = $(3850, 4800, 5300, \frac{\pi}{4})$ Goal waypoint = $(22750, 6850, 5500, -\frac{3}{4}\pi)$ Safety margin = 500 m (horizontal), 100 m (vertical)
Pollution model	Explicit linear as in eq. 3.14, cost growth rate = 0.005 m^{-1} from the center Initial center at (78.834200°N, 12.682600°E, 5000 m AMSL) Updated center at (78.834200°N, 12.682600°E, 5000 m AMSL)
Criteria of success	Total FT $\leqslant$ 21125 s (simplified case), total FT $\leqslant$ 26348 s (full case) NCOL $= 0$ (GPP and LPP) max(CT) $\leqslant$ 4.55 s (only for LPP)

None of the generated paths resulted in NCOL > 0. Contrary to that, the simulated path which bases on the reference path misses the global end point and crashes into ground (see top left closeup in Fig. 5.33). In practice, however, the human operator would take control over the UAV as soon as it was in range, so the crash would not happen. Moreover, the simulated aircraft

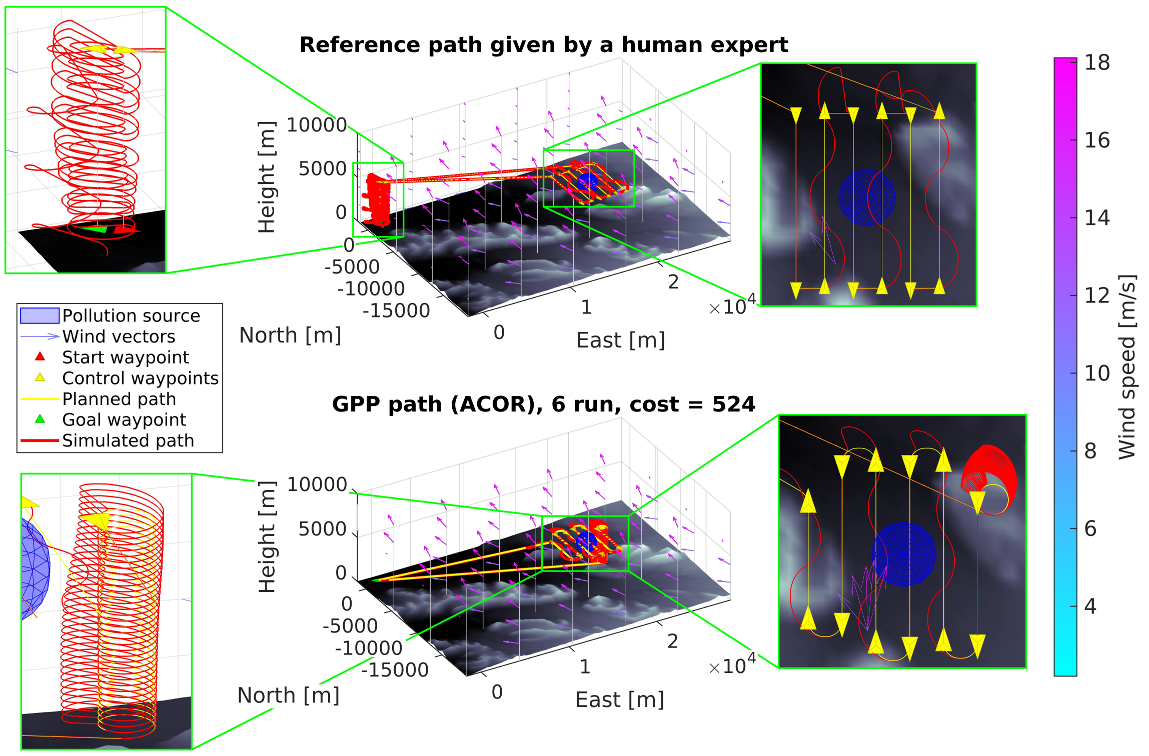

Fig. 5.33: The expert-provided path (top) and the best GPP-generated path (bottom)

was blown off the map while attempting helical ascend and descend close to the map border.

The paths generated by GPP provide comparable results to the reference human-based path. In simulation, these paths tend to be somewhat longer (12.06% based on mean) and less energy-efficient (6.45% based on mean). Nonetheless, GPP automatically generates a densely-sampled waypoints, which provide more predictable and feasible paths.

Complete case with global and local re-planning

The experiment was extended to include dynamic re-planning of both global and local paths, as specified in Tab. 5.27. As in the preceding simplified scenario, 30 independent runs were computed using varying random seed. The results of the complete scenario are given in Tab. 5.26.

Paths 19, 25, 27, 29 violated the FT success criterion. Path 19 is the longest and least energy-efficient ($\text{LEN} = 443148$ m, $\text{FT} = 36065$ s, $\text{EEE} = 1042\ Wh$). All paths met the other criteria, i.e., related to CT and NCOL. Hence, the success ratio of this experiment is 86.67%. The shortest and most energy-efficient is path 23 with $\text{LEN} = 304098$ m, $\text{FT} = 20729$ s and $\text{EEE} = 601$ Wh. It is also the smoothest path with $\text{SMOO} = 0.836$.

5.7.4 Discussion

APP was positively validated on two different use cases inspired by real measurement mission scenarios. The algorithm achieved success ratio from 83.33% to 100%, depending on the use case. The algorithm tends to provide close-to-optimal solution. Nevertheless, in some cases a sub-optimal solution is returned. APP is meant to be used as a guidance tool for a human mission planner, so a small fraction of sub-optimal solutions is a minor issue. However, the algorithm must be improved if fully autonomous application is planned.

$\text{ACO}_{\mathbb{R}}$-based GPP provides results comparable to a path designed by a human expert. For the first use case, most of the GPP-generated paths provided slightly shorter, smoother and

Tab. 5.28: Metrics computed for the simplified APPBlackCarbonKongsvegen scenario

#	CT [s]	FT [s]	CFE	COST	LEN [m]	SMOO	EEE [Wh]	NCOL
1	66.197	17426.153	13850	525	265305	0.771	506.069	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
5	70.899	17424.458	17450	524	265283	0.770	506.020	0
6	24.703	17427.402	11750	524	265320	0.770	506.105	0
7	32.209	18842.038	14750	555	280815	0.782	550.911	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
11	46.187	18499.162	18350	534	282759	0.773	537.934	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
20	19.359	17425.978	10250	525	265304	0.770	506.062	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
25	42.071	15676.260	21050	529	246465	0.795	455.864	0
26	43.427	20360.692	19250	641	302490	0.794	598.328	0
27	58.823	19622.437	23450	588	294537	0.783	572.530	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
30	30.891	17428.969	12950	527	265341	0.770	506.150	0
Ref	N/A	16512.259	N/A	N/A	239218	0.803	481.414	6312

more energy-efficient paths. On the contrary, the second case showed results that are mostly acceptable, but slightly inferior to the human-provided path. Nevertheless, APP was compared to a very limited number of human-generated reference paths. To reliably evaluate APP's performance, the number of reference paths should be expanded to include more diverse mission scenarios as well as a group of several human experts. This will be addressed by the further research.

It can be observed APP performs better for shorter paths with little to no altitude changes and small wind speed. When the wind speed becomes comparable to the UAV flight speed and path length increases[10], the APP returns solutions inferior to the human-provided path. It is caused by differences in path shape during optimization and simulation. The reason is to simplify the impact of wind on energy expenditure. This means that during optimization wind modifies v_g value, but does not change the direction of the UAV, so the shape of the path remains intact. A possible solution of this issue is to use wind-enabled Dubins paths, such as in research conducted by Mittal et al. [62].

Interestingly, the simulation model provide steeper ascend than helices generated by Dubins airplane paths, despite both using the same kinematic constraints as in Tab.5.3. Fig. 5.34, which is a closeup of Fig. 5.33, presents the issue.

The ascending helix[11] generated by the simulated controller for the human-based path (left

[10] That is, mostly due to significant altitude changes requiring helical ascend and descend maneuvers.
[11] The word "helix" is used here informally to name a maneuver consisting of climbing while turning.

Tab. 5.29: Metrics computed for the dynamic APPBlackCarbonKongsvegen scenario

#	CT [s]	FT [s]	CFE	COST	LEN [m]	SMOO	EEE [Wh]	NCOL
1	32.9 + 19.2 + 0.2	22513	16850 + 10850	524 + 214	324064	0.798	653	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
19	22.3 + 29.1 + 0.1	36065	12350 + 14150	526 + 521	443148	0.836	1042	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
23	31.6 + 17.3 + 0.2	20729	16850 + 10250	566 + 209	304098	0.814	601	0
24	32.6 + 16.7 + 0.1	22576	16850 + 10250	782 + 215	324706	0.801	655	0
25	25.7 + 24.8 + 0.1	26401	13850 + 13250	527 + 397	367347	0.825	765	0
26	42.4 + 34.6 + 0.1	22693	23450 + 15950	525 + 308	324166	0.800	658	0
27	43.6 + 21.2 + 0.2	29201	23750 + 12350	523 + 311	387451	0.812	847	0
28	47.5 + 16.4 + 0.1	22514	25250 + 9650	525 + 214	324077	0.798	653	0
29	27.3 + 19.2 + 0.1	29370	13850 + 11150	726 + 233	376827	0.796	853	0
30	42.4 + 17.3 + 0.2	22512	22850 + 10250	521 + 214	324053	0.799	653	0

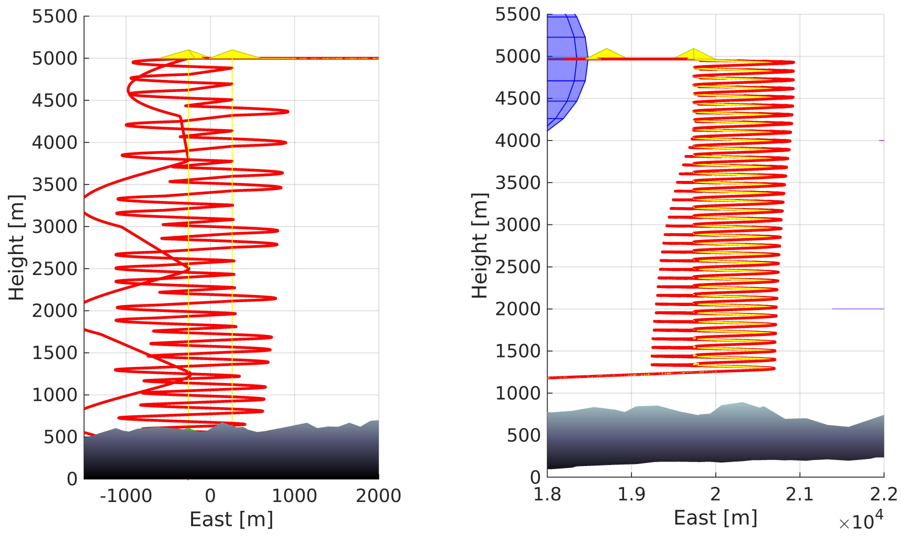

Fig. 5.34: Ascending for a human-provided (left) and GPP-generated (right) paths

figure) has visibly higher pitch versus the right one, which was generated by APP based on Dubins airplane path (right figure). It is caused by the controller trying to follow the path as strictly as possible. Therefore, to mitigate altitude errors, the controller lowers climbing rate if the UAV gets blown away from its theoretical path. Human-provided path consists of just a few waypoints, so this issue is not present.

The problem can be solved in two ways. First is to deform Dubins path according to estimated wind velocity. Second is to explicitly ignore waypoints, which would be missed without decreasing climb rate. Both approaches will be tested in the future research.

The fidelity of the current energy expenditure model is debatable. In the current model, propeller power depends on altitude, airspeed and angle of attack, but the influence of other maneuvers, such as turning is neglected. This is especially visible in the case of yawing induced by the difference between the thrust of both propellers of TS. Introducing a more reliable model could significantly change the output of GPP. For example, climbing on long zigzag-shaped paths might be preferred to helices to limit turning maneuvers.

In its current implementation, LPP provides obstacle-free feasible paths. They are not optimized for energy expenditure, however. Two alternative solutions to this problem are planned.

The first one would use the same approach as in GPP, but with simplified criteria. For example, only collision avoidance and energy expenditure could be considered, as LPP will not be used for measuring pollution and path length is highly positively correlated with energy expenditure.

The second approach would be to employ the probabilistic completeness feature of RRT*. It means to continue searching for better paths after the algorithm finds the first feasible solution. While not addressing energy expenditure directly, this should provide significantly shorter paths. Moreover, as in the case of GPP, a wind-enabled Dubins path implementation can be used, e.g., based on [62].

5.8 Summary

The chapter covered extensive tests of APP in simulation. First, the Twin Stratos (TS) fixed-wind UAV was briefly described, followed by metrics used during the verification study. Then, chosen components of the environment map were tested. 2.5D elevation-based terrain map was compared with its 3D voxel-based implementation, proving the better performance of the former and the wind map model was compared against wind data available through free web services.

Next, the final form of the kinematic model and the controller were described. They were calibrated using TS specification and verified on a simple helical path in simulation.

Verification of GPP started from testing the effects of the criteria. Each criterion was tested independently to show its effect on the resulting optimized global path. Then, the optimal single-objective optimization algorithm was chosen through a comparative analysis of four different nature-inspired algorithms: I-GWO, $ACO_{\mathbb{R}}$, PSO and GA. The comparison in a complex abstract scene shown the superiority of $ACO_{\mathbb{R}}$, while also pointing up its major disadvantage – relatively high variability of the results.

LPP was verified afterwards. Three RRT-based algorithms were compared on a similar abstract map as before. The tests were carried out using several randomly generated configurations of the scene to handle random nature of the algorithms. The tests highlighted RRT* as the optimal solution, which significantly outpaced BiRRT and RRT. RRT* was then calibrated and validated on three real flight scenarios that required dynamic re-planning. The tests pointed out inconsistency of the results as the major disadvantage of RRT algorithms. This could be potentially counteracted by RRT*'s probabilistic completeness feature, but is yet to be tested in the further research.

Finally, compound tests of APP were carried out on two real measurement mission scenarios: smog profiling over Żywiec and BC measurements on Svalbard. Each case begun with the comparison of APP-generated paths versus a reference path supplied by a human expert. Then, the complete mission scenario was simulated, which employed dynamic re-planning features of GPP and LPP. The tests showed that the paths generated by APP achieve performance comparable to the human-provided paths. Most of the paths generated by APP fulfilled the success criteria defined for each use case and none provided an path, that resulted in collisions.

GPP was found to be a solid alternative to a human expert, though not ready yet for completely autonomous flights, i.e., without human supervision. LPP fulfilled its task of providing obstacle-free fallback paths in case of communication errors with GCS. LPP-generated paths are not optimized for minimal energy expenditure, tough.

6. Summary

The thesis addresses the problem of adaptive path planning for pollution sampling with a UAV flying in a limited environment. The research was inspired by the LEADER project, which targets air pollution measurements using an autonomous HALE aircraft. The UAV is used to autonomously fly through potentially polluted ares and to measure the pollutant intensity and distribution.

The HALE aircraft developed for the project has very limited energy resources, however. Thus, it is crucial to plan and optimize its flight path for minimal energy expenditure. The HALE airplane is subject to much more restrictive kinematic constraints than, e.g., a multirotor. Hence, to avoid finding solutions which would later turn out to be sub-optimal, the planner must take into account these limitations already at the optimization stage. Moreover, the scene configuration may change during the flight – be it due to worsening weather, the activation of a new NFZ or difference between estimated and measured pollution distribution. Therefore, the planner must be adaptive.

The goal of this doctoral dissertation was to develop an adaptive path planning algorithm that would solve these issues. The research started from a state-of-the-art analysis, which targeted the specific requirements of the LEADER project. It led to the conception of two-stage adaptive path planner, that is based on the models of the UAV and its environment. This idea ultimately led to formulating the formal basis of the model-based Adaptive Path Planner (APP) consisting of Global Path Planner (GPP) and Local Path Planner (LPP). The planner was then optimized, calibrated and validated on the series of MIL simulations. The verification study highlighted its most prominent advantages, such as eponymous adaptability, and also marked some of its flaws, such as high variability of the results, especially in the case of LPP.

The thesis addressed many concerns of the LEADER project in the field of mechanical engineering. According to the author's opinion, the most important contributions of this doctoral dissertation include:

- The formal description of a novel model-based Adaptive Path Planner (chapter 4), which provides adaptive collision-free paths optimized for minimum energy expenditure and subject to kinematic constraints of the HALE UAV.
- Formulation of the single-objective multi-criteria optimization problem considering obstacle avoidance, minimizing energy expenditure, measurement strategy and kinematic constraints of the UAV.
- Comparative analysis of four global single-objective optimization algorithms (I-GWO, $ACO_{\mathbb{R}}$, PSO and GA) used by GPP for finding the optimal global path.
- Comparative analysis of three stochastic RRT-based planning algorithms (RRT, RRT* and BiRRT) employed by LPP to rapidly compute a fallback local path.
- The formal description of a novel environment model (map). The map models several important aspects of the scene: terrain, current and forecast wind velocity, airspace structure and measurement strategy.
- Extensive MIL tests of APP and its components on several use cases inspired by real pollution measurement missions planned over the course of the LEADER project.

6.1 Conclusions

The research carried out during the thesis has led to the following conclusions:

- As the result of the thesis a model-based adaptive planning algorithm was developed. After formulating the formal basis of APP, the algorithm was positively verified on a limited number of test cases. The algorithm provides a feasible path for an autonomous HALE UAV flying in a limited environment.
- Implementing APP as two separate algorithms enables fine-tuning each of them to fulfill their specific tasks. GPP is meant for global mission planning and optimizing the resulting path for minimal energy expenditure. LPP must provide a feasible path in limited time. This path is used as a fallback and does not have to be optimal in terms of energy expenditure.
- GPP generates paths which are collision-free and optimized to minimize energy expended by the UAV. The quality of the paths is comparable with paths provided by a human expert. GPP gives better results for shorter path, especially if wind speed is low. For longer path with significant wind speed and large changes in altitude, the results are acceptable, but inferior to a path generated by a human.
- Dubins airplane paths provide optimal paths as long as wind influence can be neglected. If wind speed approaches airspeed resulting in significant deformations of the flight path, wind-enabled model must be used. Difference between the final path and the theoretical path expected by the controller is the major cause of sub-optimality of APP-provided paths under windy conditions.
- $\mathrm{ACO}_{\mathbb{R}}$ was found the most suitable global path planning algorithm of the four nature-inspired global optimization algorithms considered in the thesis. The others included I-GWO, PSO and GA. However, the major drawback of $\mathrm{ACO}_{\mathbb{R}}$ is the high variability of path quality. For example, GA provides slightly inferior, but more consistent results. Hence, GA may become the optimal solution if APP is meant to be used for fully autonomous path planning.
- The criterion used to estimate energy takes into account the effect of wind speed on flight time (indirectly), but not the shape of the path. As a result, the criterion becomes unreliable if wind speed approaches airspeed.
- The energy expenditure and minimal path length criteria provide similar results, so using them simultaneously is redundant. Nonetheless, each criterion has different features. The more accurate energy-based criterion is preferred if computation time is not an issue. Otherwise, the length-based criterion should be used.
- By zeroing the pollution intensity criterion, APP can be employed to plan energy-optimized point-to-point paths. This way, the algorithm remains usable outside its primary application.
- LPP dynamically recalculates local paths, providing new kinematically-feasible and obstacle free paths. While feasible, this path is rarely optimal in terms of minimal energy expenditure, flight time or path length.
- Due to stochastic nature of RRT-based algorithms, the output of LPP differs significantly even for the same initial conditions (excluding random seed). Therefore, frequent recalculations of the local path lead to the final flight path being sub-optimal. a possible solution is to exploit the probability completeness of RRT* by allowing the algorithm to search for better connections after finding a feasible solution.

- RRT* was found the most suitable local path planning algorithm of the three RRT-based algorithms considered in the thesis. The others included RRT and BiRRT. This could be improved by allowing RRT* to continue the optimization after a feasible path is found.
- Kinematic guidance model used for MIL tests is useful to roughly estimate the flight of the UAV at lower altitudes. However, as altitude increases, it must be replaced with a dynamic model which takes into account lowering air density and its effects on lift force.
- A truly model-based adaptive path optimization, i.e., using the UAV model to calculate the cost function, has not been achieved yet. Simulating the path provided by every candidate solution significantly increases the optimization time, so the planner is no longer usable for dynamic events. In its current implementation, the model of the UAV (either kinematic or dynamic) can be used only to validate the path planned using, e.g., Dubins paths or heuristic rules. Nevertheless, the environment model plays a significant role during optimization.
- Elevation-based 2.5D terrain representation was found the optimal terrain model for APP. The elevation-based model provides higher computation performance compared to the voxel-based one. For outdoor applications that do not model ceiling or complex shapes the former model is preferred.

6.2 Future remarks

The non-exhaustive list of remarks to be considered in the future research is as follows:

- APP may be further expanded to solve the Zermelo problem (i.e., including wind) instead of Dubins. For that purpose a wind-enabled model will be employed. For example, the 2D model by Mittal et al. [62] can be used as a base and then expanded to 3D.
- Expanding wind maps with vertical air currents will allow utilizing the soaring technique [41], i.e., staying inside the masses of warm air, thus further improving flight efficiency.
- Discrete wind maps, as well as measurement maps, can be modeled as octrees, similarly to voxel-based terrain maps. Thus, the size of the maps can be reduced – especially for maps with high sampling resolution. Moreover, the concept of voxel-based terrain map can be expanded by reintroducing probabilities as described in [36] instead of binary values to represent cell occupancy.
- The $SMOO$ metric is less informative for densely-sampled paths. For example, a straight line path with a single 180° turn in the middle would still have $SMOO \approx 1$, if sampling density is large enough. Hence, the definition of $SMOO$ should be reworked to provide a more robust metric.
- The concept of model-based path validation will be expanded to the dynamic guidance model (2.4 and 2.5), and then to the full non-linear dynamic model of TS.
- The research lacks multi-objective optimization approach. Future research will compare the effects of the single-objective optimization methods from the thesis with their multi-objective counterparts, e.g., multi-objective GA.
- The thesis used only MIL verification. Further research will be extended to SIL verification, and eventually PIL and HIL validation when TS is completed.
- An alternative non-stochastic version of LPP will be developed. It will use the concept of 3D visibility graphs found in [63–65].

Bibliography

[1] E. Dhulkefl, A. Durdu, and H. Terzioğlu, "DIJKSTRA ALGORITHM USING UAV PATH PLANNING," *Konya Mühendislik Bilimleri Dergisi*, vol. 8, pp. 92–105, 2020. `https://doi.org/10.36306/konjes.822225` [25.12.2021].

[2] L. Yang, J. Qi, J. Xiao, and X. Yong, "A literature review of UAV 3D path planning," in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pp. 2376–2381, 2014. `https://doi.org/10.1109/WCICA.2014.7053093` [27.12.2021].

[3] Y. V. Pehlivanoglu, O. Baysal, and A. Hacioglu, "Path planning for autonomous UAV via vibrational genetic algorithm," *Aircraft Engineering and Aerospace Technology*, 2007. `https://doi.org/10.1108/00022660710758222` [28.12.2021].

[4] C. Cifaldi, L. N. Mascarello, and F. Quagliotti, *Regulations: The European Way*, pp. 1–29. Cham: Springer International Publishing, 2018. `https://doi.org/10.1007/978-3-319-32193-6_159-1` [27.11.2021].

[5] Y. Zhao, Z. Zheng, and Y. Liu, "Survey on computational-intelligence-based UAV path planning," *Knowledge-Based Systems*, vol. 158, pp. 54–64, 2018. `https://doi.org/10.1016/j.knosys.2018.05.033` [26.12.2021].

[6] F. Schøler, A. la Cour-Harbo, and M. Bisgaard, "Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference, Portland, OR, USA*, pp. 8–11, Citeseer, 2011.

[7] M. Garcia, A. Viguria, and A. Ollero, "Dynamic Graph-Search Algorithm for Global Path Planning in Presence of Hazardous Weather," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1, pp. 285–295, 2013.

[8] H. Duan, Y. Yu, X. Zhang, and S. Shao, "Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm," *Simulation Modelling Practice and Theory*, vol. 18, no. 8, pp. 1104–1115, 2010. `https://doi.org/10.1016/j.simpat.2009.10.006` [16.04.2022].

[9] X. Ling and Y. Hao, "Effective 3-D Path Planning for UAV in Presence of Threat Netting," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 1298–1302, 2015. `https://doi.org/10.1109/CSNT.2015.217` [16.04.2022].

[10] R. P. Anderson and D. Milutinović, "A Stochastic Approach to Dubins Vehicle Tracking Problems," *IEEE Transactions on Automatic Control*, vol. 59, no. 10, pp. 2801–2806, 2014. `https://doi.org/10.1109/TAC.2014.2314224` [26.11.2021].

[11] A. Rahmani, X. C. Ding, and M. Egerstedt, "Optimal Motion Primitives for Multi-UAV Convoy Protection," in *2010 IEEE International Conference on Robotics and Automation*, pp. 4469–4474, 2010. `https://doi.org/10.1109/ROBOT.2010.5509221` [27.11.2021].

[12] J. Meng, S. Kay, A. Li, and V. M. Pawar, "UAV Path Planning System Based on 3D Informed RRT* for Dynamic Obstacle Avoidance," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1653–1658, IEEE, 2018. `https://discovery.ucl.ac.uk/id/eprint/10121867/3/Pawar_ROBIO_paper_2018.pdf` [24.12.2021].

[13] C. R. Páez, "Simulations of High-Altitude Long-Endurance platforms in Global System Identification," *German Aerospace Center (DLR), Robotics and Mechatronics Center, Tech. Rep*, vol. 6, 2017.

[14] "Optimal Design of a UAV Based Atmospheric Pollution Profiling System," tech. rep., Norwegian Research Centre (NORCE), 2020.

[15] EU Action on Black Carbon in the Arctic, 2019, "Review of Observation Capacities and Data Availability for Black Carbon in the Arctic Region: EU Action on Black Carbon in the Arctic – Technical Report 1," vol. 4, p. 35, December 2019. `https://www.amap.no/documents/download/6033/inline` [28.11.2021].

[16] S. Gilardoni, A. Lupi, M. Mazzola, D. M. Cappelletti, B. Moroni, L. Ferrero, P. Markuszewski, A. Rozwadowska, R. Krejci, P. Zieger, *et al.*, "Atmospheric black carbon in svalbard (abc svalbard)," *SESS report*, pp. 200–201, 2019.

[17] "Black carbon and ozone as Arctic climate forcers," *Arctic Monitoring and Assessment Programme (AMAP)*, vol. 7, p. 116, 2015. `https://www.amap.no/documents/download/2506/inline` [28.11.2021].

[18] D. Vallot, R. Pettersson, A. Luckman, D. Benn, T. Zwinger, W. Pelt, J. Kohler, M. Schäfer, B. Claremar, and N. Hulton, "Basal dynamics of Kronebreen, a fast-flowing tidewater glacier in Svalbard: non-local spatio-temporal response to water input," *Journal of Glaciology*, vol. 63, pp. 1–13, 11 2017. `http://dx.doi.org/10.1017/jog.2017.69` [30.12.2021].

[19] M. Fengler, "Meteomatics Weather API Connector," 2022. `https://www.mathworks.com/matlabcentral/fileexchange/63992-meteomatics-weather-api-connector` [27.06.2022].

[20] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice.* Princeton University Press, 2012. `https://doi.org/10.1515/9781400840601` [09.11.2021].

[21] "Military specification (MIL)-F-8785C. Flying Qualities of Piloted Airplane," 5 November 1980.

[22] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957. `https://doi.org/10.2307/2372560` [26.11.2021].

[23] A. M. Shkel and V. Lumelsky, "Classification of the Dubins set," *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–202, 2001. `https://doi.org/10.1016/S0921-8890(00)00127-5` [26.11.2021].

[24] T. McLain, R. W. Beard, and M. Owen, "Implementing Dubins Airplane Paths on Fixed-wing UAVs," 2014.

[25] 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, 2007. `https://doi.org/10.1109/CDC.2007.4434966` [27.11.2021].

[26] S. Hota and D. Ghose, “Optimal Geometrical Path in 3D with Curvature Constraint,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 113–118, 2010. `https://doi.org/10.1109/IROS.2010.5653663` [27.11.2021].

[27] C. Hanson, J. Richardson, and A. Girard, “Path Planning of a Dubins Vehicle for Sequential Target Observation with Ranged Sensors,” in *Proceedings of the 2011 American Control Conference*, pp. 1698–1703, 2011. `https://doi.org/10.1109/ACC.2011.5990964` [26.11.2021].

[28] Y. Bestaoui Sebbane, *Motion Planning*, pp. 59–170. Cham: Springer International Publishing, 2014. `https://doi.org/10.1007/978-3-319-03707-3_2"` [04.12.2021].

[29] J. A. Reeds and L. A. Shepp, “Optimal paths for a car that goes both forwards and backwards,” *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[30] C. Yong and E. J. Barth, “Real-time dynamic path planning for dubins’ nonholonomic robot,” in *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 2418–2423, 2006. `https://doi.org/10.1109/CDC.2006.377829` [26.11.2021].

[31] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino, “Path Tracking Control for Dubin’s Cars,” in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3123–3128 vol.4, 1996. `https://doi.org/10.1109/ROBOT.1996.509187` [26.11.2021].

[32] R. V. Cowlagi and P. Tsiotras, “Shortest Distance Problems in Graphs Using History-Dependent Transition Costs with Application to Kinodynamic Path Planning,” in *2009 American Control Conference*, pp. 414–419, 2009. `https://doi.org/10.1109/ACC.2009.5160149` [26.11.2021].

[33] 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. `https://doi.org/10.1002/zamm.19310110205` [18.12.2021].

[34] G. Yang and V. Kapila, “Optimal path planning for unmanned air vehicles with kinematic and tactical constraints,” in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 2, pp. 1301–1306, 2002. `https://doi.org/10.1109/CDC.2002.1184695` [15.04.2022].

[35] M. Rexer and C. Hirt, “Comparison of free high resolution digital elevation data sets (ASTER GDEM2, SRTM v2.1/v4.1) and validation against accurate heights from the Australian National Gravity Database,” *Australian Journal of Earth Sciences*, vol. 61, no. 2, pp. 213–226, 2014. `https://doi.org/10.1080/08120099.2014.884983` [21.02.2022].

[36] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: an efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[37] MathWorks, Inc., “Documentation of occupancyMap3D in MATLAB Help Center.” `https://www.mathworks.com/help/nav/ref/occupancymap3d.html` [23.01.2022].

[38] M. Kosior, “Wind forecast map for adaptive path planning with an unmanned aerial vehicle,” in *Proceedings of Metody Komputerowe – 2022*, (Gliwice, Poland), p. TBD, 6 2022. (Accepted for publication).

[39] B. W. Kerns and S. S. Chen, "ECMWF and GFS model forecast verification during DYNAMO: Multiscale variability in MJO initiation over the equatorial Indian Ocean," *Journal of Geophysical Research: Atmospheres*, vol. 119, no. 7, pp. 3736–3755, 2014. https://doi.org/10.1002/2013JD020833 [24.04.2022].

[40] F. A. d. A. Andrade, "Real-time and offline path planning of Unmanned Aerial Vehicles for maritime and coastal applications." http://hdl.handle.net/11250/2640229 [28.12.2021].

[41] S. Tabor, I. Guilliard, and A. Kolobov, "ArduSoar: an Open-Source Thermalling Controller for Resource-Constrained Autopilots," 2018. https://arxiv.org/abs/1802.08215 [23.12.2021].

[42] Polish Air Navigation Services Agency (PANSA), "Official AUP/UUP map for Poland." https://airspace.pansa.pl [27.04.2022].

[43] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon* (M. H. Hebert, C. Thorpe, and A. Stentz, eds.), pp. 203–220, Boston, MA: Springer US, 1997. https://doi.org/10.1007/978-1-4615-6325-9_11 [27.12.2021].

[44] O. Alvear, N. R. Zema, E. Natalizio, and C. T. Calafate, "Using UAV-Based Systems to Monitor Air Pollution in Areas with Poor Accessibility," *Journal of Advanced Transportation*, 08 2017. https://doi.org/10.1155/2017/8204353 [14.05.2022].

[45] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization.* Mit Press, 2019.

[46] M. Torres, D. A. Pelta, J. L. Verdegay, and J. C. Torres, "Coverage path planning with unmanned aerial vehicles for 3D terrain reconstruction," *Expert Systems with Applications*, vol. 55, pp. 441–451, 2016. https://doi.org/10.1016/j.eswa.2016.02.007 [31.05.2022].

[47] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1155–1173, 2008. https://doi.org/10.1016/j.ejor.2006.06.046 [03.06.2022].

[48] J. D. Anderson, *Aircraft performance and design*, vol. 1. WCB/McGraw-Hill Boston, 1999.

[49] S. Park, J. Deyst, and J. How, "A New Nonlinear Guidance Logic for Trajectory Tracking," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, p. 4900, 2004.

[50] S. Guillén Ruiz, L. V. Calderita, A. Hidalgo-Paniagua, and J. P. Bandera Rubio, "Measuring Smoothness as a Factor for Efficient and Socially Accepted Robot Motion," *Sensors*, vol. 20, no. 23, 2020. https://doi.org/10.3390/s20236822 [10.05.2022].

[51] N. D. M. Ceballos, J. A. Valencia, and N. L. Ospina, "Quantitative Performance Metrics for Mobile Robots Navigation," in *Mobile Robots Navigation* (A. Barrera, ed.), ch. 24, Rijeka: IntechOpen, 2010.

[52] MathWorks, Inc., "Documentation of fixedwing in MATLAB Help Center." https://www.mathworks.com/help/uav/ref/fixedwing.html [31.01.2022].

[53] S. Mirjalili, "Improved Grey Wolf Optimizer (I-GWO)," 2022. https://www.mathworks.com/matlabcentral/fileexchange/81253-improved-grey-wolf-optimizer-i-gwo [01.06.2022].

[54] S. Mirjalili, "Improved Grey Wolf Optimizer (GWO)." `https://www.mathworks.com/matlabcentral/fileexchange/81253-improved-grey-wolf-optimizer-i-gwo` [10.11.2021].

[55] M. H. Nadimi-Shahraki, S. Taghian, and S. Mirjalili, "An improved grey wolf optimizer for solving engineering problems," *Expert Systems with Applications*, vol. 166, p. 113917, 2021. `https://doi.org/10.1016/j.eswa.2020.113917` [10.11.2021].

[56] M. K. Heris, "ACO for Continuous Domains in MATLAB," 2015. `https://yarpiz.com/67/ypea104-acor` [03.06.2022].

[57] MathWorks, Inc., "Documentation of the Particle Swarm Optimization in MATLAB Help Center." `https://www.mathworks.com/help/gads/particleswarm.html` [17.06.2022].

[58] MathWorks, Inc., "Documentation of the Genetic Algorithm in MATLAB Help Center." `https://www.mathworks.com/help/gads/genetic-algorithm-options.html` [12.06.2022].

[59] MathWorks, Inc., "Documentation of plannerRRTStar in MATLAB Help Center." `https://www.mathworks.com/help/nav/ref/plannerrrtstar.html?searchHighlight=rrtstar&s_tid=srchtitle_rrtstar_1` [21.05.2022].

[60] L. Myllyvirta and E. Howard, "Five things we learned from the world's biggest air pollution database," 2018. `https://unearthed.greenpeace.org/2018/05/02/air-pollution-cities-worst-global-data-world-health-organisation/` [17.06.2022].

[61] DlaPilota.pl web page, "Żar," 2022. `https://lotniska.dlapilota.pl/zar` [23.06.2022].

[62] K. Mittal, J. Song, S. Gupta, and T. A. Wettergren, "Rapid path planning for Dubins vehicles under environmental currents," *Robotics and Autonomous Systems*, vol. 134, p. 103646, 2020. `https://doi.org/10.1016/j.robot.2020.103646` [26.06.2022].

[63] S. Huang and R. S. H. Teo, "Computationally Efficient Visibility Graph-Based Generation of 3D Shortest Collision-Free Path Among Polyhedral Obstacles for Unmanned Aerial Vehicles," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1218–1223, IEEE, 2019.

[64] Y. You, C. Cai, and Y. Wu, "3D Visibility Graph Based Motion Planning and Control," in *Proceedings of the 2019 5th International Conference on Robotics and Artificial Intelligence*, ICRAI '19, (New York, NY, USA), p. 48–53, Association for Computing Machinery, 2019. `https://doi.org/10.1145/3373724.3373735` [26.06.2022].

[65] M. N. Bygi and M. Ghodsi, "3D Visibility Graph," *Computational Science and its Applications, Kuala Lampur*, 2007.

[66] H.-M. Huang, "Autonomy Levels for Unmanned Systems Framework. Volume I: Terminology. Version 2.0," *NIST Special Publication: Gaithersburg, MD, USA*, 2008. `https://www.nist.gov/system/files/documents/el/isd/ks/NISTSP_1011-I-2-0.pdf` [27.11.2021].

[67] International Civil Aviation Organization, "Cir 328, Unmanned Aircraft Systems (UAS)," 2011. `https://www.icao.int/meetings/uas/documents/circular%20328_en.pdf` [27.11.2021].

[68] European Aviation Safety Agency, “Technical Opinion. Introduction of a regulatory framework for the operation of unmanned aircraft,” 2015. https://www.easa.europa.eu/sites/default/files/dfu/Introduction%20of%20a%20regulatory%20framework%20for%20the%20operation%20of%20unmanned%20aircraft.pdf [27.11.2021].

[69] “Online Oxford dictionary.” https://www.oxfordlearnersdictionaries.com [08.11.2021].

[70] N. Melzer, *Human rights implications of the usage of drones and unmanned robots in warfare.* European Parliament, 2013. http://dx.doi.org/10.2861/213 [27.11.2021].

[71] B. T. Clough, “Metrics, Schmetrics! How the Heck Do You Determine a UAV’s Autonomy Anyway?,” tech. rep., Air Force Research Lab Wright-Patterson AFB OH, 2002.

[72] H.-M. Huang, “Autonomy Levels for Unmanned Systems (ALFUS).” Presentation downloaded from https://www.nist.gov/document/alfus-bgpdf [27.11.2021].

[73] M. Protti and R. Barzan, “UAV Autonomy. Which level is desirable? Which level is acceptable? Alenia Aeronautica Viewpoint,” in *Platform Innovations and System Integration for Unmanned Air, Land and Sea Vehicles (AVT-SCI Joint Symposium)*, pp. 12–1–12–12, 2007.

[74] A. Lampe and R. Chatila, “Performance Measure for the Evaluation of Mobile Robot Autonomy,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 4057–4062, IEEE, 2006.

[75] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control.* MIT press, 1992.

[76] B. Hasslacher and M. W. Tilden, “Living Machines,” *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 143–169, 1995.

[77] R. W. Proud, J. J. Hart, and R. B. Mrozinski, “Methods for Determining the Level of Autonomy to Design into a Human Spaceflight Vehicle: A Function Specific Approach,” tech. rep., National Aeronautics and Space Administration Johnson Space Center, 2003.

[78] R. Williams, “Autonomous Systems Overview,” *BAE Systems*, 2008. http://www.aircraftbuilders.com/files/2716/File/BAE_%20Systems_Text_Version.pdf [01.08.2020].

[79] SAE On-Road Automated Vehicle Standards Committee *et al.*, “Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems,” *SAE Standard J.*, vol. 3016, pp. 1–16, 2021.

[80] F. Glover and K. Sörensen, “Metaheuristics,” *Scholarpedia*, vol. 10, no. 4, p. 6532, 2015. http://dx.doi.org/10.4249/scholarpedia.6532 (rev. #149834) [13.12.2021].

[81] X.-S. Yang, “Metaheuristic Optimization,” *Scholarpedia*, vol. 6, no. 8, p. 11472, 2011. http://dx.doi.org/10.4249/scholarpedia.11472 (rev. #91488) [13.12.2021].

[82] S. Karaman and E. Frazzoli, “Incremental Sampling-based Algorithms for Optimal Motion Planning,” *Robotics Science and Systems VI*, vol. 104, no. 2, 2010. https://arxiv.org/pdf/1005.0416.pdf [26.11.2021].

[83] R. C. Arkin, *Behavior-Based Robotics.* Cambridge, Mass.: MIT Press, 1998.

[84] M. J. Matarić, *The Robotics Primer.* Cambridge, Mass.: MIT Press, 2007.

[85] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots.* MIT press, 2004.

[86] W. G. Walter, "An Imitation of Life," *Scientific american*, vol. 182, no. 5, pp. 42–45, 1950. `http://people.csail.mit.edu/brooks/idocs/walter50imitation.pdf`.

[87] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology.* Cambridge, Mass.: MIT Press, 1986.

[88] R. A. Brooks, "New approaches to robotics," *Science*, vol. 253, pp. 1227–1232, 1991.

[89] S. Gopikrishnan, B. Shravan, H. Gole, P. Barve, and L. Ravikumar, "Path Planning Algorithms: A comparative study," *Space Transportation Systems*, 2011.

[90] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. `https://doi.org/10.1007%2FBF01386390` [04.11.2021].

[91] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* Cambridge, Mass.: MIT Press, 3 ed.

[92] J. T. Economou, G. Kladis, A. Tsourdos, and B. A. White, "UAV Optimum Energy Assignment using Dijkstra's Algorithm," in *2007 European Control Conference (ECC)*, pp. 287–292, 2007. `https://doi.org/10.23919/ECC.2007.7068353` [26.12.2021].

[93] E. J. Dhulkefl and A. Durdu, "Path Planning Algorithms for Unmanned Aerial Vehicles," *International Journal of Trend in Scientific Research and Development*, vol. Volume-3, pp. 359–362, 06 2019. `https://doi.org/10.31142/ijtsrd23696` [25.12.2021].

[94] A. Candra, M. A. Budiman, and K. Hartanto, "Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial," in *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, pp. 28–32, 2020. `https://doi.org/10.1109/DATABIA50434.2020.9190342` [27.12.2021].

[95] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. `https://doi.org/10.1109/TSSC.1968.300136` [09.11.2021].

[96] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"," *ACM SIGART Bulletin*, no. 37, pp. 28–29, 1972.

[97] N. J. Nilsson, *The Quest for Artificial Intelligence.* Cambridge University Press, 2009.

[98] G. Zhang and L.-T. Hsu, "A New Path Planning Algorithm Using a GNSS Localization Error Map for UAVs in an Urban Area," *Journal of Intelligent & Robotic Systems*, vol. 94, no. 1, pp. 219–235, 2019. `https://doi.org/10.1007/s10846-018-0894-5` [27.12.2021].

[99] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," in *IJCAI*, vol. 95, pp. 1652–1659, 1995.

[100] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation.* MIT press, 2005.

[101] H. Choset, "Robotic Motion Planning: A* and D* Search." `http://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf` [04.04.2022].

[102] D. Ferguson and A. Stentz, “Field D*: An Interpolation-Based Path Planner and Replanner,” in *Robotics Research* (S. Thrun, R. Brooks, and H. Durrant-Whyte, eds.), (Berlin, Heidelberg), pp. 239–253, Springer Berlin Heidelberg, 2007. `https://doi.org/10.1007/978-3-540-48113-3_22` [27.12.2021].

[103] J. Carsten, D. Ferguson, and A. Stentz, “3D Field D: Improved Path Planning and Replanning in Three Dimensions,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3381–3386, 2006. `https://doi.org/10.1109/IROS.2006.282516` [27.12.2021].

[104] S. Koenig and M. Likhachev, “D* lite,” *AAAI/iaai*, vol. 15, 2002. `https://www.aaai.org/Papers/AAAI/2002/AAAI02-072.pdf` [27.12.2021].

[105] T. Lozano-Pérez and M. A. Wesley, “An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles,” *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979. `https://doi.org/10.1145/359156.359164` [17.04.2022].

[106] N. J. Nilsson, “A mobile automaton: An application of artificial intelligence techniques,” in *Int. Joint Conf. Artif. Intell.*, pp. 509–520, 1969.

[107] M. B. Ignat’yev, F. M. Kulakov, and A. M. Pokrovskiĭ, *Robot-manipulator control algorithms.* Arlington , Virginia: Joint Publications Research Service, 1973.

[108] B. C. Shah and S. K. Gupta, “Speeding Up A* Search on Visibility Graphs Defined Over Quadtrees to Enable Long Distance Path Planning for Unmanned Surface Vehicles,” in *ICAPS*, pp. 527–535, AAAI Press, 2016.

[109] E. Masehian and M. R. Amin-Naseri, “A Voronoi Diagram-Visibility Graph–Potential Field Compound Algorithm for Robot Path Planning,” *Journal of Robotic Systems*, vol. 21, no. 6, pp. 275–300, 2004. `https://doi.org/10.1002/rob.20014` [19.04.2022].

[110] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[111] J. J. Kuffner and S. M. LaValle, “RRT-Connect : An Efficient Approach to Single-Query Path Planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[112] M. V. Ramana, S. A. Varma, and M. Kothari, “Motion Planning for a Fixed-Wing UAV in Urban Environments,” *4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016*, vol. 49, no. 1, pp. 419–424, 2016. `https://doi.org/10.1016/j.ifacol.2016.03.090` [19.12.2021].

[113] S. Karaman and E. Frazzoli, “Sampling-based motion planning with deterministic μ-calculus specifications,” in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 2222–2229, 2009. `https://doi.org/10.1109/CDC.2009.5400278` [29.12.2021].

[114] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Informed sampling for asymptotically optimal path planning,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 966–984, 2018.

[115] X.-S. Yang, *Nature-Inspired Optimization Algorithms.* Oxford: Elsevier, 2014. `https://doi.org/10.1016/C2013-0-01368-0` [28.12.2021].

[116] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* The MIT Press, 04 1992. https://doi.org/10.7551/mitpress/1090.001.0001 [28.12.2021].

[117] Z. Wang, C. Qin, B. Wan, and W. W. Song, "A Comparative Study of Common Nature-Inspired Algorithms for Continuous Function Optimization," *Entropy*, vol. 23, no. 7, 2021. https://doi.org/10.3390/e23070874 [28.12.2021].

[118] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995. https://doi.org/10.1109/ICNN.1995.488968 [28.12.2021].

[119] A. Colorni, M. Dorigo, V. Maniezzo, *et al.*, "Distributed Optimization by Ant Colonies," in *Proceedings of the first European conference on artificial life*, vol. 142, (Paris, France), pp. 134–142, 1991.

[120] D. N. Kumar and M. J. Reddy, "Ant Colony Optimization for Multi-Purpose Reservoir Operation," *Water Resources Management*, vol. 20, pp. 879–898, 10 2006. https://doi.org/10.1007/s11269-005-9012-0 [16.04.2022].

[121] M. Dorigo and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997. https://doi.org/10.1109/4235.585892 [16.04.2022].

[122] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996. https://doi.org/10.1109/3477.484436 [16.04.2022].

[123] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014. https://doi.org/10.1016/j.advengsoft.2013.12.007 [10.11.2021].

[124] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, "Grey wolf optimizer: a review of recent variants and applications," *Neural computing and applications*, vol. 30, no. 2, pp. 413–435, 2018.

[125] F. Kiani, A. Seyyedabbasi, R. Aliyev, M. A. Shah, and M. U. Gulle, "3D Path Planning Method for Multi-UAVs Inspired by Grey Wolf Algorithms," *Journal of Internet Technology*, vol. 22, no. 4, pp. 743–755, 2021. https://jit.ndhu.edu.tw/article/view/2539 [19.04.2022].

[126] S. Vanneste, B. Bellekens, and M. Weyn, "3DVFH+: Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap," in *MORSE 2014 Model-Driven Robot Software Engineering: proceedings of the 1st International Workshop on Model-Driven Robot Software Engineering co-located with International Conference on Software Technologies: Applications and Foundations (STAF 2014), York, UK, July 21, 2014/Assmann, Uwe [edit.]*, no. 1319, pp. 91–102, 2014.

[127] T. Hebecker, R. Buchholz, and F. Ortmeier, "Model-Based Local Path Planning for UAVs," *Journal of Intelligent & Robotic Systems*, vol. 78, no. 1, pp. 127–142, 2015. https://rdcu.be/cLGv7 [19.04.2022].

[128] A. Alihodzic, E. Tuba, R. Capor-Hrosik, E. Dolicanin, and M. Tuba, "Unmanned Aerial Vehicle Path Planning Problem by Adjusted Elephant Herding Optimization," in *2017 25th Telecommunication Forum (TELFOR)*, pp. 1–4, 2017. `https://doi.org/10.1109/TELFOR.2017.8249468` [19.04.2022].

[129] X. Yue and W. Zhang, "UAV Path Planning Based on K-Means Algorithm and Simulated Annealing Algorithm," in *2018 37th Chinese Control Conference (CCC)*, pp. 2290–2295, 2018. `https://doi.org/10.23919/ChiCC.2018.8483993` [19.04.2022].

[130] S. Bortoff, "Path Planning for UAVs," in *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, vol. 1, pp. 364–368, 2000.

Abstract

From military reconnaissance, through conducting measurements in remote locations to package delivery – the growing popularity of Unmanned Aerial Vehicles (UAVs) encourages employing them for various missions. For example, UAVs measure air pollution. A novel approach is to build a High-Altitude Long Endurance UAV able to stay airborne for prolonged amounts of time and use it to measure air pollution over a broad range of altitudes – from hundreds of meters above ground level up to lower stratosphere.

The goal of this doctoral dissertation was to design and verify the model-based adaptive path planning algorithm for pollution sampling with a HALE UAV flying autonomously in a limited environment. Under the course of the thesis Adaptive Path Planner (APP) was developed and positively verified in Model-in-the-Loop (MIL) simulation.

The author proposed and formulated the theoretical foundation of a two-stage model-based APP algorithm consisting of Global Path Planner (GPP) and Local Path Planner (LPP). The output of GPP and LPP can be verified with MIL before deploying it to the UAV. Furthermore, the author presented the concept of an environment map. It consists of a terrain map, a wind map, an airspace map and a measure map. Each of them describes the most prominent components of the scene important for providing the optimal path.

GPP is a global optimization algorithm, which provides an obstacle-free feasible path optimized for minimum energy expenditure while subject to kinematic constraints of the UAV. GPP is meant to be mainly a mission planner that is run on a PC-class workstation in a Ground Control Station (GCS). The algorithm can adaptively recalculate the path.

LPP is a local planning algorithm employed for rapidly computing obstacle-free path in GCS, as well as locally using the UAV onboard computer. LPP is used as a fallback when the communication with GCS is down. Hence, minimal computation time is favored over minimizing energy expenditure. a local path is used, for example, to perform a Return-To-Home (RTH) maneuver or to guide the UAV to an emergency landing spot.

Extensive tests of APP were carried out to compare different flavors of global single-objective optimization algorithms employed by GPP (I-GWO, $\text{ACO}_{\mathbb{R}}$, PSO and GA) and planning algorithms used by LPP (RRT, RRT* and BiRRT). The results were analyzed and the optimal algorithms chosen. Then, their crucial parameters were further calibrated.

Finally, GPP and LPP were integrated and the whole APP was verified in MIL simulation on selected use cases inspired by real pollution sampling missions in Poland and in the Arctic. The comparative analysis of generated paths against reference paths supplied by a human expert showed that APP performance successfully allows it to replace the human. Further research positively verified the adaptive re-planning capability of APP.

Streszczenie

Od wojskowego zwiadu, poprzez badania w trudnodostępnych miejscach, aż po usługi kurierskie – rosnąca popularność bezzałogowych statków powietrznych (BSP) zachęca do stosowania ich podczas różnorodnych misji. BSP wykorzystywane są np. do pomiaru zanieczyszczenia powietrza. Jedno z nowych zastosowań polega na zbudowaniu BSP o dużej długotrwałości lotu (HALE), a następnie wykorzystaniu go do prowadzenia pomiarów w szerokim przedziale wysokości – od kilkuset metrów nad powierzchnią ziemi aż po dolną stratosferę.

Celem niniejszej rozprawy doktorskiej były zaprojektowanie i weryfikacja opartego na modelu, adaptacyjnego algorytmu planowania ścieżki do próbkowania zanieczyszczeń za pomocą BSP klasy HALE latającego autonomicznie w środowisku z ograniczeniami. W ramach pracy zaprojektowano adaptacyjny planer (APP), który następnie pozytywnie zweryfikowano za pomocą metody Model-in-the-Loop (MIL) w symulacjach wykorzystujących model BSP.

Owocem pracy autora jest opis formalny dwuetapowego algorytmu APP, który tworzą planer globalny (GPP) oraz planer lokalny (LPP). Ścieżki wygenerowane przez GPP i LPP mogą być zweryfikowane w symulacji typu MIL przed przesłaniem ich do BSP. W ramach pracy autor przedstawił także koncepcję mapy środowiska wykorzystywanej w optymalizacji i składającej się z map terenu, wiatru, przestrzeniu powietrznej oraz mapy pomiarowej.

GPP jest algorytmem optymalizacji globalnej, który poszukuje wolnej od przeszkód ścieżki, optymalnej pod kątem minimalnego zużycia energii i podlegającej ograniczeniom kinematycznym BSP. GPP to algorytm planujący misję, uruchamiany w stacji kontroli naziemnej (SKN). Algorytm ma też możliwość adaptacyjnego przeplanowywania ścieżki.

LPP to algorytm lokalnego planowania wykorzystywany do szybkiego generowania wolnych od przeszkód ścieżek zarówno w SKN, jak również za pomocą komputera pokładowego BSP. LPP wykorzystywany jest w sytuacjach awaryjnych, gdy zerwana zostaje komunikacja z SKN, dlatego minimalizacja zużycia energii poświęcona jest na rzecz minimalnego czasu obliczeń. Ścieżka lokalna wykorzystywana jest np. podczas powrotu do punktu startu (RTH) oraz doprowadzenia BSP do punktu lądowania awaryjnego.

Przeprowadzono badania porównawcze algorytmów jednokryterialnej optymalizacji globalnej, z których korzysta GPP (I-GWO, ACO$_\mathbb{R}$, PSO i GA) oraz algorytmów planujących wykorzystywanycyh przez LPP (RRT, RRT* i BiRRT). Następnie dostrojono kluczowe parametry wybranych algorythmów. Algorytm APP został zweryfikowany metodą MIL przez symulację wybranych przypadków użycia inspirowanych misjami badawczymi prowadzonymi w Polsce i w Arktyce. Analiza porównawcza ścieżek wygenerowanych przez algorytm względem zaprojektowanych przez eksperta potwierdziła możliwości APP. Dalsze badania udowodniły zdolność algorytmu APP do adaptacyjnego przeplanowywania ścieżki przelotu BSP.

Used hardware and software

The majority of the research was carried out on a personal computer with Ubuntu 20.04 LTS. The algorithms were developed and verified using **MATLAB R2022a** (9.12.0.1884302) 64-bit (glnxa64) licensed under the Academic License (master license: 31464320, the author's PC: 40876907) (`https://www.mathworks.com/products/matlab.html`). Other MATLAB Toolboxes included:

- **UAV Toolbox**, version 1.3,
- **Navigation Toolbox**, version 2.2;
- **Mapping Toolbox**, version 5.3.

Preliminary studies not described here also used:

- **Blender** 2.93.5 licensed under GNU General Public License, `https://www.blender.org`,
- **BlenderGIS** 2.2.6 licensed under GPL-3.0, `https://github.com/domlysz/BlenderGIS`,
- **CloudCompare** 2.11.1 (Anoia) licensed under GNU General Public License,
- **FlightGear** 2020.3.11 licensed under GNU General Public License, `https://www.flightgear.org`,

The computations as well as writing the thesis were carried out on the author's PC. As for computation performance, the PC had Intel Core i7-9750H, NVIDIA GeForce RTX 2060 and 32 GB (DDR4, 2666MHz). Additional software used while writing the thesis included:

- **LibreOffice** 6.0.7.3 licensed under Mozilla Public License, v. 2.0, `https://www.libreoffice.org`,
- **TEXstudio** 2.12.22 licensed under GNU General Public License v. 2.0, `http://texstudio.sourceforge.net`.
- **Visual Paradigm Online** `https://online.visual-paradigm.com`.

Appendices

A. Elementary terms and definitions

Unmanned System and Unmanned Vehicle

Unmanned System (UMS) is “a powered physical system, with no human operator aboard the principal components, which acts in the physical world to accomplish assigned tasks. It may be mobile or stationary.” (...) “Examples include unmanned ground vehicles (UGV), unmanned aerial vehicles/systems (UAV/UAS), unmanned maritime vehicles (UMV)–unmanned underwater vehicles (UUV) or unmanned water surface borne vehicles (USV)–unattended munitions (UM), and unattended ground sensors (UGS). Missiles, rockets, and their submunitions, and artillery are not considered the principal components of UMSs” [66]. Unmanned Vehicles of any kind (UxVs) belong to a subset of UMS.

Unmanned Aerial Vehicle and Unmanned Aircraft

According to Helnarska et al. the terms used to reference UAVs have evolved significantly since their first appearance during the Second World War, when the UAVs were simply called “pilotless aircrafts”. After the War, between the 40s and 50s, a somewhat colloquial term “drone” – referring to the engine noise – was adopted. Then, in the 1960s the terms “remotely piloted vehicles” and “remotely piloted aircraft” become popular.

Nowadays, two terms have gained the most attention: *Unmanned Aerial Vehicle* (UAV) and *Unmanned Aircraft* (UA). The first term is used by US National Institute of Standards and Technology (US NIST) [66], while the second was officially introduced by International Civil Aviation Organization (ICAO) in 2011 [67]. In the thesis these terms will be used interchangeably.

Remotely Piloted Aircraft (System)

ICAO also introduced the terms *Remotely Piloted Aircraft* (RPA) and *Remotely Piloted Aircraft System* (RPAS) [67]. As European Union Aviation Safety Agency (EASA) has pointed out, RPAS it is a subset of *Unmanned Aircraft System* (UAS), i.e. UAS includes both autonomous and remotely piloted aircrafts [68]. Similarly, an RPA is a subclass of UAV. Therefore, the terms above should not be confused.

Autonomy vs automation

The term autonomy (gr. *autonomos* – having its own laws [69]) in the field of robotics means the robot acts according to the current state of itself and its sensors without any direct interaction from a human or another robot. It also means acting according to its own ruleset [66, 70] or even having “free will” [71]. On the contrary, the term *automation* simply refers to self-regulation during the execution of a given program. In the words of Huang, automation is “human-less operation”, whereas autonomy is “human-like performance” [72].

Some considerations should be taken regarding the autonomy in the terms of UAVs. ICAO defines an *autonomous aircraft* as “an unmanned aircraft that does not allow pilot intervention in the management of the flight” [67]. This implies the autonomous UAV cannot have any kind of fallback remote control. However, the autonomy of a vehicle or a system usually is expressed by the autonomy level from a multi-leveled scale [71, 73, 74]. Choosing a particular scale depends mostly on the context. In the thesis the broader meaning of autonomy will be used to include a mainly autonomous UAV with a remote control fallback mode.

The level of autonomy depends on the complexity of the mission [71, 73] and environment [74]. It can be understood also as a degree of feasibility of the mission without human intervention [74]. Hence, patrolling a known area requires a lower autonomy level than managing a military operation of a group of robots in hostile territory. This topic is discussed in-detail e.g. by Protti and Barzan in [73].
Examples of autonomy metrics:

- 10-level categorical[1] linear scale of automation by Sheridan [75]
- MAP survival space – a biologically-inspired three-dimensional scale of Mobility, Aqcuisition and Protection [76],
- 10-level Autonomous Control Level (ACL) dedicated to UAVs – consisting of four independent dimensions: Observe, Orient, Decide, Act (OODA) [71],
- 8-level scale by Proud et al. – also uses OODA concept [77],
- 6-level categorical scale by the US Navy Office of Naval Research [78],
- Autonomy Levels for Unmanned Systems (ALFUS) by US NIST – three-dimensional scale of Human Independence (HI), Mission Complexity (MC) and Environmental Complexity (EC) [66, 72, 74].
- Scale by the Society of Automotive Engineers[2] (SAE) – 6-degree scale where at level 0 the vehicle is operated entirely by human and at level 5 is operated only by the artificial intelligence [79].

High-Altitude Long Endurance UAVs

A High-Altitude Long Endurance (HALE) UAVs, or alternatively High-Altitude Pseduo Satellites (HAPSs), are "fixed-wing solar electrical platforms prone of perpetually flying in stratosphere" [13].

Dynamic path planning

Dynamic path planning means the ability of the UMS to re-plan either the entire path or its specific part in response to dynamic events, that occur during the mission. Examples include changing weather conditions or the emergence of new obstacles. Dynamic re-planning can be initiated explicitly by a human operator or implicitly by an autonomous decisive module.

Pollution sampling

Pollution sampling is understood as taking measurements in sampled locations of 1D, 2D or 3D space to provide the map of the pollutant distribution. Nonetheless, the algorithms described in the thesis are not restricted to measuring pollutants and can be used to sample other parameters distributed in space as well.

Metaheuristic

A *metaheuristic* a high-level framework used to develop heuristic optimization algorithms. It provides general strategies or guidelines independent of the optimization problem. However,

[1] Consequent levels are just for ordering – they do not have any numerical meaning in Sheridan's scale.

[2] As the name suggests, the SAE scale is dedicated to autonomous and quasi-autonomous ground vehicles. Nevertheless, it is mentioned here for the sake of completeness.

a problem-specific implementation of a heuristic optimization algorithm developed using that framework is also called a metaheuristic [80]. An optimization that uses metaheuristic algorithms is called *metaheuristic optimization* [81].

Algorithm completeness

An algorithm is considered *complete* if it returns a solution if one exists and otherwise returns failure. *Resolution completeness* relaxes this requirement – a resolution complete algorithm guarantees completeness only with a properly set resolution parameter. *Probabilistic completeness* in turn guarantees "that the probability that the planner fails to return a solution, if one exists, decays to zero as the number of samples approaches infinity" [82].

B. Chosen path planning algorithms

The chapter contains a short overview of different control strategies and algorithms which are usable for global optimization, as well as local path planning. The most well-known algorithms are presented first and then followed by more recent state-of-the-art algorithms. The emphasis is put on the stochastic algorithms used in the thesis. The provided list of algorithms is not exhaustive.

B.1 Path planning strategies

According to [20] path planning algorithms employ two opposite strategies: deliberative and reactive. This distinction is similar to the one used in control systems and described e.g., in [83, 84]. Thus, by following this analogy, a hybrid strategy which uses both strategies can also be distinguished.

B.1.1 Deliberative strategy

The beginnings of the deliberative control strategy date back to the 70s [83, 85]. Siegwart et al. in [85] distinguish 4 stages of the control: (1) perception, (2) localization, (3) cognition and (4) motion control. Deliberative approach executes them sequentially in this exact order. Hence, the control loop is *Sense-Plan-Act* (SPA) [84].

The deliberative control uses the global knowledge of the system gathered before the mission starts. Effective planning requires then feasible maps with the proper terrain model and careful static obstacle placement. The deliberative algorithms always are sequential and centralized. Even though they are precise, they are relatively slow to compute due to the amount of processed data. Moreover, the deliberative control systems require complex models of the plant. Besides that, they need the model of the scene to be well-defined and have difficulties handling dynamic and unexpected events [84]. Therefore deliberative algorithms are used mostly for global planning before the mission starts [20].

B.1.2 Reactive strategy

In the mid-80s an opposite approach emerged. Although the reactive strategy has been used already e.g., by Walter in [86], the idea became especially popular after Braitenberg published his book in 1986. His work was dedicated to emotion modeling using simple mobile robots [87].

The reactive strategy makes no use of the global knowledge. Instead it uses data gathered by the sensors located on the plant during the mission [20]. Reactive control systems lack planning and complex models. According to Brooks, a reactive system works by the means of *situatedness* and *embodiment* [88]. A trait of that kind of systems is their lack of memory and connecting the outputs of the sensors straight with the inputs of the actuators [86, 87]. Therefore, the control loop reduces to *Sense-Act* (SA) [84]. Being reactive in the context of path planning means using local knowledge of the obstacle field to plan the trajectory [28].

Having no memory means a reactive system lacks a knowledge representation as well. Inability to store the previous states of the system also effects in inability to learn and to further self-optimize the implemented control algorithm. Moreover, reaction-based systems tends to oscillate due to the frequent state switching and the lack of software-based signal damping between the sensor and the actuator [84]. Nevertheless, they feature short response time and high efficiency in a relatively simple but dynamic and uncertain environment [28]. Usually they are not used for global planning [20].

Many reactive control systems use relatively simple biologically-inspired algorithms. Examples include modeling primitive emotions as in Braitenberg's vehicles [87] or simple chemically-induced reactions, such as chemotaxis[1] used by Alvear et al. [44].

B.2 Path planning algorithms

Some algorithms are designed specifically to address the problem of finding an optimal path. Generally, they can be divided int deterministic and stochastic ones. Deterministic (or exact) algorithms are complete, i.e., they always provide the optimal solution if it exists or fails otherwise. They provide the best possible solution in the global sense according to the chosen criteria [89]. However, while exact, they are also computation-heavy, relatively slow and complex. Time complexity of node-based deterministic algorithms is falls between $O(m\log(n))$ and $O(n^2)$ [2].

Contrary to the exact methods, the stochastic algorithms, i.e., heuristic or metaheuristic methods, provide optimal or near optimal (i.e., "good enough") solution in reasonable time [55]. Some of them may be considered probabilstically complete, i.e., the probability of failure as the number of iterations approaches infinity [82]. The thesis uses three-dimensional representation of the scene. Therefore, for computation efficiency, only stochastic planning algorithms are considered. Exact algorithms are given for reference.

B.2.1 Dijkstra's algorithm

The algorithm was designed in 1956 by a dutch mathematician and computer scientist, Edsger Wybe Dijkstra [90]. Dijkstra's algorithm finds the shortest path between the two nodes of a weighted directed graph with non-negative weights of the edges [90–92]. It is a greedy searching algorithm, which means it chooses the best-rated partial solution in any given moment, however Dijkstra's implementation always returns the shortest path available globally [91, 93]. Even though Yang et al. classify Dijkstra's algorithm as a real-time capable [2], with the complexity of $O(n^2)$ [94], it is inferior compared to other node-based algorithms, e.g., A* or D*.

Dijkstra's algorithm is shown in Algorithm 1. The algorithm requires a non-negative directed graph $G(E, V)$, where E and V are the sets of edges and vertices, respectively. w denotes the weights and s stands for the source node with the initial shortest path length of 0. An initially empty set $\mathcal{P}$ holds vertices, whose final shortest-path weights from the source s have already been determined. The vertices of G are stored in a min-priority queue Q, sorted by their shortest paths d.

In each iteration, the algorithm extracts the shortest path estimate u. It is appended to $\mathcal{P}$, while simultaneously removed from Q. Then, all edges of the vertices, that are children of u are relaxed. Relaxation involves "rewiring" the node v via u if the resulting path is shorter than the current path from s to v. v's predecessor π is changed to u in the process. The steps after the initialization are repeated until Q is emptied.

Algorithm 1 Dijkstra's algorithm, adapted from [91, 94]

```
1: procedure $\mathcal{P} = \textsc{Dijkstra}(G, w, s)$
2:     for all vertex $v \in G.V$ do
3:         Initialize shortest path $v.d \leftarrow \infty$
```

[1] Chemotaxis is the orderly movement towards ("Run") or away from ("Tumble") a chemical stimulus found in primitive microorganisms, such as bacteria [44].

```
 4:         Initialize predecessor $v.\pi \leftarrow \emptyset$
 5:     Initialize shortest path for source $d[s] \leftarrow 0$
 6:     Initialize path $\mathcal{P} \leftarrow \emptyset$
 7:     Initialize priority query $Q \leftarrow G.V$
 8:     **while** $Q \neq \emptyset$ **do**
 9:         $u \leftarrow$ ExtractMin($Q$)
10:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{u\}$
11:         **for all** $v \in$ GetChildren($G, u$) **do**                    ▷ Relax
12:             **if** $v.d > u.d + w(u, v)$ **then**
13:                 $v.d \leftarrow u.d + w(u, v)$
14:                 $v.\pi \leftarrow u$
15:             **end if**
16:     **return** $\mathcal{P}$
```

Dijkstra's algorithm is one of the most popular algorithms [94] and a subject of multiple extensions. For example, the extension by Cowlagi and Tsiotras allows two-node transitions while subject to kinodynamic constraints [32]. Dijkstra's algorithm was successfully used in numerous applications, including UAV path planning [1, 93]. For example, Dhulkefl et al. navigated their UAV in simulation and real-world conditions using Dijkstra's algorithm [1] and Economou et al. used it for energy-efficient flight path planning [92].

B.2.2 A* algorithm

A* is a best-first algorithm. Although faster than Dijkstra's, it does not guarantee finding the globally optimal solution [94]. The complexity of A* is $O(mn), 0 \leqslant m \leqslant n$ [94]. According to a review by Yang et al., A* is valid for online computations [2].

Hart et al. described this algorithm in 1968 in [95] and corrected its formal basis in 1972 [96]. The A* algorithm (Algorithm 2) was designed especially for the Shakey project. The eponymous robot used the algorithm for planning the path of its movement [95, 97].

Initially created for 2D-space, A* works by dividing the area into a grid. Each cell (graph node) is then encoded as free or occupied by an obstacle. The algorithm then finds out an adjacent cell the robot should move into – until it reaches its destination or entire area has been explored and no solution found [89].

While Dijkstra's algorithm searches uniformly the whole state space, A* use directed search by minimizing a so-called F-score function defined as

$$F = G + H$$

where G is the movement cost from the initial position to the current one and H is a heuristic function. By computing H, e.g., euclidean or Manhattan distance from the current position to goal, the algorithm guesses the preferred direction of expansion. Hence, contrary to Dijkstra's algorithm, it does not have to process all the nodes in the graph [89].

A* is presented in Algorithm 2. Starting from the cell containing the initial position, the algorithm moves outwards. It works by specifying the start node s and the goal node g, as well as the heuristic function H used to estimate distance. First, Q and C are initialized as empty open and close sets, respectively. The current node c is set to s.

Next steps are repeated until g is reached. Open set Q is filled with c's neighbors. Then, for each neighbor v the F-score is computed. Now, a neighbor with the lowest F-score becomes the current node c. It is also appended to the close set C. If g is reached, C is returned as the resulting path.

Algorithm 2 A* algorithm, based on [94, 98]

```
1: procedure $C = \text{AStar}(s, g, H)$
2:     Initialize open set $Q \leftarrow \emptyset$
3:     Initialize closed set $C \leftarrow \emptyset$
4:     Initialize current node $c \leftarrow s$
5:     while $c \neq g$ do
6:         Update open set $Q \leftarrow \text{GetNeighbors}(c)$
7:         for all vertex $v \in Q$ do
8:             Compute $F$-score $F(v) = G(v) + H(v)$
9:         Update current node $c \leftarrow \text{GetMinDist}(Q)$
10:        Update closed set $C \leftarrow C \cup c$
11:    return $C$
```

Dhulkefl et al. compared the performance of A* versus Dijkstra's algorithm in their path planning tests. They found that both provide paths of the same lengths. A*, however, offered significant savings in computation time due to the narrowing of the state space [93].

B.2.3 D* algorithm

D* is an informed incremental search algorithm invented by Anthony Stentz in 1994 [43]. Its name is a tribute to A*, while D refers to its dynamic nature. It works similarly to A*, however it changes the heuristic function H dynamically. Stentz published in 1995 an extended version called Focussed D* [99]. The extension was designed specially for real-time path planning and to fully complete D* as a dynamic equivalent of A*.

A simplified interpretation of D* by Choset et al. is shown in Algorithm 3. The state space of D* consists of states representing the positions of the robot in space connected by weighted arcs. The robot starts at a given state and moves to goal via weighted arcs, which incur traversal cost [43]. The algorithm requires a list L of the states available to the robot and uses the following notation:

- X represents a state.
- O is the priority queue.
- L is the list of all states.
- G is the goal state.
- S is the start state.
- $t(X)$ is value of state with regards to the priority queue:
  $t(X) = NEW$, if X has never been in O.
  $t(X) = OPEN$, if X is currently in O.
  $t(X) = CLOSED$, if X was in O but currently is not [100].

Detailed description of D* can be found in [100] or, in its original form, in [43].

Algorithm 3 D* algorithm [101]

```
1: procedure $\mathcal{P} = \text{DStar}(L)$
2:     for all $X \in L$ do
3:         $t(X) \leftarrow NEW$
4:     $h(G) \leftarrow 0$
5:     $\text{Insert}(O, G, h(g))$
```

```
 6:    $X_c \leftarrow S$
 7:    $P \leftarrow$ InitPlan$(O, L, X_c, G)$
 8:    if $P = NULL$ then
 9:        return $NULL$
10:    end if
11:    while $X_c \neq G$ do
12:        PrepareRepair$(O, L, X_c)$
13:        $P \leftarrow$ RepairReplan$(O, L, X_c, G)$
14:        if $P = NULL$ then
15:            return $NULL$
16:        end if
17:        $X_c \leftarrow$ the second element of P {Move to the next state in P}.
18:    return $X_c$
```

D* is considered more complex than A*, but can be used in dynamic events [89, 100, 101]. The algorithm can be utilized for real-time path planning [2].

As stated by its author, D* is a general algorithm that can be applied to any dynamic path optimization problems [43]. The algorithm has multiple extensions, e.g., Field D* [102], 3D Field D* [103] or D* Lite [104].

Field D* extends the basic algorithm by allowing the node to transition to the adjacent cells not only via other nodes, but through edges. It lifts off the heading restrictions[2] and allows for smoother movement [102, 103].

3D Field D* extends it further to 3D-space. The authors highlight the features of the algorithm, e.g., axis-dependent weighting is especially usable for UAVs, for which climbing cost is much higher [103].

D* Lite is an independent algorithm by Koenig and Likhachev. While it mimics the behavior of Focused D*, it is algorithmically different and easier to understand and analyze [104].

B.2.4 Visibility Graphs

Lozano-Pérez and Wesley [105] attribute this method to the research of Nilsson [106] and Ignat'yev et al. [107], independently. While not a path planning algorithm in strict sense, Visibility Graph (VG) provides a convenient graph-based representation of the obstacles in the scene. This representation may be used then to look for the optimal path with graph search algorithms, for example Dijkstra's or A* [6]. As for computation complexity, Yang et al. classify VG-based algorithms as real-time capable with time complexity[3] of $O(n \log n) \leqslant T \leqslant O(n^2)$ [2].

In two-dimensional space VG is a graph, where the nodes are the vertices of the scene, and arcs connect two vertices A and B. A and B must be mutually visible, that is, the segment $\overline{\text{AB}}$ must not intersect an obstacle. The set of initial vertices consists of the start and goal points and their corresponding arcs. Only arcs which are tangent to a pair of polygons are necessary. The basic concept of VG is shown on an example 2D graph in Fig. B.1. By adding bitangents or portions of curved objects, this method can be extended to non-polygonal scenes, e.g., Tangent Visibility Graph (TVG) [65].

VG-based path planner is covered by Shah and Gupta in [108], for example. The researchers used quadtrees and 2D VGs to speed up the search for an optimal path for an unmanned surface vehicle using A* algorithm [108].

VGs have been used in different applications apart from path planning, such as sensor networks, pattern recognition and rendering [65]. As stated by Schøler et al., VGs has been used ex-

[2] For 2D space it means being constrained to $0, \frac{\pi}{2}, \frac{\pi}{4}$, etc. [102].
[3] Depends on the exact algorithm used to build the VG, see [65].

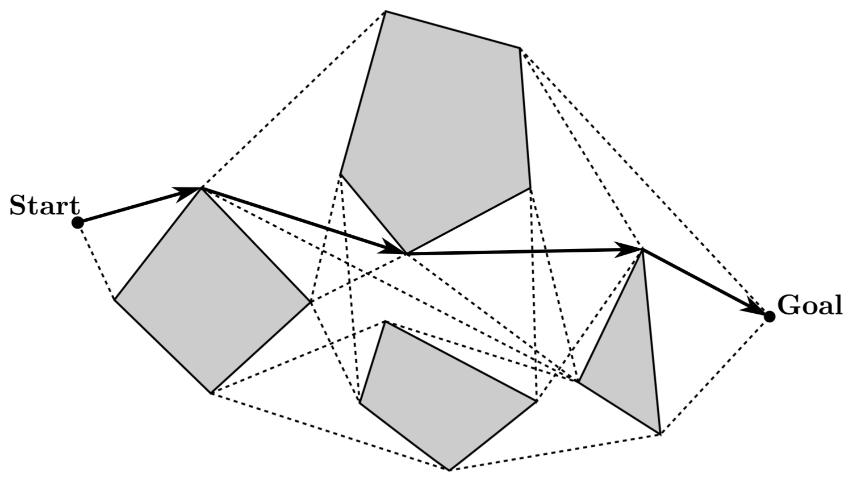

Fig. B.1: An example of a 2D visibility graph [65]

tensively for path planning even though finding an optimal solution this way in three-dimensional space is considered NP-complete [6, 65]. Intuitively, by introducing the third dimension the shortest path around a polyhedral obstacle in general not only traverses the vertices of the polyhedron, but also points on the edges of the polyhedron [6, 105]. A more detailed explanation is given by Bygi and Ghodsi in [65].

Researchers, however, tackled the problem using different approaches. For example, Bygi and Ghodsi proposed an interesting concept to expand VG into "true" 3D by introducing a new structure called *pseudo-graph*. A pseudo-graph is a three-dimensional counterpart with a similar structure to a graph, but with arcs connecting three vertices instead of two. The authors used an algorithm to build a pseudo-graph with complexity of $O(n^3 \log n)$, but also noted it could be optimized to $O(n^2 \log n + k)$, where k is the number of visibility edges of pseudo-graph [65].

Another algorithm, VPP invented by Masehian and Amin-Naseri fuses visibility graphs with Voronoi diagram and potential fields. The authors focuses on its planar implementation, which is verified in simulation. Nonetheless, they also discuss its possible usage in three-dimensional space [109].

As for UAV-specific approaches, Huang and Teo searched for a collision-free 3D path of a UAV by applying 2D VG over a finite set of planes. The algorithm was verified in simulation [63]. The pseudo-code of their three-dimensional expansion of VG is shown in Algorithm 4.

Algorithm 4 3D VG algorithm using multiple planar VGs, based on [63]

```
1: procedure $\mathcal{P}$ = VG
2:     Initialize
3:     Calculate azimuth and elevation angles
4:     Calculate rotation matrix
5:     while Full angle of 3D not reached do
6:         Plane with a rotating angle
7:         Find intersections with obstacles
8:         Find the shortest path using 2D VG
9:         Increase angle
10:    Find the shortest path $\mathcal{P}$
11:    return $\mathcal{P}$
```

Another approach by Schøler et al. used a method for generating a VG inside a rectangular cuboid with convex holes. The method finds a near-optimal flight path for their small-scale helicopter in an environment with multiple obstacles [6].

B.2.5 Rapidly-exploring Random Tree algorithm

The concept of Rapidly-exploring Random Trees (RRTs) was introduced by LaValle in 1998. RRTs were used to solve holonomic, nonholonomic and kinodynamic path planning problems [110]. A tree (Fig. B.2) is a special case of a directed graph, where every node has exactly one parent (except the root) [20]. The nodes represent physical states, while edges are feasible paths connecting them. Travel cost from the node j to i is represented as c_{ij} [20]. The generation of the tree is described, e.g., in [110, 111].

An RRT is a probabilistic-complete sampling algorithm that boasts a set of features discussed, e.g., by LaValle and Kuffner. They include path consistency, simplicity (thus low computation effort) and a bias towards exploring yet unexplored regions [110, 111]. Beard and McLain also point its feasibility to be extended to vehicles with non-linear dynamics and use it for 3D path planning. Moreover, it can be applied to continuous domain planners as well. Nevertheless, even though it is real-time capable with time complexity of $O(n \log n) \leqslant T \leqslant O(n^2)$ [2], the returned solution is rarely close-to-optimal [82, 89]. According to Karaman and Frazzoli, RRT is one of the most widely-used approaches to the motion planning problem [82]. For example Ramana et al. used RRT to plan and then successfully simulate the path of a fixed wing UAV flying in urban environment [112].

RRT algorithm explores the search space uniformly, but randomly. RRT's workflow is shown in Algorithm 5. Assume $\mathbf{w}_{start}$ and $\mathbf{w}_{goal}$ are vectors representing the start and goal waypoints placed on an obstacle map $\mathcal{M}$. RRT first assumes $\mathbf{w}_{start}$ as the root of the tree. Then, it randomly selects a waypoint $\mathbf{w}_{rand}$ somewhere on the map. A waypoint $\mathbf{w}_{nearest}$ is selected from the tree nodes as the nearest to $\mathbf{w}_{rand}$ ($\mathbf{w}_{start}$ for the first iteration). A new waypoint $\mathbf{w}_{new}$ is then placed along $\overline{\mathbf{w}_{rand}\mathbf{w}_{nearest}}$ at a fixed distance D (a parameter of RRT) and $\mathbf{w}_{rand}$ is discarded. Here, the bar operator denotes the line segment from $\mathbf{w}_{rand}$ to $\mathbf{w}_{nearest}$. If $\overline{\mathbf{w}_{nearest}\mathbf{w}_{new}}$ is feasible (collision-free), $\mathbf{w}_{new}$ is added as a new node to the tree and connected to $\mathbf{w}_{nearest}$. Otherwise, it is rejected and a new $\mathbf{w}_{rand}$ is drawn. The steps are repeated until w_{new} can be connected directly to $\mathbf{w}_{goal}$. Then, the final path segment from $\mathbf{w}_{goal}$ to the current $\mathbf{w}_{new}$ is added to the tree. The shortest path from $\mathbf{w}_{start}$ to $\mathbf{w}_{goal}$ is ,is returned as the final waypoint path $\mathcal{W}$. The base algorithm is discussed in-detail in [20, 110, 111].

Algorithm 5 RRT algorithm, rewritten based on [20, 82]

```
1: procedure $\mathcal{W}$ = PlanRRT($\mathcal{M}, \mathbf{w}_{start}, \mathbf{w}_{goal}, D$)
2:     Initialize RRT graph $\mathcal{T} = (V, E)$ as $V = \{\mathbf{w}_{start}\}, E = \{\emptyset\}$
3:     while $\mathbf{w}_{goal} \notin V$ do
4:         $\mathbf{w}_{rand} \leftarrow$ GenerateRandomWaypoint($\mathcal{M}$)
5:         $\mathbf{w}_{nearest} \leftarrow$ FindNearestWaypoint($\mathbf{w}_{rand}, V$)
6:         $\mathbf{w}_{new} \leftarrow$ PlanPath($\mathbf{w}_{nearest}, \mathbf{w}_{rand}, D$)
7:         if PathFeasible($\mathcal{M}, \mathbf{w}_{nearest}, \mathbf{w}_{new}$) then
8:             $V \leftarrow V \cup \{\mathbf{w}_{new}\}$
```

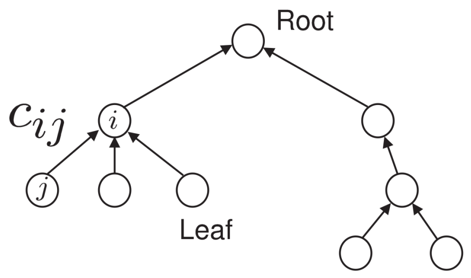

Fig. B.2: A Rapidly-exploring Random Tree [20]

```
 9:            $E \leftarrow E \cup \{(\mathbf{w}_{nearest}, \mathbf{w}_{new})\}$
10:        end if
11:        if PathFeasible($\mathcal{M}, \mathbf{w}_{new}, \mathbf{w}_{goal}$) then
12:            $V \leftarrow V \cup \{\mathbf{w}_{goal}\}$
13:            $E \leftarrow E \cup \{(\mathbf{w}_{nearest}, \mathbf{w}_{goal})\}$
14:        end if
15:    $\mathcal{W} \leftarrow$ FindShortestPath($\mathcal{T}$)
16:    return $\mathcal{W}$
```

Beard and McLain further explain 3D extension of RRT for UAVs, as well as suggest a smoothing method in [20]. The waypoint path $\mathcal{W}$ returned by Algorithm 5 can be further simplified and the total amount of waypoints possibly reduced by applying Algorithm 6. The algorithm first initializes the smoothed path $\mathcal{W}_{smooth}$ with the first waypoint $\mathbf{w}_{start}$ and sets its local start point $\mathbf{w}_{begin}$ to $\mathbf{w}_{start}$. Then, it iterates over the nodes in the input waypoint path $\mathcal{W}$ until the path between $\mathbf{w}_{begin}$ and $\mathbf{w}_{new}$ is obstructed. The algorithm inserts the previous (feasible) $\mathbf{w}_{prev}$ into $\mathcal{W}_{smooth}$. Then, it moves to the next step, i.e., switches $\mathbf{w}_{begin}$ to the next waypoint in $\mathcal{W}_{smooth}$ (identical to $\mathbf{w}_{prev}$ from the previous step). The algorithm keeps running until the final node in $\mathcal{W}$ is reached ($j = N$). The final node $\mathbf{w}_{goal}$ is then appended to $\mathcal{W}_{smooth}$. The waypoint path $\mathcal{W}_{smooth}$ returned by the algorithm consists of the minimal number of nodes required for collision-free transition from $\mathbf{w}_{start}$ to $\mathbf{w}_{goal}$.

Algorithm 6 RRT smoothing algorithm, adapted from [20]

```
 1: procedure $\mathcal{W}_{smooth}$ = SmoothRRTPath($\mathcal{T}, \mathcal{W}$)
 2:     Initialize smoothed path $\mathcal{W}_{smooth} \leftarrow \{\mathbf{w}_{start}\}$
 3:     Initialize pointer to current node in $\mathcal{W}_{smooth} : i \leftarrow 1$
 4:     Initialize pointer to next node in $\mathcal{W} : j \leftarrow 2$
 5:     while $j < N$ do
 6:         $\mathbf{w}_{begin} \leftarrow$ GetNode($\mathcal{W}_{smooth}, i$)
 7:         $\mathbf{w}_{new} \leftarrow$ GetNode($\mathcal{W}, j + 1$)
 8:         if not PathFeasible($\mathcal{T}, \mathbf{w}_{begin}, \mathbf{w}_{new}$) then
 9:             Get last feasible node: $\mathbf{w}_{prev} \leftarrow$ GetNode($\mathcal{W}, j$)
10:             Add deconflicted node to smoothed path: $\mathcal{W}_{smooth} \leftarrow \mathcal{W}_{smooth} \cup \{\mathbf{w}_{prev}\}$
11:             $i \leftarrow i + 1$
12:         end if
13:         $j \leftarrow j + 1$
14:     Add last node from $\mathcal{W} : \mathcal{W}_{smooth} \leftarrow \mathcal{W}_{smooth} \cup \{\mathbf{w}_{goal}\}$
15:     return $\mathcal{W}_{smooth}$
```

B.2.6 RRT* algorithm

Despite the popularity of RRT, Karaman and Frazzoli proved in 2010 that “under mild technical conditions, the cost of the best path in the RRT converges almost surely to a non-optimal value” [82]. Hence, they suggested their own Rapidly-exploring Random Graph (RRG) [113] and its tree-based implementation called RRT* [82]. As for UAV applications, RRT* was used by Meng et al. to develop a path planning system for flying in cluttered environment. The authors compared RRT* with its variant called 3D Informed RRT*, proving the better performance of the second [12].

Contrary to RRT, RRT* guarantees asymptotic optimality, that is, almost sure convergence to an optimum solution [82]. So the algorithm returns a solution that is not just feasible, but also quasi-optimal. As the original RRT algorithm, RRT* has the same time complexity and is real-time capable as well [2]. RRT* is presented in Algorithm 7.

RRT* begins similarly to RRT (see: Algorithm 5). After initializing the tree $\mathcal{T}$ and the waypoints $\mathbf{w}_{rand}$, $\mathbf{w}_{nearest}$ and $\mathbf{w}_{new}$, the algorithm checks whether the path from $\mathbf{w}_{nearest}$ to $\mathbf{w}_{new}$ is feasible, i.e, obstacle-free (Line 7). Then, it initializes a minimal-cost waypoint $\mathbf{w}_{min}$ as $\mathbf{w}_{nearest}$ and retrieves a set of waypoints $\mathbf{W}_{near}$ located not further away than D_{near} (a parameter of RRT*) from $\mathbf{w}_{new}$. If a path between the near waypoint $\mathbf{w}_{near}$ is feasible and costs less than the current cheapest path from $\mathbf{w}_{start}$ to $\mathbf{w}_{min}$, the current $\mathbf{w}_{min}$ is replaced with $\mathbf{w}_{near}$.

Between lines 19 to 25, the algorithm scans $\mathbf{W}_{near}$ for possible optimizations of path costs. It checks if the complete path from $\mathbf{w}_{start}$ to each $\mathbf{w}_{near} \in \mathbf{W}_{near}$ via its parent $\mathbf{w}_{parent}$ is more expensive than the path from $\mathbf{w}_{start}$ to $\mathbf{w}_{near}$ via $\mathbf{w}_{new}$. If true, $\mathbf{w}_{parent}$ of $\mathbf{w}_{near}$ is replaced with $\mathbf{w}_{new}$, thus "rewiring" the final edge to $\mathbf{w}_{near}$. After that, RRT* proceeds similarly to RRT.

Algorithm 7 RRT* algorithm, rewritten based on [20, 82].

```
1: procedure $\mathcal{W}$ = PlanRRTStar($\mathcal{M}, \mathbf{w}_{start}, \mathbf{w}_{goal}, k_{max}, D_{new}, D_{near}$)
2:     Initialize RRT graph $\mathcal{T} = (V, E)$ as $V = \{\mathbf{w}_{start}\}, E = \{\emptyset\}$
3:     for $k = 1$ to $k_{max}$ do
4:         $\mathbf{w}_{rand} \leftarrow$ GenerateRandomWaypoint($\mathcal{M}$)
5:         $\mathbf{w}_{nearest} \leftarrow$ FindNearestWaypoint($\mathbf{w}_{rand}, V$)
6:         $\mathbf{w}_{new} \leftarrow$ PlanPath($\mathbf{w}_{nearest}, \mathbf{w}_{rand}, D_{new}$)
7:         if PathFeasible($\mathcal{M}, \mathbf{w}_{nearest}, \mathbf{w}_{new}$) then
8:             $V \leftarrow V \cup \{\mathbf{w}_{new}\}$
9:             $\mathbf{w}_{min} \leftarrow \mathbf{w}_{nearest}$
10:            $\mathbf{W}_{near} \leftarrow$ FindNearWaypoints($\mathbf{w}_{new}, V, D_{near}$)
11:            for all $\mathbf{w}_{near} \in \mathbf{W}_{near}$ do
12:                if PathFeasible($\mathcal{M}, \mathbf{w}_{near}, \mathbf{w}_{new}$) then
13:                    $c_{new} \leftarrow$ Cost($\mathbf{w}_{start}, \mathbf{w}_{near}$) + $|\overline{\mathbf{w}_{near}\mathbf{w}_{new}}|$
14:                    if Cost($\mathbf{w}_{start}, \mathbf{w}_{new}$) $> c_{new}$ then
15:                        $\mathbf{w}_{min} \leftarrow \mathbf{w}_{near}$
16:                    end if
17:                end if
18:            $E \leftarrow E \cup \{(\mathbf{w}_{min}, \mathbf{w}_{new})\}$
19:            for all $\mathbf{w}_{near} \in \mathbf{W}_{near} \backslash \{\mathbf{w}_{min}\}$ do        ▷ "Rewire" the nodes
20:                $c_{new} \leftarrow$ Cost($\mathbf{w}_{start}, \mathbf{w}_{new}$) + $|\overline{\mathbf{w}_{near}\mathbf{w}_{new}}|$
21:                if PathFeasible($\mathcal{M}, \mathbf{w}_{near}, \mathbf{w}_{new}$) and Cost($\mathbf{w}_{start}, \mathbf{w}_{near}$) $> c_{new}$ then
22:                    $\mathbf{w}_{parent} \leftarrow$ Parent($\mathbf{w}_{near}$)
23:                    $E \leftarrow E \backslash \{(\mathbf{w}_{parent}, \mathbf{w}_{near})\}$
24:                    $E \leftarrow E \cup \{(\mathbf{w}_{new}, \mathbf{w}_{near})\}$
25:                end if
26:        end if
27:        if PathFeasible($\mathcal{M}, \mathbf{w}_{new}, \mathbf{w}_{goal}$) then
28:            $V \leftarrow V \cup \{\mathbf{w}_{goal}\}$
29:            $E \leftarrow E \cup \{(\mathbf{w}_{nearest}, \mathbf{w}_{goal})\}$
30:        end if
31:    $\mathcal{W} \leftarrow$ FindShortestPath($\mathcal{T}$)
32:    return $\mathcal{W}$
```

Contrary to its predecessor, RRT* algorithm continues to optimize the tree even after the goal has been reached. Interestingly, the path returned by RRT* can be further optimized by smoothing using the same Algorithm 6 as for the original RRT.

RRT* was further improved by reducing the search space. A work by Gammell et al. revealed the inefficiencies of RRT* when used as a single-query path planner. The authors introduced pruning the tree of the vertices not contributing to a better solution, as well as reducing the rewiring to a specified neighborhood region. They implemented the idea in their Informed RRT* algorithm [114]. While the original work [114] used hyperspheroids as the rewiring neighborhood, Meng et al. simplified it using an oblique cylinder for their 3D implementation of Informed RRT*, and thus improving the overall performance [12].

B.3 General optimization algorithms

The algorithms mentioned in this section can be considered general in that sense they are suitable for different optimization tasks. Many of the algorithms use relatively simple nature-inspired heuristics instead of exact solutions. While the exact algorithms provide the global optimum, they require high computational effort[4], especially for large solution space [5, 55]. More difficulties arise if dealing with real-world problems characterized by dynamic or noisy objective functions and non-linear constraints. Moreover, exact algorithms are prone to falling into local minima/maxima while optimizing non-convex search landspace [55].

Contrary to the exact methods, these stochastic algorithms, i.e., heuristic or metaheuristic methods, provide optimal or near optimal (i.e., "good enough") solution in reasonable time [55]. Nadimi-Shahraki et al. divide the metaheuristic algorithms based on their origins into algorithms inspired by nature and non-nature-inspired [55] as in Fig. B.3.

The first group includes Tabu Search (TS), Iterated Local Search (ILS) and Adaptive Dimensional Search (ADS), among others. Most of the algorithms, however, are nature-inspired. They can be further divided into three groups. Evolutionary Algorithms base on the Darwin's theory of evolution and include: Genetic Algorithms (GA), Genetic Programming (GP), Evolution Strategy (ES), Differential Evolution (DE) etc. Swarm Intelligence Algorithms are inspired by the collective behavior of animal swarms. Many optimization algorithms fall into this category, for example: Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), Grey Wolf Optimizer (GWO), Krill Herd (KH) and Whale Optimization Algorithm (WOA). Finally, there are algorithms based on the laws of physics, such as Gravitational Search Algorithm (GSA), Charged System Search (CSS) or Henry Gas Solubility Optimization (HGSO) [55]. As for optimization algorithms, the thesis focuses on the chosen nature-inspired evolutionary and swarm intelligence algorithms.

B.3.1 Genetic Algorithm

Genetic Algorithm (GA) is a stochastic optimization method, which mimics Charles Darwin's theory of natural selection [3, 115]. GA was developed by John Holland et al. in 1960s and 1970s [115, 116]. GA falls into the group of Evolutionary Algorithms (EAs), which are a population-based solvers for trial-and-error problems with a meta-heuristic or stochastic optimization approach [5].

A virtual population of chromosomes (i.e., solutions) acts like their biological counterparts evolving over generations and adapting to the environment. For that the equivalents of selection, recombination, crossover and mutation processes are employed. GAs methodology makes use of biologically-inspired terms, e.g., an objective function value, a design candidate and design variables are called fitness, individual and genes, respectively [3]. A generic form of a GA is presented in Algorithm 8.

At the beginning the initial population $\mathcal{P}$ is initialized with M individuals $\mathbf{x}_i(t)$, where $i = 1, 2, ..., M$ and t is the number of the current iteration (or generation). Each $\mathbf{x}_i(t)$ is a vector of length equal to the dimensions of the state space. The optimization starts with the computation

[4] Their execution time increases exponentially proportional to the number of variables [55].

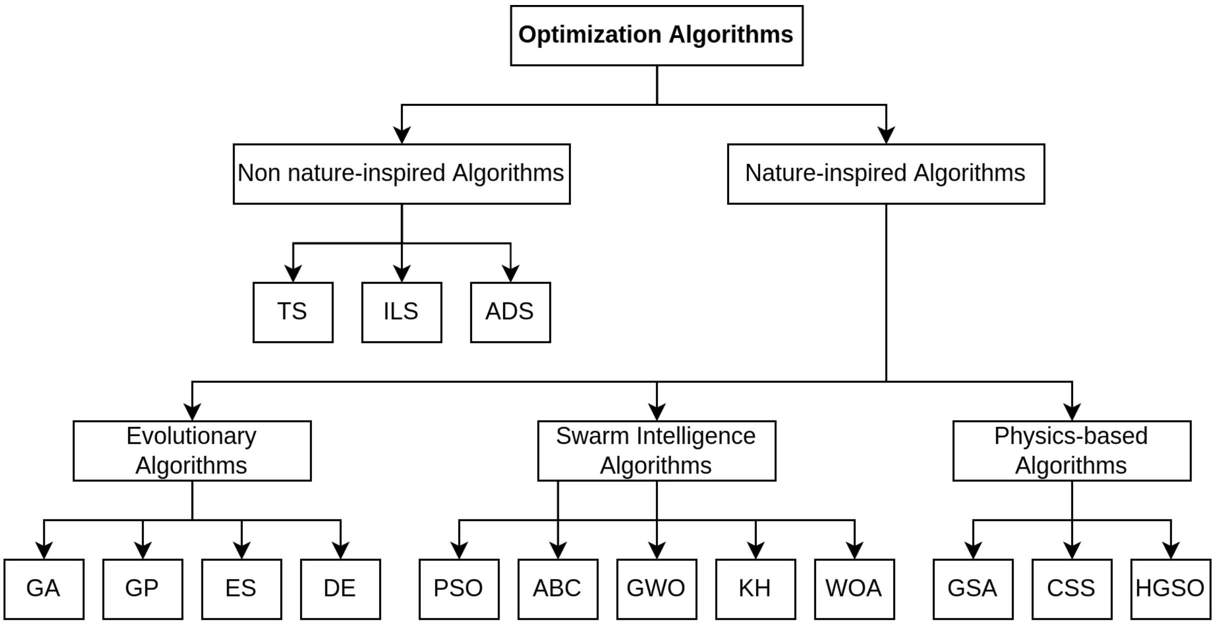

Fig. B.3: An example classification of optimization algorithms, adapted from [55]

of the fitness function $f(\cdot)$ for each individual $\mathbf{x}_i(t)$ in the population $\mathcal{P}$. The globally best individual $\mathbf{p}_g(t)$ is chosen according to $f(\cdot)$. If desired precision δ is achieved, GA returns $\mathbf{p}_g(t)$. Otherwise operators of selection s_o, crossover c_o and mutation m_o are applied. Parameters ϑ_s, ϑ_c and ϑ_m are the probabilities of selection, crossover and mutation, respectively. GA continues to next generations until N-th iteration or until δ is achieved.

Algorithm 8 Genetic Algorithm, adapted from [115, 117]

```
1: procedure $\mathbf{p}_g(t) = \mathrm{GA}(M, N, \delta, \vartheta_s, \vartheta_c, \vartheta_m)$
2:     Initialize population $\mathcal{P}$ with random individuals $\mathbf{x}_i(t), i = 1, 2, ..., M$
3:     Initialize iteration counter $t \leftarrow 1$
4:     Initialize global best solution $\mathbf{p}_g(t) \leftarrow \{\emptyset\}$
5:     repeat
6:         for all $\mathbf{x}_i(t) \in \mathcal{P}$ do
7:             Compute fitness $f(\mathbf{x}_i(t))$
8:         Update $p_g(t) \leftarrow \textsc{Best}(\mathcal{P}, f(\cdot))$
9:         if $\textsc{Precision}(\mathbf{p}_g(t)) \leqslant \delta$ then
10:            break
11:        end if
12:        Perform selection $s_o$ according to $\vartheta_s$
13:        Perform crossover $c_o$ according to $\vartheta_c$
14:        Perform mutation $m_o$ according to $\vartheta_m$
15:        $t \leftarrow t + 1$
16:    until $t \leqslant N$
17:    return $\mathbf{p}_g(t)$
```

As noted by Pehlivanoglu et al., one of the key features of GAs is that instead of searching from a single point in the state space, they move from multiple points. The method is also mathematically simple, not requiring to calculate derivatives or gradients of the objective function. Moreover, GA-based optimization can be implemented as parallel processes, partially mitigating one of their major disadvantages – lack of computation efficiency due to repeated evaluation of the objective function [3]. Despite its advantages, it is restricted to offline-only computations with time complexity of $T \geqslant O(n^2)$ [2].

Yang in [115] places GAs among the most popular evolutionary algorithms in terms of the diversity of their applications. Moreover, GAs are often used as a base for many modern algorithms [115], including UAV-specific. For example, Pehlivanoglu et al. used improved Vibrational Genetic Algorithm (VGA) to guide their UAV over artificially generated terrain in simulation [3].

B.3.2 Particle Swarm Optimization algorithm

In 1995 James Kennedy and Russell Eberhart described an algorithm, which simulated the movement of a bird flock. They called it Particle Swarm Optimization (PSO) [118]. PSO is a meta-heuristic optimization method where the particles, or solutions, are updated every iteration based on the best global and local solutions [40]. As GA, PSO is also classified as an EA.

PSO works by computing the movement of the swarm of M individuals. The movement of the swarm is represented by the positions $x_{i,d}$ and velocities $v_{i,d}$ of the i-th individual in the d-th dimension, $i = 1, 2, ..., M$. To simplify the notation, the values $x_{i,d}$ and $v_{i,d}$ for different dimensions can be stored in vectorized form as $\mathbf{x}_i$ and $\mathbf{v}_i$, respectively. Then, the equations used to compute the movement can be expressed formally as

$$\begin{aligned} \mathbf{v}_i(t+1) &= \mathbf{v}_i + c_1 * \text{rand}_1 * \big(\mathbf{p}_i(t) - \mathbf{x}_i(t)\big) + c_2 * \text{rand}_2 * \big(\mathbf{p}_g(t) - \mathbf{v}_i(t)\big) \\ \mathbf{x}_i(t+1) &= \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \end{aligned} \tag{B.1}$$

where $\mathbf{p}_g(t)$ is the globally best solution, $\mathbf{p}_i(t)$ is the locally best solution of the i-th individual, c_1 and c_2 represent learning factors, rand_1 and rand_2 are random uniform distributions in range $\langle 0, 1\rangle$, and t is the current iteration. Distribution of $\mathbf{x}_i$ should be uniform to evenly cover the state space [115]. While $\mathbf{v}_i$ can have theoretically any value, it is usually bound to a range $\langle 0, v_{max}\rangle$ [115]. PSO's pseudo code is shown in Algorithm 9.

PSO begins with the initialization of the swarm $\mathcal{S}$ of M individuals with the randomized values of $x_{i,d} \in \mathbf{x}_i$ and $v_{i,d} \in \mathbf{v}_i$. The number of the current iteration is stored as t. Then, the fitness function $f(\cdot)$ is computed for each individual. Next, global $\mathbf{p}_g$ and local $\mathbf{p}_i$ best solutions are chosen. If the precision requirement is met, PSO returns the optimized result. Otherwise t is incremented and the optimization continues up to N iterations.

Algorithm 9 PSO algorithm, adapted from [115, 117]

```
1:  procedure $\mathbf{p}_g(t), \mathbf{p}_i(t) = \text{PSO}(M, N, \delta, c_1, c_2)$
2:      Initialize swarm $\mathcal{S}$ of $M$ individuals with random $x_{i,d} \in \mathbf{x}_i$ and $v_{i,d} \in \mathbf{v}_i$
3:      Initialize iteration counter $t \leftarrow 1$
4:      Initialize global best solution $\mathbf{p}_g(t) \leftarrow \{\emptyset\}$
5:      Initialize $M$ local best solutions $\mathbf{p}_i(t) \leftarrow \{\emptyset\}$
6:      repeat
7:          for all $\mathbf{x}_i(t) \in \mathcal{S}$ do
8:              Compute fitness $f(\mathbf{x}_i(t))$
9:          Update $p_g(t) \leftarrow \text{BestGlobal}(\mathcal{S}, f(\cdot))$
10:         Update $p_i(t) \leftarrow \text{BestLocal}(\mathbf{x}_i, f(\cdot))$
11:         if $\text{Precision}(\mathbf{p}_g(t)) \leqslant \delta$ then
12:             break
13:         end if
14:         Update $\mathbf{x}_i$ and $\mathbf{v}_i$ according to (B.1)
15:         $t \leftarrow t + 1$
```

```
16:     until $t \leqslant N$
17:     return $\mathbf{p}_g(t)$, $\mathbf{p}_i(t)$
```

The algorithm could be improved by introducing the inertia factor to $\mathbf{v}_i$ to improve convergence, so the last equation (B.1) becomes

$$\mathbf{v}_i(t+1) = \theta(t) * \mathbf{v}_i + c_1 * \text{rand}_1 * (\mathbf{p}_i(t) - \mathbf{x}_i(t)) + c_2 * \text{rand}_2 * (\mathbf{p}_g(t) - \mathbf{x}_i(t))$$

where $\theta(t)$ is the inertia function. In the simplest case $\theta(t)$ is constant, e.g., in range $\langle 0.5, 0.9 \rangle$ [115].

As highlighted by the authors themselves, a major advantage of PSO is simplicity. Its simple concept employs only primitive mathematical operators and can be implemented in a few lines of code. Thus, it is memory-efficient. Even tough Kennedy and Eberhart consider it fast [118], Yang et al. classify it as real-time incapable with time complexity of $T \geqslant O(n^2)$ [2]. Despite the simplicity of basic PSO, even simpler implementations of the algorithm exist. For example, Yang in 2008 developed Accelerated PSO (APSO). APSO discards the local best solutions $\mathbf{p}_i$ and returns global $\mathbf{p}_g$ only [115].

PSO has been used for almost every area in optimization, computational intelligence, and design applications [115]. A UAV-related example is Andrade's work considering offline path planning of an autonomous UAV [40].

B.3.3 Ant Colony Optimization algorithm

Ant Colony Optimization (ACO) bases on the results of collective research on real ant systems by Colorni et al. from 1991 [119]. The algorithm mimics the natural optimization behavior used by ant foragers to solve the traveling salesman problem (i.e., finding the shortest path between their nest and a food source) [8, 120]. Even though a single ant has a minimal cognitive capabilities, the colony is highly structured, behaves logically and can solve different optimization problems [119, 121].

The ant foragers find the path to food by exploiting pheromone information and without visual cues. A forager deposits pheromone track as it walks, which tends to attracts other foragers. The pheromone evaporates over time, however. The shorter the path, the more often an ant traverses it, thereby strengthening the pheromone track and attracting even more ants [8, 119]. Algorithm 10 presents a variant of ACO called Ant Colony System (ACS) by Dorigo and Gambardella [121]. The key principle is the same, however.

Algorithm 10 ACS algorithm [120] (a simplification of the original ACS available in [121])

```
1: procedure ACS
2:     Initialize
3:     while stopping condition not satisfied do
4:         Position each ant in a starting node
5:         repeat
6:             for all ant do
7:                 Choose next node by applying the state transition rule
8:                 Apply step by step pheromone update
9:         until every ant has built a solution
10:        Update best solution
11:        Apply offline pheromone update
12:    return shortest path
```

ACO is an example of distributed optimization. The agents (ants), however, are not aware of their cooperative behavior. They unintentionally work together through low-level interactions instead [119]. From technical point of view, ACO bases on an autocatalytic process supported by "greedy force". The two processes working together tend to converge to the globally optimal solution, unlike each of them running independently [119].

The original ACO was initially developed to be a metaheuristic for combinatorial optimization. Nevertheless, it can be successfully adapted to solve continuous problems without significant modification to its initial concept. For example, Socha and Dorigo developed $ACO_{\mathbb{R}}$ algorithm for continuous domains by utilizing a probability density function instead of a discrete probability distribution found in ACO [47].

The advantages of ACO include simplicity [119], versatility [8, 122], robustness [8, 122] and population-based approach [122]. Nevertheless, Ling and Hao state searching too long and easily trapping into local optimum as its disadvantages [9]. Yang et al. classify the basic ACO algorithm as offline-only with time complexity of $T \geqslant O(n^2)$ [2]. While originally used to solve the traveling salesman probem, the algorithm can also tackle different optimization problems. Examples include the job-shop scheduling problem, the vehicle routing problem and the quadratic assignment problem [8].

As for UAV-related use case, Duan et al. used a hybrid meta-heuristic ACO-DE algorithm to solve the three-dimension path planning problem in combat field environment [8]. They optimized the pheromone trail of the improved ACO model by applying Differential Evolution (DE) algorithm during the process of ant pheromone updating. The researchers used a 3D mesh-based model of the scene and assigned a threat value to each node of the mesh. Then, they used ACO-DE to minimize the path cost. ACO-DE accelerated the global convergence speed while preserving the strong robustness of the basic ACO in simulation [8].

Ling and Hao report improvement of the basic ACO in their UAV-related research. The researchers added threat cost factor of the current node into state transition probability. Thus, the improved algorithm drives the ant colonies towards the nodes having the smaller integrated cost. They proved the feasibility of their improved ACO in offline and online simulations [9].

B.3.4 Improved Grey Wolf Optimizer

The Grey Wolf Optimizer (GWO) was originally proposed by Mirjalili et al. in 2014 [123]. The authors proven it is highly competitive compared to other heuristics such as particle swarm optimization (PSO), gravitational search algorithm (GSA) and differential evolution (DE). Moreover, they successfully used GWO for sample mechanical engineering problems as well as to design an optical buffer [123]. Since then, it has been successfully applied for solving different optimization problems – from engineering to medical [55, 124].

GWO is a nature-inspired single-objective[5] metaheuristic swarm intelligence algorithm, which mimics the leadership and hunting behavior of a pack of grey wolves (*Canis lupus*) [123, 124]. Exactly as in nature, the pack is divided into four classes of wolves (i.e., solutions).

The alpha wolf (α) is the dominant specimen characterized by the best fitness value. Being the leader, it has the most influence on the behavior of the pack. Beta (β) is a second-tier wolf, which assists the alpha in decision-making. Next in the hierarchy are subordinate wolves (δ). Subordinates have to submit to alphas and betas, although they still dominate omegas (ω), the lowest tier of the wolves [123].

The algorithm works by mimicking the three major phases of the wolf hunt, i.e., the optimization of their positions versus the position of the prey. The phases are (1) encircling, (2) hunting and (3) attacking the prey. In each phase the position[6] of a wolf (a vector of optimized variables) is mathematically modeled according to its tier in the pack and the positions of the higher-tier

[5] Although the authors argue it can be extended to be multi-objective [123].

[6] For multiple optimization variables the position is defined in hyperspace.

wolves. In GWO the optimization is guided by α, β and δ (in that order), while the ω wolves follow these three [123].

GWO has evolved into many variants to overcome deficiencies found in the original algorithm, i.e., avoiding local minima and accelerate convergence. One of the newest variants called Improved GWO (I-GWO) was proposed in 2021 by Nadimi-Shahraki et al. in [55]. The authors introduced the dimension learning-based hunting (DLH) search strategy, which mimics individual wolf hunting behavior. The pseudo code of I-GWO is shown in Algorithm 11.

Algorithm 11 The I-GWO algorithm, adapted from [55]

```
1:  procedure $\mathbf{x}$ = I-GWO($N, D, k_{max}$)
2:      Randomly distribute $N$ wolves in the search space and calculate their fitness
3:      for $k = 2$ to $k_{max}$ do
4:          Find $\mathbf{x}_\alpha, \mathbf{x}_\beta, \mathbf{x}_\delta$
5:          for $i = 1$ to $N$ do
6:              Compute 1st new position candidate $\mathbf{x}_{i-GWO}(t+1)$
7:              Compute neighborhood radius $R_i(t)$
8:              Construct neighborhood $\mathbf{x}_i(t)$ with radius $R_i$
9:              for $d = 1$ to $D$ do                ▷ Construct 2nd new position candidate
10:                 $x_{i-DLH,d}(t+1) = x_{i,d}(t) + rand \times (x_{n,d}(t) - x_{r,d}(t))$
11:             $\mathbf{x}_i(t+1) \leftarrow$ SelectBest($\mathbf{x}_{i-GWO}(t+1), \mathbf{x}_{i-DLH}(t+1)$)
12:             if SelectBest($\mathbf{x}_i(t+1), \mathbf{x}_i(t)$) = $\mathbf{x}_i(t+1)$ then
13:                 Update the population with $\mathbf{x}_i(t+1)$
14:             end if
15:     return global optimum $\mathbf{x}$
```

Assume D-dimensional optimization problem. First, N wolves are distributed randomly in the search space in a given range and their fitness values are calculated. Then, the first new position candidate $\mathbf{x}_{i-GWO}$ is computed, exactly as in GWO. Contrary to the original GWO, in DLH each wolf learns from its neighbors as well. Hence, the second new position candidate is $\mathbf{x}_{i-DLH}$ calculated. Next, the best of the two is chosen as $\mathbf{x}_i(t+1)$. If it is better than $\mathbf{x}_i(t)$, the population is updated. The whole process is repeated up to k_{max} iterations. The algorithm returns the globally optimal position $\mathbf{x}$.

GWO was successfully used for UAVs, e.g., in comparative studies by Kiani et al. The authors compared different flavors of GWO-based algorithms[7] while planning an obstacle-free 3D flight path in simulation.

B.4 Other algorithms

Many more algorithms exist, that are not described or even mentioned here. For example, a paper by Garcia et al. presents an extension of a graph-based Lazy Theta* algorithm. It is designed especially for dynamic global path planning in hazardous weather [7]. As the HALE UAV considered in the thesis is meant to fly only in favorable weather conditions, this idea is not discussed here.

Vanneste et al. invented 3DVFH+ algorithm, which uses a polar-based Octomap[8]. The algorithm achieves real-time performance in three-dimensional obstacle avoidance [126]. Nevertheless, it is more suited towards confined and obstacle-dense spaces (i.e., indoors).

Hebecker et al. describes a concept of a reactive local path planner, which uses a wavefront algorithm to avoid static obstacles. Interestingly, the algorithm uses LIDAR sensor data instead

[7] Specifically, I-GWO (i.e., Incremental GWO, not Improved GWO) and Ex-GWO (Extended GWO) [125]

[8] Original Octomap is a probability-driven volumetric 3D environment model by Hornung et al. [36].

of an abstract map. The researchers proved the algorithm able to plan collision-free paths in a static 3D space [127].

Many different optimization algorithms also feature UAVs. Alihodzic et al. verified Elephant Herding Optimization (EHO) algorithm in simulation as a path planner for a UAV flying over a battlefield. The researchers used EHO to minimize threat and fuel cost functions and concluded that the performance of EHO is promising [128].

A paper by Yue and Zhang employs Simulated Annealing (SA) and K-means algorithms. The authors used the algorithms to plan a flight path of their UAV to search for signs of life in a disaster site after an earthquake. They decomposed the task in two stages, similarly to the approach used in the thesis [129]. A two-step approach was used also by Bortoff, who used Voronoi-based path planner. The planner first constructs a suboptimal rough-cut path is generated through the radar sites basing on Voronoi polygons. Then, nonlinear ordinary differential equations are simulated, using the graph solution as an initial condition [130].

B.5 Summary

This chapter covered a short state-of-the-art review of algorithms used for UAV-based path planning. The algorithms were divided into specialized path-planning and obstacle-avoiding algorithms, as well as more general ones suitable for diverse optimization tasks. However, the list of algorithms mentioned in this chapter is not meant to be exhaustive. The algorithms were chosen based on their diversity, as well as their availability in MATLAB.