Chapter 2
Motion Planning

Abstract Motion planning, still an active research topic is presented in this chapter. It is a fundamental task for an aerial robot that must plan complex strategies and establish long-term plans. Many topics are considered in this chapter. Controllability concerns the possibility for an aerial robot to fly from an initial position to another one. It provides an answer to the question whether the state can be driven to a specific point from any (nearby) initial condition and under an appropriate choice of control. Controllability of an aerial robot represented by its translational kinematic model is studied. The problem of trajectory planning, important aspect of aerial robot guidance, follows: trim trajectories and maneuvers are introduced as well as Dubins and Zermelo problems. The trajectories must be flyable and safe, Thus, nonholonomic motion planning is studied using the notions of differential flatness and nilpotence. As aerial robots are likely to operate in a dynamic environment, Collision avoidance is a fundamental part of motion planning. The operational environment is considered to be three dimensional, it may contain zones that the robot is not allowed to enter and these zones may not be fully characterized at the start of a mission. 3D aerial path planning has the objective to complete the given mission efficiently while maximizing the safety of the aircraft. To solve this problem, deterministic and probabilistic approaches are presented. To cope with imperfect information and dynamic environments, efficient replanning algorithms must be developed that correct previous solutions for a fraction of the computation required to generate such solutions from scratch, Thus, replanning methods such as incremental and anytime algorithms are studied.

2.1 Introduction

Motion planning, still an active research topic, is a fundamental task for an aerial robot that must plan complex strategies and establish long-term plans. One important component of motion planning is obstacle and collision avoidance [106].

Y. Bestaoui Sebbane, *Planning and Decision Making for Aerial Robots*,
Intelligent Systems, Control and Automation: Science and Engineering 71,
DOI: 10.1007/978-3-319-03707-3_2, © Springer International Publishing Switzerland 2014

The problem is to find efficient algorithms that give an open loop control flying the aerial robot from a start state to a goal state. In order to implement an effective planning strategy, a deep analysis of various contributing elements is needed. Mission tasks, required payload and surveillance systems drive the platform choice, but platform characteristics strongly influence the path [44]. The type of mission defines the environment for planning actions, the path constraints and the required optimization process [27]. Motion planning problems for nonholonomic and underactuated aerial systems can be characterized along a number of dimensions: Is the system drift-free or not? Are there configuration obstacles? What form do control constraints take? Is it more important to find a solution that minimizes time or energy, or is it more important to find a satisficing motion plan quickly? Approaches can be different depending on the used model. Aerial robots are usually more sensitive to weather conditions than manned aircraft due to their size, so the issue of avoiding weather hazards even has more impact and it has to be taken into consideration. Several alternatives can be used to calculate trajectories that avoid regions with dangerous weather and with the least deviation from the shortest trajectory.

Section 2.2 presents controllability analysis of the translational kinematics of an aerial robot. Section 2.3 presents trajectory planning including trim trajectories and maneuvers. Section 2.4 introduces nonholonomic motion planning when no obstacles are in the flight path. It consists in generating a collision free trajectory from the initial to the final desired positions, respecting the nonholonomic constraints. Solving the problem of planning in a cluttered environment with obstacles or restricted or prohibited areas is then considered, it can be obtained by several methods. Section 2.5 analyses how the aircraft can reason with respect to its representation of space, first formulating the problem of collision/obstacle avoidance then presenting some discrete and continuous algorithms for the resolution of this problem [97]. Common approaches to nonholonomic motion planning can be divided roughly into methods that perform complete search (often based on discretization) and iterative refinement methods [88, 123, 125]. In this section, the algorithms of obstacle avoidance for aerial robots. In Section 2.6, some approaches are presented because the need for replanning may also substantially revise the path planning strategy for the selected type of missions [28, 69, 153].

2.2 Controllability

A vector field arises in a situation where there is a direction and magnitude assigned to each point of space. The classic example of a vector field in the real world is the velocity of a steady wind. A vector field is mathematically the choice of a vector for each point of a region in space. In general, let U denote an open set of Euclidean space $\mathbb{R}^n$, then a vector field on U is given by a function $f : U \to \mathbb{R}^n$. Vector fields and differential equations give rise to families of transformations of space called flows. Various notions of accessibility can be defined. As modeling involves approximation and some degree of uncertainty in dealing with dynamic systems, properties whose

validity in nominal situations may imply validity in almost all situations are called generic properties. These are properties that hold on open and dense subsets of suitable domains of definition, provided they hold at some points of such domains. A basic notion is that of reachable state and controllability. Controllability concerns the possibility of steering the system from a state X_0 to another state X_1. For linear systems, controllability is a structural property, in the sense that any linear system can be split into a controllable subsystem and an autonomous uncontrollable one. For non linear systems such as aerial robots, controllability is more difficult to characterize than for linear systems. The linearization principle for controllability is useful in reducing questions of local controllability to the linear case. Controllability of a dynamic system provides a definite answer to the question whether the state can be driven to a specific point from any (nearby) initial condition and under an appropriate choice of control. There are two basic approaches to check controllability:

1. Linear controllability by first linearizing the nonlinear system at an equilibrium point (typically assumed to be the origin) or along a given trajectory and then invoking the Kalman rank condition or calculating the controllability grammian for the resulting linear system, either time-invariant or time-varying.
2. Apply Chow's theorem essentially a rank condition on the Lie brackets of vector fields of the system.

For nonlinear systems, local controllability along a trajectory corresponds to the possibility of controlling this system around this trajectory. This section focuses on evaluating controllability: given a description of the system, Lie algebraic tests can be used to study the reachable state space. Controllability refers to having the ability to move on demand between two arbitrary configurations. Non holonomic systems are not locally controllable, yet in many cases they are globally controllable. The nonlinear translational kinematics model can be written as a set of first-order ordinary differential equation as:

$$\begin{aligned} \dot{X} &= F(X, U) \\ Y &= h(X) \end{aligned} \tag{2.1}$$

where X is the state vector $X \in \mathbb{R}^n$, U is the control vector $U \in \mathbb{R}^m$, Y is the measured output $Y \in \mathbb{R}^p$. A system of the form (2.1) is said to be controllable at X_0 if there exists a neighborhood V of X_0 such that any state $X_1 \in V$ is reachable from X_0. Controllability is an important notion for affine systems with or without drift. Sussmann [146] and Jurdjevic [68] introduced the theory of Lie groups and their associated Lie algebras into the context of nonlinear control to express notions such as controllability, observability and realization theory. Some of the early works on nonlinear controllability of driftless systems were based on linearization of nonlinear systems. It was observed that if the linearization of a nonlinear system at an equilibrium point is controllable, the system itself is locally controllable. Later, a differential geometric approach to the problem was adopted in which a control system was viewed as a family of vector fields. It was observed that a lot of the interesting control theoretic information was contained in the Lie brackets of these vector fields.

Chow's theorem [68] leads to the characterization of controllability for systems without drift. It provides a Lie algebra rank test, for controllability of nonlinear systems without drift, similar in spirit to that of Kalman's rank condition for linear systems [25, 30, 131, 155].

Let's begin with a brief study of controllability of nonlinear systems applied to this kinematic model, when wind velocity is assumed to be constant:

$$\begin{aligned} \dot{x} &= V\cos\chi\cos\gamma + W_x \\ \dot{y} &= V\sin\chi\cos\gamma + W_y \\ \dot{z} &= V\sin\gamma + W_z \\ V &= u_1 \\ \dot{\chi} &= u_2 \\ \dot{\gamma} &= u_3 \end{aligned} \tag{2.2}$$

or in an affine nonlinear control system with drift:

$$\dot{X} = g_1u_1 + g_2u_2 + g_3u_3 + f(X) = G(X)U + f(X) \tag{2.3}$$

where the state variable is defined as $X = (x, y, z, \chi, \gamma)^T$, the control variable by $U = (V, \dot{\chi}, \dot{\gamma})^T$, and the input functions are given by:

$$\begin{aligned} g_1 &= (\cos\gamma\cos\chi, \cos\gamma\sin\chi, \sin\gamma, 0, 0)^T \\ g_2 &= (0, 0, 0, 1, 0)^T \\ g_3 &= (0, 0, 0, 0, 1)^T \\ f(X) &= \left(W_x, W_y, W_z\right) \end{aligned} \tag{2.4}$$

or equivalently

$$G(X) = [g_1 \;\; g_2 \;\; g_3] \tag{2.5}$$

Several important results have been derived based on the structure of the Lie algebra generated by the control vector fields. Assume $X \in M \subset \mathbb{R}^6$ where M is a smooth manifold. Let $X(t, X_0, u)$ denote the solution for $t \geq 0$ for a particular input function u and initial condition $X(0) = X_0$. Let $R^{\nu}(X_0, T) = \left\{X \in M, \textbf{there exists an admissible input } u : [0, T] \to u\right\}$ such that

$$X(t, X_0, u) \in V, 0 \leq t \leq T \textbf{ and } X(T) = X_f \tag{2.6}$$

By definition, all the Lie brackets that can be generated using these vector fields belong to A. The accessibility algebra A of the system (2.2) is the smallest sub algebra of $V \subset \mathbb{R}^n$ that contains f, g_1, g_2, g_3. The accessibility distribution Δ_A of the system (2.2) is defined as:

$$\Delta_A = span\,\{\nu | \nu \in A\} \tag{2.7}$$

thus Δ_A is the involutive closure of $\Delta = span\{f, g_1, g_2, g_3\}$. The computation of Δ_A may be organized as an iterative procedure:

$$\Delta_A = \mathbf{span}\{\nu | \nu \in \Delta_i; \forall i \geq 1\} \tag{2.8}$$

With:

$$\begin{gathered} \Delta_1 = \Delta = \mathbf{span}\{f, g_1, g_2, g_3\} \\ \Delta_i = \Delta_{i-1} + span\{[g, \nu], g \in \Delta_1, \nu \in \Delta_{i-1}\}, i \geq 2 \end{gathered} \tag{2.9}$$

This procedure stops after K steps, where K is the smallest integer such that $\Delta_{K+1} = \Delta_K > \Delta_A$. This number is called the nonholonomy degree of the system and is related to the 'level' of Lie brackets that must be included in Δ_A. Since $dim(\Delta_A) \leq n$, so the following condition is necessary: $K \leq n - m$. To find Δ_A of this system (2.2):

$$\Delta_1 = span\{f, g_1, g_2, g_3\} \tag{2.10}$$

At the second level, the following relationship can be written:

$$\Delta_2 = \Delta_1 + span\left\{ \begin{array}{c} g_4 = [f, g_1], g_5 = [f, g_2], g_6 = [f, g_3], \\ g_7 = [g_1, g_2], g_8 = [g_1, g_3], g_9 = [g_2, g_3] \end{array} \right\} \tag{2.11}$$

with

$$\begin{gathered} [f, [f, g_1]] = [f, [f, g_2]] = [f, [f, g_3]] = [g_3, [f, g_3]] = 0_{6\times 1} \\ [g_1, [f, g_1]] = V[f, g_3] \\ [g_1, [f, g_3]] = -[g_3, [f, g_1]] = -V[f, g_1] \\ [g_1, [f, g_2]] = -[g_2, [f, g_1]] = -\tan\gamma[f, g_2] \\ [g_2, [f, g_3]] = -[g_3, [f, g_2]] = -V[f, g_2] \\ [g_2, [f, g_2]] = (V\sin\chi\cos\gamma \; V\cos\chi\cos\gamma \; 0\; 0\; 0\; 0)^T \end{gathered}$$

Identical calculations are made for levels 3 and 4 to obtain

$$\Delta_4 = \Delta_3 = \Delta_A = span\{f, g_1, g_2, g_3, [f, g_1], [f, g_2], [f, g_3]\} \tag{2.12}$$

The Lie brackets are zero since level 2, thus this system is nilpotent. The rank of the matrix obtained: $M = (f, g_1, g_2, g_3, [f, g_1], [f, g_2])$. Straightforward calculations allow to write:

$$rank(M) = rank\begin{pmatrix} V\cos\chi\cos\gamma + W_x & 0\;0\;0 & V\cos\chi\sin\gamma & V\sin\chi\cos\gamma \\ V\sin\chi\cos\gamma + W_y & 0\;0\;0 & V\sin\chi\sin\gamma & -V\cos\chi\cos\gamma \\ V\sin\gamma + W_z & 0\;0\;0 & -V\cos\gamma & 0 \\ 0 & 1\;0\;0 & 0 & 0 \\ 0 & 0\;1\;0 & 0 & 0 \\ 0 & 0\;0\;1 & 0 & 0 \end{pmatrix} \tag{2.13}$$

where the determinant of this matrix is given by:

$$det(M) = -V^3 \cos\gamma - W_z V^2 \sin\gamma \cos\gamma - W_x V^2 \sin\gamma \cos^2\gamma - W_y V^2 \cos\gamma \cos^2\gamma \tag{2.14}$$

So the previous relation (2.14) must be studied, verifying that the determinant is different from zero. With this condition, system (2.2) verifies the Lie Algebra rank condition and is locally accessible. Therefore the non-holonomy degree of the system is: $(K = 3)$ for $det(M) \neq 0$ and its satisfies the condition $(K = 3) \leq (n - m = 3)$.

When wind is neglected, aerial robots such as helicopters (property of symmetry) when represented by drift-free nonholonomic systems (2.2) are small time nonlinearly controllable and hence there is a solution to their navigation problem of planning a feasible motion and determining the corresponding steering inputs when no bounds are assumed on the controls [60]. For airplane like velocity, because of the stall velocity $V_{stall} \leq V \leq V_{max}$, the small time local controllability is no longer retained. However, reachability property is still valid as long as $det(M) \neq 0$. However, it is well known that when the velocity of the wind is too large versus the velocity of the aerial robot, it may become uncontrollable. To the author's knowledge, a formal proof of controllability in the general case is yet to be found.

The notion of the observability of a system concerns the possibility of recovering the state $X(t)$ from knowledge of the measured output $Y(t)$, the input $U(t)$ and possibly a finite number of their time derivatives $Y^{(k)}(t), k \geq 0$ and $U^{(l)}, l \geq 0$. The structural property which can be easily characterized in a nonlinear framework concerns the existence of an open and dense submanifold of the state space R^n around whose points the system is locally observable. The use of an observer that evaluates the state from the knowledge of inputs and outputs is in order whenever the state itself is not directly measurable, but its value is required for computing a feedback or for monitoring the system behavior [109]. In contrast to the linear situation, observability of a given nonlinear system is necessary but not sufficient to assure the possibility of constructing an observer.

2.3 Trajectory Planning

Planning trajectories is a fundamental aspect of aerial robot guidance. An aerial robot charts its own course using navigation by guidance and tracking control and by multi-functional autopilots with set points changed automatically when required [73, 100, 130]. Guidance is the logic that issues steering commands to the aerial robot to accomplish certain flight objectives. The guidance algorithm generates the autopilot commands that steer the aerial robot. Most applications of aircraft guidance have involved applications where operational conditions and the environment are either known a priori and are relatively stationary or can be accurately predicted [79, 84, 85]. In these cases, open loop trajectories are often used. Changes in problem parameters are frequently treated as perturbations on the nominal conditions and handled through techniques like neighboring optimal control [7, 159]. In contrast, autonomous guidance of aerial robots involves determining a trajectory based on partial and evolving knowledge of geography and mission. In such a setting, changes in the problem parameters will often exceed the level of perturbations and require a

more radical update of the trajectory [72]. In guidance studies, only local information on the wind flowfield is assumed to be available and a quasi-optimal trajectory is determined, namely a trajectory that approximates the behavior of the optimal trajectory [2, 3, 104].

To lead the aerial robot from an initial configuration $q(t_i) = q_i$ to a final configuration $q(t_f) = q_f$ in the absence of obstacles, a trajectory $q(t)$ for $t \in [t_i, t_f]$ has to be planned. The trajectory $q(t)$ can be broken down into a geometric path $q(s)$ with $\frac{dq(s)}{ds} \neq 0$ for any value of s and a timing law $s = s(t)$ with the parameter s varying between $s(t_i) = s_i$ and $s(t_f) = s_f$ in a monotonic fashion, i.e. with $\dot{s}(t) \geq 0$ for $t \in [t_i, t_f]$. A possible choice for s is the arc length along the path; in this case it would be $s_i = 0$ and $s_f = L$ where L is the length of the path. The above space time separation implies that

$$\dot{q} = \frac{dq}{dt} = \frac{dq}{ds}\dot{s} = q'\dot{s} = q'V \tag{2.15}$$

where the prime symbol denotes differentiation with respect to s. The generalized velocity vector is then obtained as the product of the vector q', which is directed as the tangent to the path in configuration space, by the scalar $\dot{s}$ that varies its modulus. Nonholonomic constraints (1.71) or (1.78) can then be rewritten as:

$$A(q)\dot{q} = A(q)q'\dot{s} = 0 \tag{2.16}$$

if $\dot{s} \neq 0$ for $t \in [t_i, t_f]$, then:

$$A(q)q' = 0 \tag{2.17}$$

This condition that must be verified at all points by the tangent vector on the configuration space path, characterizes the motion of geometric path admissibility induced by the kinematic constraint that actually affects generalized velocities.

Path planning focuses on finding a path through free space from an initial to a final location. The focus in this section in on turning a sequence of configurations into a smooth curve that is then passed to the control system of the aerial robot [151]. In 2D the curves fall into two categories

- **Curves** whose coordinates have a closed form expressions for example B-splines, quintic polynomials or polar splines
- **Curves** whose curvature is a function of their arc length for example clothoids, cubic spirals, quintic G^2 splines or intrinsic splines

For non holonomic vehicles such as mobile or aerial robots, dynamic model and actuators constraints that directly affect path are used to reject infeasible paths. The term feasible means that the path will be continuously flyable and safe. The flyable path should be smooth, i.e without twists and cusps. The smoothness of the path is determined by amount of bending of the path, measured by curvature and torsion of the path [1, 6]. The purpose of the following paragraphs is to propose a 3D flight path to the aerial vehicle joining the initial and final configurations. This section considers Trim trajectories followed by Maneuvers.

2.3.1 Trim Trajectory Generation

In a trimmed maneuver, the aerial robot is accelerated under the action of non-zero resultant aerodynamic and gravitational forces and moments, these effects are balanced by effects such as centrifugal and gyroscopic inertial forces and moments. Under the trim condition, the aerial robot motion is uniform in the body fixed frame. The trim trajectories have the advantage of facilitating the planning and control problems. The aerodynamic coefficients which are variable in time and space become stationary under this condition and their identification becomes easier [12, 158]. Trim trajectories are characterized by the stationarity of the body-fixed velocity components and the controls. Using (1.59) and (1.61), this condition can be formalized by:

$$\dot{V}(t) \equiv 0 \qquad \dot{\Omega}(t) \equiv 0 \qquad \forall t \in [0, t_f] \tag{2.18}$$

Focusing on the angular velocity kinematics transformation (1.66):

$$\Omega = J(\eta_2)^{-1}\dot{\eta_2} \tag{2.19}$$

Based on the dynamics equations, all forces and moments acting on the aerial robot depending on the velocity vector are constant except the vector of the aerostatic forces and moments τ_s which depends on the attitude variables, the roll ϕ and the pitch θ angles. It follows that, in order to guarantee the stationarity of this vector, the roll angle ϕ_e and pitch θ_e angle must be constant. By deriving the above equation with respect to time and using condition (2.18), the following relations can be written:

$$\begin{aligned} p_e &= -\dot{\psi}_e \sin\theta_e \\ q_e &= \dot{\psi}_e \sin(\phi_e)\cos(\theta_e) \\ r_e &= \dot{\psi}_e \cos(\phi_e)\cos(\theta_e) \end{aligned} \tag{2.20}$$

The components of the body-fixed angular velocity vector depend on the roll angle ϕ_e, pitch angle θ_e and the yaw rate $\dot{\psi}_e$. It is thus possible to characterize the geometry of the trim trajectories as follows:

$$\dot{\eta}_1 = R(\eta_2)\begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix} = R_z(\psi)\begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{2.21}$$

where

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = R_y(\theta_e)R_x(\phi_e)\begin{pmatrix} u_e \\ v_e \\ w_e \end{pmatrix} \tag{2.22}$$

Hence, when the translational kinematics are used, the following relation is obtained:

$$\dot{\eta}_1 = \begin{pmatrix} V_e \cos(\gamma_e) \cos(\dot{\psi}_e t + \psi_0) \\ V_e \cos(\gamma_e) \sin(\dot{\psi}_e t + \psi_0) \\ -V_e \sin(\gamma_e) \end{pmatrix} \tag{2.23}$$

The flight path angle is represented by:

$$\gamma_e = acos\left(\frac{\sqrt{a^2 + b^2}}{V_e}\right) \tag{2.24}$$

The navigation velocity is given $V_e = \|V\|$, and ψ_0 is the initial heading angle. Integrating the above equation, the geometric characterization of trim trajectories can be described by:

$$\eta_1 = \begin{pmatrix} \frac{V_e}{\dot{\psi}_e} \cos(\gamma_e) \sin\left(\frac{\dot{\psi}_e}{V_e} s + \psi_0\right) \\ -\frac{V_e}{\dot{\psi}_e} \cos(\gamma_e) \cos\left(\frac{\dot{\psi}_e}{V_e} s + \psi_0\right) \\ -\sin(\gamma) s \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{2.25}$$

where the integration constants are given by

$$\begin{aligned} x_1 &= x_0 - \frac{V_e}{\dot{\psi}_e} \cos\gamma_e \sin\psi_e \\ y_1 &= y_0 + \frac{V_e}{\dot{\psi}_e} \cos\gamma_e \cos\psi_e \\ z_1 &= z_0 \end{aligned} \tag{2.26}$$

where $(x_0, y_0, z_0)^T$ is the initial position of the aerial robot. It is possible to parametrize the trim trajectory by the vector $\mathcal{T}_e = (\phi_e, \theta_e, \dot{\psi}_e, u_e, v_e, w_e)$ where the subscript e denotes the equilibrium values and the curvilinear abscissa s is considered as for a uniform motion: $s = V_e t$. Depending on the values of γ_e and $\dot{\psi}_e$, the trajectories can be represented by a helix (see Fig. 2.1), a circle arc or a straight line. The above kinematic analysis of the trim trajectories shows that their geometry depends on the body-fixed linear velocity vector V_e, the roll angle ϕ_e, pitch angle θ_e and the rate of yaw angle $\dot{\psi}_e$. The choice of these quantities should satisfy the dynamic equations, the controls saturation and envelope protection constraints.

Curvature and torsion are constant for trim trajectories

$$\begin{aligned} \kappa(s) &= \chi_1 \cos(\gamma_0) \\ \tau(s) &= \chi_1 \sin(\gamma_0) \end{aligned} \tag{2.27}$$

The role of the trajectory generator is to generate a feasible time trajectory for the aerial robot. Once the path has been calculated in the Earth fixed frame, motion must be investigated using the dynamic model and reference trajectories determined

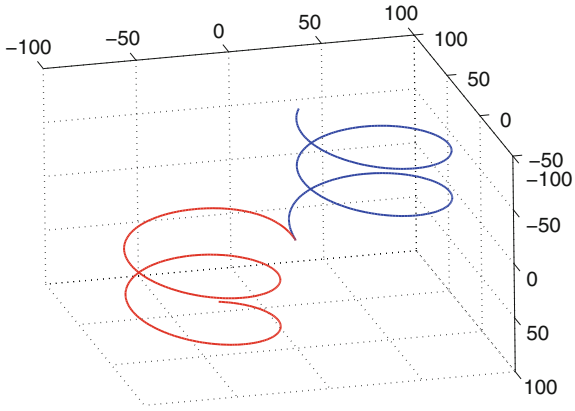

Fig. 2.1 3D helical trim trajectories

taking into account actuators constraints (inequality constraints) and the under-actuation (equality constraints) of an aerial vehicle and limitations on curvature and torsion [10].

2.3.2 Leg-Based Guidance

A leg specifies the flight path to get to a given way-point [8, 81]. There are four different kind of legs:

1. **Basic legs**: specify common leg primitives such as Direct to Fix, Track a Fix.
2. **Iterative legs**: allow for specifying repetitive sequences of legs.
3. **Intersection legs**: provide a junction point for converging paths or a forking point where a decision on what leg to fly can be made.
4. **Parametric legs**: specify legs whose trajectory can be computed given the parameters of a generating algorithm.

Besides basic trajectories, the flight plan specification constructs for conditional and iterative behaviour together with mission oriented area scanning and other built-in patterns. The flight plan submitted to the aerial robot contains a nominal path that can be modified in real-time by updating a number of specific parameters. Moreover, the flight plan can be accompanied by alternatives to respond to emergency situations as well as specific procedures for approach and departure operations. Based on Area Navigation (**RNAV**), the flight plan specification language proposes the leg as main flight component. A leg determines not only a destination point but also the trajectory that the aerial robot should follow in order to reach it. Since most current auto pilots only provide elementary way point guidance, a method for translating flight plan legs to a sequence of waypoints which can be correctly interpreted by the autopilot is needed. A path terminator defines a specific flight path and a specific type of termination for each leg. Leg types are identified by a two letter code that describes

the path (e.g. Heading, course, track…) and the termination condition (e.g the path terminates at a fix, an altitude a given distance to a radiobeacon, a given time....). Besides path terminators, two basic waypoints types exist in RNAV: **Fly-Over** (FO) and **Fly-By** (FB) waypoints. The former entails the actual over fly of the waypoint before heading to the next waypoint. On the other hand, in an Fly-By waypoint, the aerial robot starts turning before reaching the waypoint in such a way that it is obtained a turning arc tangential with two flight segments that precede and follow the waypoint [122].

2.3.3 Dubins and Zermelo Problems

The problem of transferring a dynamical system from an arbitrary initial configuration to a desired target in minimum time is of fundamental interest as an optimal control problem [20]. A family of primitives is deduced from the resolution of this optimization problem [46, 58].

2.3.3.1 Dubins Problem

Dubins [36] characterized curvature constrained shortest path between an initial and final configuration. Dubins problem can be formulated as follows

Problem 2.1. Dubins Problem

$$min \int_0^T dt \tag{2.28}$$

subject to

$$\begin{gathered} \dot{x} = V\cos\chi \qquad \dot{y} = V\sin\chi \qquad \dot{\chi} = \frac{V}{R}u \\ u \in [-1, 1] \end{gathered} \tag{2.29}$$

where R is the minimum turning of the aerial robot and u is the available control.

A vehicle with dynamics as expressed by (2.29) is termed as a Dubins vehicle. The Hamiltonian is classically given by:

$$H = 1 + \lambda_1 V\cos\chi + \lambda_2 V\sin\chi + \lambda_3 \frac{V}{R}u \tag{2.30}$$

where the Lagrange multipliers are represented by $\lambda_1, \lambda_2, \lambda_3$. The application of the necessary condition of optimality gives

$$\begin{aligned} \dot{\lambda}_1 &= 0 \\ \dot{\lambda}_2 &= 0 \\ \dot{\lambda}_3 &= -\lambda_1 V \sin\chi + \lambda_2 V \cos\chi \end{aligned} \tag{2.31}$$

Theorem 2.1. *In 2D, according to Dubins' theorem, a mobile robot minimum time optimal trajectory under maximum control constraint and constant velocity has six solutions {RSL, RSR, LSL, LSR, RLR, LRL} where R represents Right, S: straight and L Left. Such paths are a concatenation of an arc of a minimum-radius circle (either in the positive or negative direction) with either an arc of a minimum radius circle (in the opposite direction) or with a straight segment.*

This Theorem 2.1 gives a complete characterization of the Dubins distance and path between an initial and final configurations with free heading. Each of these paths can be explicitly computed and therefore finding the optimum path and length between any two configurations can be done in constant time [87, 140].

A regular control u_k is a control where the optimality condition $H_{u_k} = 0$ explicitly contains the control u_k, so that $H_{u_k} = \frac{\partial H}{\partial u}(u_k)$ is not identically zero. The Weierstrass condition is a necessary condition for a regular optimal control to be a strong relative minimum. Disturbances in the operational environment make the true trajectory deviate from the planned trajectory and therefore limit the effectiveness of deterministic path planning techniques. This is especially true for small aerial robots [31]. Their slower speed and limited propulsion and control forces make them less capable to directly reject the effect of atmospheric disturbances. Trajectory plan for an aerial robot must incorporate wind as a significant factor that can affect both the feasibility and optimality of trajectory [103].

2.3.3.2 Zermelo Problem

Zermelo's problem was originally formulated to find the quickest nautical path for a ship at sea in the presence of currents, from a given departure point in $\mathbb{R}^2$ to a given destination point [68]. It can also be applied to the particular case of an aerial robot with a constant altitude and a zero flight path angle and the wind velocity represented by $W = (W_x, W_y)$ [133].

2D Zermelo Problem on a Flat Earth

Problem 2.2. 2D Zermelo Problem on a flat Earth Time optimal trajectory generation can be formulated as follows:

$$min \int_0^T dt \tag{2.32}$$

subject to

$$\begin{gathered} \dot{x} = u_1(t) + W_x \qquad \dot{y} = u_2(t) + W_y \\ u_1^2(t) + u_2^2(t) \leq V_{max}^2 \end{gathered} \tag{2.33}$$

If the terminal point is reachable at any time, then it is reachable in the minimal time. However, if the wind is too strong, there may be points that are not reachable at all [90]. The Hamiltonian is classically given by:

$$H = 1 + \lambda_1 \left(u_1(t) + W_x\right) + \lambda_2 \left(u_2(t) + W_y\right) \tag{2.34}$$

where the Lagrange multipliers are represented by λ_1, λ_2. The application of the necessary condition of optimality gives

$$\begin{aligned} \dot{\lambda_1} &= -\lambda_1 \frac{\partial W_x}{\partial x} - \lambda_2 \frac{\partial W_y}{\partial x} \\ \dot{\lambda_2} &= -\lambda_1 \frac{\partial W_x}{\partial y} - \lambda_2 \frac{\partial W_y}{\partial y} \end{aligned} \tag{2.35}$$

Each extremal control $u^*(t)$ must satisfy $\|u^*(t)\| = V_{max}$ for almost all t. The maximality condition yields that

$$u^*(t) = V_{max} \frac{\lambda(t)}{\|\lambda\|} \tag{2.36}$$

for almost all t, as $\lambda(t)$ cannot be identically zero.

Zermelo's navigation formula consists of a differential equation for $u^*(t)$ expressed in terms of only the drift vector and its derivatives. The derivation can be explained as follows. Let the angle $\mu(t)$ given by $u_1(t) = V_{max} \cos \mu(t)$ and $u_2(t) = V_{max} \sin \mu(t)$ then

$$\cos \mu(t) = \frac{\lambda_1}{\|\lambda\|}, \sin \mu(t) = \frac{\lambda_2}{\|\lambda\|} \tag{2.37}$$

Differentiating these relations, the following equalities can be given:

$$\begin{aligned} \dot{\lambda}_1 &= \cos \mu \|\dot{\lambda}\| - \dot{\mu} \|\lambda\| \sin \mu \\ \dot{\lambda}_2 &= \sin \mu \|\dot{\lambda}\| + \dot{\mu} \|\lambda\| \cos \mu \end{aligned} \tag{2.38}$$

Finally the Zermelo navigation equation is given by:

$$\frac{d\mu}{dt} = -\cos^2 \mu \frac{\partial W_x}{\partial y} + \sin \mu \cos \mu \left(\frac{\partial W_x}{\partial x} - \frac{\partial W_y}{\partial y}\right) + \sin^2 \mu \frac{\partial W_y}{\partial y} \tag{2.39}$$

When the problem is to find minimum-time paths through a 2D region of position-dependent vector velocity [64]:

$$\begin{aligned} \dot{x} &= V \sin\chi + W_x(x, y) \\ \dot{y} &= V \cos\chi + W_x(x, y) \end{aligned} \tag{2.40}$$

The heading angle is the control available for achieving the minimum time objective. If $W_x(x, y) = -V_V y$ and $W_y(x, y) = 0$, it has been proved in [64] that the time to go is given by

$$T_{go} = \frac{1}{V_W} \left(\tan\chi_f - \tan\chi\right) \tag{2.41}$$

Some researches have addressed later the problem of optimal path planning of an aerial robot at a constant altitude and velocity in the presence of wind with known magnitude and direction [114, 152]. Such a system is known as a Dubins airplane. A dynamic programming method to find the minimum time waypoint path for an aerial robot flying in known wind was proposed by Jennings [66, 67]. The problem of generating optimal path from an initial position and orientation to a final position and orientation in the 2D plane for an aircraft with bounded turning radius in the presence of a constant known wind. In the absence of wind, this is the Dubins car problem [45]. The original problem of finding the optimal path with no wind to a final orientation is transformed over a moving virtual target whose velocity is equal and opposite to the velocity of the wind. In many real scenarios, the direction of wind is not known a priori or it changes from time to time. Therefore, it can be more relevant to design path planners that are robust to wind disturbances. An approach based on overlaying a vector field of desired headings and then command the aerial robot to follow the vector field was proposed by [110]. A receding Horizon controller was used in [82] to generate trajectories for an aerial robot operating in an environment with disturbances. The proposed algorithm modifies the on-line receding horizon optimization constraints (such as turn radius and speed limits) to ensure that it remains feasible even when the vehicle is acted by unknown but bounded disturbances [48, 101].

3D Zermelo Problem on a Flat Earth

Now, the wind optimal time trajectory planning problem for an aerial vehicle in a 3D space is considered. An aerial robot must travel through a windy region. The magnitude and the direction of the winds are known to be functions of position, i.e. $W_x = W_x(x, y, z)$, $W_y = W_y(x, y, z)$ and $W_z = W_z(x, y, z)$ where (x, y, z) are 3D coordinates and (W_x, W_y, W_z) are the velocity components of the wind. The aerial robot velocity relative to the air is constant V. The minimum-time path from point A to point B is sought. The kinematic model of the aerial robot is

$$\begin{aligned} \dot{x} &= V \cos\chi \cos\gamma + W_x \\ \dot{y} &= V \sin\chi \cos\gamma + W_y \\ \dot{z} &= V \sin\gamma + W_z \end{aligned} \tag{2.42}$$

where χ is the heading angle of the aerial robot relative to the inertial frame, γ is the flight path angle. The Hamiltonian of the minimum time optimization problem is [52, 53]

$$\begin{aligned} H = \lambda_1(V\cos\chi\cos\gamma + W_x) + \lambda_2(V\sin\chi\cos\gamma + W_y) \\ +\lambda_3(V\sin\gamma + W_z) + 1 \end{aligned} \tag{2.43}$$

The Euler-Lagrange equations are

$$\dot{\lambda}_1 = -\frac{\partial H}{\partial x} = -\lambda_1\frac{\partial W_x}{\partial x} - \lambda_2\frac{\partial W_y}{\partial x} - \lambda_3\frac{\partial W_z}{\partial x}, \tag{2.44}$$

$$\dot{\lambda}_2 = -\frac{\partial H}{\partial y} = -\lambda_1\frac{\partial W_x}{\partial y} - \lambda_2\frac{\partial W_y}{\partial y} - \lambda_3\frac{\partial W_z}{\partial y}, \tag{2.45}$$

$$\dot{\lambda}_3 = -\frac{\partial H}{\partial z} = -\lambda_1\frac{\partial W_x}{\partial z} - \lambda_2\frac{\partial W_y}{\partial z} - \lambda_3\frac{\partial W_z}{\partial z}, \tag{2.46}$$

$$0 = \frac{\partial H}{\partial \chi}, \tag{2.47}$$

$$0 = \frac{\partial H}{\partial \gamma}. \tag{2.48}$$

By taking the partial derivative of the Hamiltonian with respect to the heading angle, Eq. (2.47) gives

$$\lambda_1 = \lambda_2\frac{\cos\chi}{\sin\chi}. \tag{2.49}$$

By taking the partial derivative of the Hamiltonian with respect to the flight path angle, Eq. (2.48), the following relationship is obtained

$$\sin\gamma(-\lambda_1\cos\chi - \lambda_2\sin\chi) + \lambda_3\cos\gamma = 0 \tag{2.50}$$

Introducing (2.49) into the previous Eq. (2.50), the following equation can be written:

$$\lambda_3 = \lambda_2\frac{\sin\gamma}{\sin\chi\cos\gamma} \tag{2.51}$$

Introducing the relations (2.49) and (2.51) into relation (2.43), and taking into account that $H = \dot{H} = 0$ the following relationships are obtained

$$\begin{aligned} \lambda_1 &= -\tfrac{1}{\Lambda}\cos\chi\cos\gamma \\ \lambda_2 &= -\tfrac{1}{\Lambda}\sin\chi\cos\gamma \\ \lambda_3 &= -\tfrac{1}{\Lambda}\sin\gamma \end{aligned} \tag{2.52}$$

where the parameter Λ is defined as

$$\Lambda = V + W_x \cos\chi\cos\gamma + W_y \sin\chi\cos\gamma + W_z \sin\gamma \tag{2.53}$$

The relation (2.52) is used to obtain an expression that describes the evolution of the heading.

$$\begin{aligned}\dot{\chi} = \sin^2\chi\frac{\partial W_y}{\partial x} + \sin\chi\cos\chi\left(\frac{\partial W_x}{\partial x} - \frac{\partial W_y}{\partial y}\right)\\ + \sin\gamma\sec\gamma\left(\sin\chi\frac{\partial W_z}{\partial x} - \cos\chi\frac{\partial W_z}{\partial y}\right) - \cos^2\chi\frac{\partial W_x}{\partial y}\end{aligned} \tag{2.54}$$

Now, Eq. (2.52) allows to obtain an expression that describes the evolution of the heading. With $H = \dot{H} = 0$ and differentiating (2.50) with respect to time, the following relationship is obtained:

$$\begin{aligned}-\dot{\lambda}_1\sin\gamma\cos\chi - \lambda_1(-\dot{\chi}\sin\gamma\sin\chi + \dot{\gamma}\cos\chi\cos\gamma)\\ -\dot{\lambda}_2\sin\gamma\sin\chi - \lambda_2(\dot{\chi}\sin\gamma\cos\chi + \dot{\gamma}\sin\chi\cos\gamma)\\ +\dot{\lambda}_3\cos\gamma + \lambda_3(-\dot{\gamma}\sin\gamma) = 0.\end{aligned} \tag{2.55}$$

Replacing (2.52) into the previous Eq. (2.55),

$$\dot{\gamma} = \dot{\lambda}_1\sin\gamma\cos\chi + \dot{\lambda}_2\sin\gamma\sin\chi - \dot{\lambda}_3\cos\gamma \tag{2.56}$$

Substituting $\dot{\lambda}_1, \dot{\lambda}_2, \dot{\lambda}_3$

$$\begin{aligned}\dot{\gamma} = \cos^2\chi\cos\gamma\sin\gamma\frac{\partial W_x}{\partial x} + \sin\chi\cos\gamma\sin\gamma\cos\chi\frac{\partial W_y}{\partial x}\\ + \sin^2\gamma\cos\chi\frac{\partial W_z}{\partial x} + \cos\chi\cos\gamma\sin\gamma\sin\chi\frac{\partial W_x}{\partial y}\\ + \sin^2\chi\cos\gamma\sin\gamma\frac{\partial W_y}{\partial y} + \sin^2\gamma\sin\chi\frac{\partial W_z}{\partial y}\\ - \cos\chi\cos^2\gamma\frac{\partial W_x}{\partial z} + \sin\chi\cos^2\gamma\frac{\partial W_y}{\partial z} + \sin\gamma\cos\gamma\frac{\partial W_z}{\partial z}\end{aligned} \tag{2.57}$$

Now, two different particular cases are considered: constant wind velocity and linear variation of wind velocity:

Constant Wind Velocity

If W_x, W_y and W_z are constant, relation (2.54) implies that $\chi = const.$, i.e., the minimum-time path are straight lines.

Linear Variation of Wind Velocity

If $W_x = \pm V_w y$ and $W_y = W_z = 0$, then relation (2.54) is reduced to

$$\dot{\chi} = \mp V_w \cos^2\chi \tag{2.58}$$

and

$$\dot{\gamma} = \pm V_w \sin\gamma \cos\gamma \sin\chi \cos\chi \tag{2.59}$$

The next paragraph presents the generalization of the Zermelo problem on a curved Earth.

Zermelo Problem on a Curved Earth

The minimum time path solution in spherical coordinates is an extension of the solution to the 3D Zermelo problem on a flat Earth. The solution is derived using spherical Earth coordinates to achieve greater generality. To compute globally optimal routes for any particular origin/destination pair, an iterative approach can be used by varying the initial (of final) heading angle until the computed path passed through both the origin and destination points. This is a viable approach, but it requires the numerical integration of multiple paths to find a solution [127]. To improve the computational efficiency while still generating globally optimal solutions of arbitrary accuracy, families of minimum-time routes are computed by integrating the optimal destination. These families of optimal paths may be interpolated to determine optimal routes to the destination from any point within an area of interest. This solution will be referred to as **Optimal Wind Routing** (OWR). The main benefit of this solution is that it produces globally optimal results in a computationally efficient manner. One computational difference between this and the **Neighboring Optimal Wind Routing** (NOWR) [64] solution is that the families of optimal routes must be computed for each destination. The families of solutions also must be recomputed as the winds change, but these computations may be done periodically off line as often as supported by weather data update rates so that the operational computations are not affected [65, 74]. This section presents the optimal wind routing solution to the Zermelo problem on the surface of a sphere. Consider the problem of choosing $\psi(t)$ to minimize the final time t_f subject to constraints:

$$\begin{array}{lll} \dot{\Phi} = \frac{1}{\cos\theta}\left(\sin\psi + u(\Phi,\Theta)\right) & \Phi(0) = \Phi_0 & \Phi(t_f) = \Phi_f \\ \dot{\theta} = \cos\psi + v(\Phi,\Theta) & \theta(0) = \theta_0 & \theta(t_f) = \theta_f \end{array} \tag{2.60}$$

where Φ is longitude, θ is latitude and ψ is the heading angle from North, u is Easterly wind and v is Northerly wind. The winds are in units of V where V is the airspeed of the aerial robot. Time is in units of R/V where R is the Earth radius. The Hamiltonian is then

$$H = \lambda_\theta \frac{\sin\psi + u(\Phi,\Theta)}{\cos\theta} + \lambda_\theta \cos\psi + v(\Phi,\Theta) + 1 \tag{2.61}$$

The adjoint equations and the optimality condition are:

$$\begin{aligned}\dot{\lambda}_{\Phi} &= -\lambda_{\Phi}\tfrac{u_{\Phi}}{\cos\theta} - \lambda_{\theta} v_{\phi} \\ \dot{\lambda}_{\theta} &= -\lambda_{\Phi}\tfrac{u_{\theta}}{\cos\theta} - \lambda_{\Phi}\tfrac{\tan\Phi}{\cos\theta}(\sin\Psi + u) - \lambda_{\theta} v_{\theta} \\ \dot{\lambda}_{\Psi} &= -\lambda_{\Phi}\tfrac{\cos\Psi}{\cos\theta} - \lambda_{\theta}\sin\psi = 0\end{aligned} \tag{2.62}$$

This gives the following relation

$$A\lambda_{\Phi} + B\lambda_{\theta} = 0 \tag{2.63}$$

where

$$A = \frac{\sin\psi}{\cos\theta}(u_{\theta} - \dot{\psi}) - u_{\Phi}\frac{\cos\psi}{\cos^2\theta} + \frac{\tan\psi}{\cos\theta}(1 + u\sin\psi + v\cos\psi) \tag{2.64}$$

and

$$B = v_{\theta}\sin\psi - \cos\psi(\frac{v_{\Phi}}{\cos\theta} + \dot{\psi}) \tag{2.65}$$

This yields to

$$\begin{aligned}\dot{\psi} = &-\frac{v_{\Phi}\cos^2\Phi}{\cos\theta} + \tan\theta\sin\psi(1 + u\sin\psi + v\cos\psi) \\ &+ \sin\psi\cos\psi\left(-\frac{u_{\phi} + v_{\theta}}{\cos\theta}\right) + u_{\theta}\sin^2\psi\end{aligned} \tag{2.66}$$

It is more convenient to integrate the equations backward from the destination to compute a family of extremals that reach the destination from anywhere within the extremal family domain. Let $T = t_f - t$, then

$$\begin{aligned}\dot{\phi} &= \frac{\sin\psi + u(\phi,\theta)}{\cos\theta} \\ \dot{\theta} &= -\cos\psi - v(\phi,\theta)\end{aligned} \tag{2.67}$$

$$\begin{aligned}\dot{\psi} = &\frac{v_{\phi}\cos^2\psi}{\cos\theta} - \tan\theta\sin\psi(1 + u\sin\psi + v\cos\psi) \\ &+ \sin\psi\cos\psi\left(\frac{u_{\Phi}}{\cos\theta}\right) - u_{\theta}\sin^2\psi\end{aligned} \tag{2.68}$$

These equations are useful for guidance of aerial robots when winds are estimated accurately.

2.3.4 Optimal Control Based Approaches

The planning algorithms for a dynamic system can be cast as an Optimal Control Problem that can be formulated as follows:

$$min_u \int_0^{\infty} q(x,u)dt \tag{2.69}$$

subject to:

$$\dot{x} = f(x,u) \tag{2.70}$$

These optimal control problems have been the primary techniques to plan trajectories in early aerospace applications [26]. Many approaches can be presented depending on the used model of the aerial robot.

2.3.4.1 Time Optimal Trajectories

The subject of this section is to formulate the trajectory generation problem in minimum time as this system has bounds on the magnitudes of the inputs and the states. The velocity is assumed to be linearly variable. As the set of allowable inputs is convex, the time optimal paths result from saturating the inputs at all times (or zero for singular control). For a linear time- invariant controllable system with bounded control inputs, the time-optimal control solution to a typical two point boundary value problem is a bang-bang function with a finite number of switches [11].

For an aerial robot represented by its kinematic model and controls being acceleration, heading and flight path rates, time optimal trajectory generation can be formulated as follows:

$$min \int_0^T dt \tag{2.71}$$

subject to

$$\dot{x} = V\cos\chi\cos\gamma \quad \dot{y} = V\sin\chi\cos\gamma \quad \dot{z} = V\sin\gamma \tag{2.72}$$

$$\dot{V} = u_1 \quad \dot{\chi} = u_2 \quad \dot{\gamma} = u_3 \tag{2.73}$$

initial and final conditions

$$\begin{gathered} x(0) = x_0, y(0) = y_0, z(0) = z_0, V(0) = V_0, \chi(0) = \chi_0, \gamma(0) = \gamma_0 \\ x(T) = x_f, y(T) = y_f, z(T) = z_f, V(T) = V_f, \chi(T) = \chi_f, \gamma(T) = \gamma_f \end{gathered} \tag{2.74}$$

Limitations on the control inputs

$$|u_1| \leq u_{1max} \qquad |u_2| \leq u_{2max} \qquad |u_3| \leq u_{3max} \tag{2.75}$$

and on this state variable

$$|\gamma| \leq \gamma_{max} \tag{2.76}$$

For points that are reachable, the resolution is based on the **Pontryagin Minimum Principle** which constitutes a generalization of Lagrange problem of the calculus

of variations. It is a local reasoning based on the comparison of trajectories corresponding to infinitesimally close control laws [14]. It provides necessary conditions for paths to be optimal. Of course, the kinematic model used below implies a perfect response to the turn commands. A major reason for using the kinematic model is the fact that only necessary conditions for optimality exist for the second order model (given by Pontryagin minimum principle). The Hamiltonian, formed by adjoining the state equation with the appropriate adjoint variable $\lambda_1, \ldots, \lambda_6$, is classically defined as follows:

$$\begin{aligned} H = 1 &+ \lambda_1 V \cos\chi \cos\gamma + \lambda_2 V \sin\chi \cos\gamma \\ &+ \lambda_3 V \sin\gamma + \lambda_4 u_1 + \lambda_5 u_2 + \lambda_6 u_3 \end{aligned} \tag{2.77}$$

where λ represents the Lagrange multiplier. The optimal control input must satisfy the following set of necessary conditions

$$\dot{X} = \frac{\partial H}{\partial \lambda} \qquad \text{where X(0), X(T) are specified} \tag{2.78}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial X} \qquad \text{where } \lambda(0), \lambda(T) \text{ are free} \tag{2.79}$$

With the transversality condition

$$H(T) = 0 \tag{2.80}$$

The co-state variables are free, i.e. unspecified, at both the initial and final times because the corresponding state variables of the system are specified. A first interesting result is the determination of a sufficient family of trajectories, i.e. a family of trajectories containing an optimal solution for linking any two configurations. A first integral of the two point boundary value problem exists and thus the hamiltonian H is constant on the optimal trajectory. Because $H(T) = 0$ from the transversality condition,

$$H(t) = 0, \forall t \in [0, t_f] \tag{2.81}$$

The co-state equations are then obtained in the standard fashion by differentiating the negative of the Hamiltonian with respect to the states.

Lagrange Multipliers Analysis

The first order necessary conditions must be satisfied by formulating the differential equations for the costates. The adjoint equations are the first part of the necessary conditions (2.79) where

$$\begin{gathered}
\dot{\lambda_1} = 0 \\
\dot{\lambda_2} = 0 \\
\dot{\lambda_3} = 0 \\
\dot{\lambda_4} = \lambda_2 \cos\chi\cos\gamma - \lambda_1 \sin\chi\cos\gamma + \lambda_3 \sin\gamma \\
\dot{\lambda_5} = \lambda_2 V \sin\chi\cos\gamma - \lambda_1 V \cos\chi\cos\gamma = \lambda_2(\dot{y} - W_y) - \lambda_1(\dot{x} - W_x) \\
\dot{\lambda_6} = \lambda_2 V \cos\chi\sin\gamma + \lambda_1 V \sin\chi\sin\gamma - \lambda_3 V \cos\gamma
\end{gathered} \tag{2.82}$$

Integrating the first three multiplier dynamics, the following relations are obtained

$$\begin{gathered}
\lambda_1 = \mu\cos\zeta = \text{constant} \\
\lambda_2 = \mu\sin\zeta = \text{constant} \\
\lambda_3 = constant \\
\dot{\lambda}_4 = -\tfrac{\mu}{2} cos\,(\chi + \gamma - \zeta) - \tfrac{\mu}{2} cos\,(\chi - \gamma - \zeta) + \lambda_3 \sin\gamma \\
\dot{\lambda}_5 = -\tfrac{\mu V}{2} sin\,(\chi + \gamma - \zeta) - \tfrac{\mu V}{2} sin\,(\chi - \gamma - \zeta) \\
\dot{\lambda}_6 = \tfrac{\mu V}{2} sin\,(\chi + \gamma - \zeta) + \tfrac{\mu V}{2} cos\,(\chi - \gamma + \zeta) - \lambda_3 V \sin\gamma
\end{gathered} \tag{2.83}$$

and

$$\lambda_5 = \lambda_2 y - \lambda_1 x + \lambda_{50} \tag{2.84}$$

Defining the Hamiltonian and multiplier dynamics in this way, the minimum principle of Pontryagin states that the control variable must be chosen to minimize the Hamiltonian at every instant.

$$H(X, u^*) \leq H(X, u) \tag{2.85}$$

On the optimal trajectory, the optimal control u^* must satisfy:

$$\lambda_4 u_1^* + \lambda_5 u_2^* + \lambda_6 u_3^* \leq \lambda_4 u_1 + \lambda_5 u_2 + \lambda_6 u_3 \tag{2.86}$$

Minimization of the Hamiltonian function subject to the control constraints requires that

$$u_j = -sgn\left\{S_j\right\} = -sgn\left\{\sum_{i=1}^{n} B_{ij}\lambda_i,\ j = 1 \ldots m\right\} \tag{2.87}$$

where S_j is the switching function associated with the jth control input u_j and the signum function is defined as

$$sgn\left\{S_j\right\} = \left\{\begin{array}{l} +1 \ \text{ if } \ S_j > 0 \\ -1 \ \text{ if } \ S_j < 0 \end{array}\right\} \tag{2.88}$$

and it becomes singular for

$$-1 < u_j < +1 \ \text{ if } \ S_j = 0$$

Leading to the following solution

$$\begin{aligned} u_1^* &= -u_{1max} sign(\lambda_4) \\ u_2^* &= -u_{2max} sign(\lambda_5) \\ u_3^* &= -u_{3max} sign(\lambda_6) \end{aligned} \tag{2.89}$$

Thus, the following multipliers can be integrated as

$$\lambda_4 = -\frac{\mu}{2} \frac{\sin(\chi + \gamma - \zeta)}{\delta_2 u_{2max} + \delta_3 u_{3max}} - \frac{\mu}{2} \frac{\sin(\chi - \gamma - \zeta)}{\delta_2 u_{2max} - \delta_3 u_{3max}} + \frac{\lambda_3 \cos\gamma}{\delta_3 u_{3max}} \tag{2.90}$$

$$\lambda_5 = \mu \sin\zeta \left(y - W_y t\right) - \mu \cos\zeta \left(x - W_x t\right) \tag{2.91}$$

Singular Controls

Let u_k denote a component of the control vector U. A regular control u_k is a control where the optimality condition $H_{u_k} = 0$ explicitly contains the control u_k, so that $H_{u_k} u_k$ is not identically zero. The Weierstrass condition is a necessary condition for a regular optimal control to be a strong relative minimum. A singular control u_k occurs when $H_{u_k} u_k \equiv 0$. It is an important case as this Hamiltonian contains control variables linearly. Such paths are a concatenation of an arc of a minimum-radius circle (either in the positive or negative direction) with either an arc of a minimum radius circle (in the opposite direction) or with a straight segment. The generalized Legendre-Clebsch condition

$$(-1)^{\iota} \frac{\partial}{\partial U} \left(\frac{d^2 H_u}{dt^2} \right) \geq 0 \tag{2.92}$$

where ι denotes the order of the singular control. The Weierstrass condition and the Legendre-Clebsch condition can be applied to individual or combinations of regular controls and the generalized Legendre-Clebsch condition can be applied to individual singular controls.

1. For the multiplier λ_4, when the singular control occurs $V = V_{max}$ and $\lambda_4 = \dot{\lambda}_4 = 0$ thus, the relation
$$\lambda_1 x - \lambda_2 y + \lambda_3 z = \lambda_7 \tag{2.93}$$
where λ_7 is an integration constant.
2. Because the values of $\lambda_1, \lambda_2, \lambda_{50}$ are constant, each value of λ_5 defines a line parallel to the characteristic direction, if $W_x = W_y = 0$. The line defined by $\lambda_5 = 0$ is the line on some switching and straight line travel must occur. Straight lines and changing in turning direction on the optimal path must occur on a single line.
$$y = \frac{\lambda_2}{\lambda_1} x - \frac{\lambda_{50}}{\lambda_1} \tag{2.94}$$

3. If both following relations are compared for the singular control in λ_6: Thus

$$\begin{aligned} \dot{x} &= \lambda_1 V \\ \dot{y} &= \lambda_2 V \\ \dot{z} &= \lambda_3 V \end{aligned} \tag{2.95}$$

Four cases can be derived from the previous relations.

- $\lambda_1 = \lambda_2 = \lambda_3 = 0$: The optimal trajectory is represented by a point.
- $\lambda_1 = \lambda_2 = 0$: The optimal trajectory is represented by a straight line along the Down axis.
- $\lambda_1 = \lambda_3 = 0$: The optimal trajectory is represented by a straight line along the North axis.
- $\lambda_2 = \lambda_3 = 0$: The optimal trajectory is represented by a straight line along the East axis.

It is necessary to fuse a boundary-valued optimal control with a singular value.

Numerical Approach

Optimal control problems are often solved numerically via direct methods. In recent years considerable attention has been focused on a class of direct transcription method called pseudo-spectral or orthogonal collocation methods [51]. In a pseudo spectral method, a finite basis of global interpolating polynomials is used to approximate the state at a set of discretization points. The time derivative of the state is approximated by differentiating the interpolating polynomial and constraining the derivative to be equal to the vector field at a finite set of collocation points. Although any set of unique collocation points can be chosen, an orthogonal collocation is chosen, i.e. the collocation points are the roots of an orthogonal polynomial (or linear combinations of such polynomials and their derivatives). Pseudo- spectral methods are commonly implemented via orthogonal collocation. Within the class of pseudo-spectral methods, there are two different implementation strategies: local and global approaches. In a local approach, the time interval is divided into a large number of subintervals called segments or finite elements and a small number of collocation points are used within each segment. The segments are then linked via continuity condition on the state, the independent variable and possibly the control. The rationale for using local collocation is that the discretization points are located so that they support the local behavior of the dynamics. In global collocation form, a single segment is used across the entire interval and the number of collocation points is varied. For smooth problems, global collocation is more accurate than local collocation for a given number of total collocation points. For non smooth problems, the accuracies of global and local collocation methods are comparable. To improve the accuracy of the direct optimization solutions and to enlarge the convergence domain of the indirect methods, a hybrid approach can be proposed to solve the optimal control problem. This cascaded computational scheme has become widely applied. The key

idea is to extract the co-states and other control structure information from a nonlinear programming approach as a first step. The indirect shooting method is then used to refine the solutions.

The three major steps to solve for the optimal maneuver solutions and to validate the results based on the first order optimality conditions are:

1. The kinematic and dynamic differentiation equations are discretized using the Euler or trapezoidal method. Commercially available software is used to get the preliminary and approximate control structures, switching times and initial co-states.
2. Using the results from step 1 as the initial guess, this software is used as a shooting method to solve the two point boundary value problem. The constraints include the final time conditions and the invariance of the Hamiltonian.
3. The results from step 2, together with the originally known initial time state conditions, are used to solve for the dynamic system response by integrating the kinematic and dynamic equations forward in time. The Hamiltonian history and the final state errors are the validation criteria.

This approach can only guarantee that the found solutions are local extrema.

2.3.4.2 Model Predictive Control

One particular approach is to solve this problem

$$min_u \int_0^\infty q(x,u)dt \tag{2.96}$$

subject to:

$$\dot{x} = f(x,u) \tag{2.97}$$

by converting it in a parameter optimization problem: nonlinear programming technique. However, this problem cannot be solved in real time [107]. The family of Model Predictive Control is among the techniques that have been used for real-time optimization. It is essentially a feedback control scheme in which the optimal control problem is solved over a finite horizon $[t, t+T]$ where at each time step t, the future states are predicted over the horizon length T based on the current measurements. The first control input of the optimal sequence is applied to the system and the optimization is repeated [15]. The closed loop implementation provides robustness against modeling uncertainties and disturbances. However, because of the finite horizon, the closed loop stability cannot be guaranteed if no special precautions are taken in the design and implementation. One way to address this issue is to use terminal constraints or cost-to go functions combined with a control Lyapunov Function $V(x)$. The proposed problem is stated as follows:

$$min_u \int_t^{t+T} (q(x,u) + u^T u)dt \tag{2.98}$$

subject to:

$$\dot{x} = f(x,u) \tag{2.99}$$

$$V(x(t+T)) \leq V(x_\sigma(t)) \tag{2.100}$$

$$\frac{\partial V}{\partial x}[f(x,u)] \leq -\epsilon\sigma(x(t)) \tag{2.101}$$

where $0 \leq \epsilon \leq 1$, $\sigma(x(t))$ is a continuous positive definite function T is the horizon length.

An alternative approach is through the use of an a priori Control Lyapunov function as terminal cost of the optimization (or cost-to-go) rather than imposing additional constraint to the problem. The proposed unconstrained receding horizon control and control Lyapunov function is computationally faster and also the stability is guaranteed as long as the Control Lyapunov function is an upper bound on the cost to go function.

$$min_u \int_T^0 (q(x,u)dt + V(x(T)) \tag{2.102}$$

subject to:

$$\dot{x} = f(x,u) \tag{2.103}$$

where $V(.)$ is a non negative continuously differentiable function with $V(0) = 0$ satisfying $V(x) \geq c||x||^2$ such that

$$min_u(\dot{V} + q)(x,u) \leq 0 \tag{2.104}$$

is exponentially stable.

With uncertainties in the system behavior, the state evolution will not match the prediction. However, the Model predictive control have some inherent implicit robustness to uncertainties due to the closed-loop implementation [137].

2.3.4.3 Optimal Guidance Approach

For aerial robot applications that involve significant interactions with their environment, the trajectory has to remain valid to changes in problem specifications and be robust to disturbances and contingencies like unknown terrain features. Gust disturbances can be significant for small- scale aerial robots, since they can easily overpower their own propulsion and control forces. The load factor limit constraints the available acceleration for avoidance maneuver. In contrast to classical guidance problems, the amount of information involved in the problem specifications and the

time rate of change of this information will be much higher than current aerial robots scenarios. A fully specified 3D environment and nonlinear aerial robot dynamics will lead to a nonlinear high dimensional stochastic optimization problem that cannot be solved with sufficient speed to allow reactive and interactive capabilities. For many aerial robotic applications, the ability to take full advantage of available information can be critical. Achieving a guidance behavior that can take into account, in real time, both the on board sensory information and the evolving knowledge of the global environment and task will require a more significant advances in technology than those enabled by higher computational power. The key challenge in autonomous guidance is therefore determining a problem formulation that integrates the on line, sensory-based planning and the global environment, or knowledge-based planning, in a framework that is both practical and theoretical sound. One approach to dealing with environmental change is to use situation dependent rules that determine costs or probabilities of actions as a function of the environment. Changes to the environment are expressed in rules that capture the fact that actions have different costs under different conditions [53]. Such rules can be learned using a process that processes execution episodes situated in a particular task context, identifying successes and failures, and then interpreting this feedback into reusable knowledge. The execution level defines the set of available situations features F while the planner level defines a set of relevant learning events and a cost function C for evaluating these events.

Events are learning opportunities in the environment from which additional knowledge will cause the planner's behavior to change. Features discriminated between those events, thereby creating the required additional knowledge. The cost function allows the learner to evaluate the event. The learner then creates a mapping from the execution features and the events to the costs: $F \times \epsilon \rightarrow C$. For each event $\epsilon \in \epsilon$ in a given situation described by features F, this learned mapping predicts a cost $c \in C$ that is based on prior experience. This mapping is called a situation dependent rule. Each planner in the system can then use the learned rules to create plans tailored to the situations it encounters later in its environment. The approach is relevant to planners that would benefit from feedback about plan execution.

Optimal Guidance Problem

Trajectory optimization involves determining a control history $u(t) \in \mathbb{R}^m$ that will fly the aircraft from a given initial state x_0 to a goal state x_{goal} [144, 145]. The duration can be fixed $t \in [0 \ldots T]$ or unspecified as in a minimum time problem. The path is a trajectory that has to satisfy the aerial robot dynamics, usually available in the form of a vector differential equation. The desired qualities of the trajectory is usually specified using a performance objective of cost function that will be minimized. The performance objective describes the cost of the trajectory starting from a state x when the system is driven by a control trajectory u. The optimal guidance problem

$$min_{u(t)} J(x, u) = \int_0^\infty g(x(\tau), u(\tau))\, d\tau \tag{2.105}$$

subject to

$$\begin{gathered} \dot{x}(t) = f\left(x(t), u(t)\right) \\ x\left(t_0\right) = x_0; x\left(t_\infty\right) = x_{goal} \\ u(t) \in U; x_p(t) \in X_{free}; x_v(t) \in X_{aircraft} \end{gathered} \tag{2.106}$$

where g is a scalar function that represents the instantaneous trajectory cost. The vector $x = \begin{pmatrix} x_p(t) \\ x_v(t) \end{pmatrix} \in \mathbb{R}^n$ is the aerial robot state. X_{free} represents the admissible region of the geographical environment, U is the set of admissible controls. The optimal control trajectory $u^*(t)$ for an optimal state $x*(t)$ is formulated as:

$$u_\infty = argmin\left(J_\infty(x)\right) \tag{2.107}$$

To solve this optimal guidance problem, all elements must be specified and then converted into a numerical optimization problem. Such explicit representations are very expensive. The challenge with most of the numerical formulations of optimal guidance problem is that they do not exploit characteristics of the guidance problem and therefore lead to high computational complexity and at the same time inflexible implementations.

In receding horizon optimization, the optimization is only performed over a finite time horizon H. The remaining part of the trajectory is ignored. The classical collocation method is one among the many different methods to find solutions for optimal control problems. However, it is difficult to use this approach for real-time applications since it is based on discretization which results in a large number of unknowns to be solved simultaneously. The problem is solved repeatedly to obtain the control action based on the most up-to-date state information. The trajectory update rate and the horizon length is usually fixed. The technique that is the most consistent with the original infinite horizon form is to approximate the discarded tail of the optimization by a cost-to-go function. The optimal input trajectory in the receding horizon with cost to go scheme is that it minimizes the composite cost:

$$u_H^*(t) = Argmin\left(J_H\left(x(t), u(t)\right) + V\left(x(t+H)\right)\right) \tag{2.108}$$

where $J_H\left(x(t), u(t)\right)$ is the finite horizon cost

$$J_H\left(x(t), u(t)\right) = \int_t^{t+H} g\left(x(\tau), u(\tau)\right) d\tau \tag{2.109}$$

and $V\left(x(t+H)\right)$ represents the cumulative cost of the discarded tail of the trajectory, i.e. the cost to go. It captures the cost of the trajectory from the state attained at the horizon end $x\left(t+H\right)$.

Computation of Spatial Value Functions

The spatial value function captures the interaction between spatial characteristics and the dynamic behavior of the aerial robot. It differs from the value function, because it is not defined over the dynamic system's entire state-space but only defines the relationship between space (the x, y, z coordinates) and the optimal velocity V^*. The optimal spatial value function describes the optimal velocity field and associated cost to go over the geographical space. One form of finite state representation that has been proposed for aerial robot guidance is the motion primitive automaton. This model was originally proposed as part of a hybrid guidance system. The motion primitives define the state and control actions space. Therefore, the richness of the set of primitives has a direct impact on the system's behavior. The curse of dimensionality typically limits the practical use of such models. The kinematic system of the aerial robot can be reduced to a set of feasible trim trajectories and maneuvers. The grid-based, finite state motion primitive automaton is a particular form of quantization of the aerial robot dynamics where the motion primitives are constrained to share a common spatial grid. Using motion primitives defined on a fixed spatial grid prevents from having to perform costly interpolations during the value iteration [1]. The velocities are discretized based on the flight performance envelope. The state space is discretized at a regular resolution:

- d_{xy} set to the minimum turn radius for the lowest, non-zero speed.
- d_z set to be consistent with lowest, non-zero horizontal speed and lowest non-zero ascent/descent speed.
- d_ψ heading resolution of 45°.

Any motion primitive must be a combination of one horizontal motion primitive and one vertical motion primitive. Regarding the horizontal motion primitives, their initial and final heading must be a multiple of 45°. Two forms of motion primitives are used: rectilinear and turns. Each can be of constant speed or accelerating/decelerating. They are described by the translational and rotational attributes and the connectivity information. The resultant motion primitives consist of the set of a horizontal motion primitive and a vertical motion primitive. They cannot be grouped arbitrarily, a mechanical capability check and safety check must be performed to determine acceptable combinations [20]. The use of a spatial value function as a working principle for the guidance system provides a systematic way to integrate the inner-loop control system and the aerial robot dynamics with the trajectory planning process [102].

2.3.5 Parametric Curves

A flight trajectory can be defined as follows [99]:

Definition 2.1. Flight trajectory A flight trajectory from $E_0 = (s_0, \dot{s}_0)$ to $E_f = (s_f, \dot{s}_f)$ in $\mathbb{R}^n$ is a continuously differentiable parametric oriented curve Θ_{E_0,E_f} in the time variable t, connecting s_0 to s_f with assigned first derivatives (tangents) $\dot{s}_0, \dot{s}_f$ at the extremes

$$\Theta_{E_0,E_f} = \left\{ s \in C^1_{[t_0,t_f]} : s(t_0) = s_0, s(t_f) = s_f, \dot{s}(t_0) = \dot{s}_0, \dot{s}(t_f) = \dot{s}_f \right\} \quad (2.110)$$

The inputs of this path planning algorithm are

- the initial configuration and velocity: $x_i, y_i, z_i, \gamma_i, \chi_i, V_i$
- the final configuration and velocity: $x_f, y_f, z_f, \gamma_f, \chi_f, V_f$.

Some curves have been studied for aerial robots at constant altitude to accomplish their mission. As already shown, Dubins curves are CSC or CCC curves along which the aerial robot moves forward [32]. Other curves such Cartesian polynomials, $\beta-$splines, Pythagorean hodographs, $\eta-$splines are proposed in this section.

2.3.5.1 Cartesian Polynomials

Parametric cubic curves are popular because they are the lowest degree polynomial curves that allow inflection points (where curvature is zero) so they are suitable for the composition of G^2 blending curves with second order geometric continuity. Sudden changes between curves of widely different radii or between long tangents and sharp curves should be avoided by the use of curves gradually increasing or decreasing radii [54]. This problem can be solved by interpolating via Cartesian polynomials using the following cubic polynomials versus a normalized arclength $0 \leq s \leq 1$

$$\begin{aligned} x(s) &= s^3 x_f - (s-1)^3 x_i + \alpha_x s^2 (s-1) + \beta_x s (s-1)^2 \\ y(s) &= s^3 y_f - (s-1)^3 y_i + \alpha_y s^2 (s-1) + \beta_y s (s-1)^2 \\ z(s) &= s^3 z_f - (s-1)^3 z_i + \alpha_z s^2 (s-1) + \beta_z s (s-1)^2 \end{aligned} \quad (2.111)$$

that automatically satisfy the boundary conditions on x, y, z. The orientation at each point being related to x', y', z', it is also necessary to impose the additional boundary conditions

$$\begin{aligned} x'(0) &= V_i \cos\gamma_i \cos\chi_i \\ y'(0) &= V_i \cos\gamma_i \sin\chi_i \\ z'(0) &= V_i \sin\gamma_i \end{aligned} \quad (2.112)$$

and

$$\begin{aligned} x'(1) &= V_f \cos\gamma_f \cos\chi_f \\ y'(1) &= V_i \cos\gamma_i \sin\chi_i \\ z'(1) &= V_i \sin\gamma_i \end{aligned} \tag{2.113}$$

The geometric inputs that drive the robot along the Cartesian path are

$$\begin{aligned} V(s) &= \pm\sqrt{(\dot{x})^2 + (\dot{y})^2 + (\dot{z})^2} \\ &= \tfrac{1}{T}\sqrt{(x'(s))^2 + (y'(s))^2 + (z'(s))^2} \end{aligned} \tag{2.114}$$

Resolving for the various parameters with the boundary conditions, the following equalities are obtained

$$\begin{aligned} \alpha_x &= \cos\gamma_f \cos\chi_f - 3x_f \\ \alpha_y &= \cos\gamma_f \sin\chi_f - 3y_f \\ \alpha_z &= \sin\gamma_f - 3z_f \end{aligned} \tag{2.115}$$

and

$$\begin{aligned} \beta_x &= \cos\gamma_i \cos\chi_i + 3x_i \\ \beta_y &= \cos\gamma_i \sin\chi_i + 3y_i \\ \beta_z &= \sin\gamma_i + 3z_i \end{aligned} \tag{2.116}$$

The evolution of the aerial robot orientation along the path can then be computed for the flight path angle

$$\gamma = \arcsin\left(\frac{dz/ds}{V(s)}\right) \tag{2.117}$$

and the heading angle

$$\chi = \arctan_2\left(\frac{dx}{ds}, \frac{dy}{ds}\right) + k\pi; \quad k = 0, 1 \tag{2.118}$$

The two possible choices for k account for the fact that the same cartesian path may be followed moving forward ($k = 0$) or backward ($k = 1$). If the initial orientation is assigned, only one of the choices for k is correct. This approach can be easily generalizable to a fourth or fifth polynomial. However the main drawback is a complicated formulation of the curvature and the torsion making control of smoothness (twists and cusps) a difficult task. The approach followed in the following subsection aims to propose an easier formulation of these two parameters.

2.3.5.2 β Splines

Cubic splines are degree three polynomials that smoothly connect to adjoining spline polynomials. They provide smoothness and continuity by ensuring that the value of the function, its first and second derivative match with those of neighboring splines

[129]. Cubic splines are continuously differentiable up to the third derivative. A large safety margin accounting for path following errors can therefore be avoided. Furthermore, the path planner can assume a collision-free path sphere volume as a safety radius of a selectable radius. It may also insert other geometries into spline segments. A univariate polynomial spline is defined as a piecewise polynomial function. In its most general form, a polynomial spline $S : [a, b] \rightarrow \mathbb{R}$ consists of polynomial pieces $P_i : [\tau_i, \tau_{i+1}] \rightarrow \mathbb{R}$ where a strictly increasing sequence of real numbers is used between the boundaries a, b as:

$$a = \tau_0 < \tau_1 < \cdots < \tau_{k-1} = b \tag{2.119}$$

that is

$$\begin{aligned} S(\tau) &= P_0(\tau) \quad \tau_0 \leq \tau \leq \tau_1 \\ S(\tau) &= P_1(\tau) \quad \tau_1 \leq \tau \leq \tau_2 \\ &\cdots \\ S(\tau) &= P_{k-2}(\tau) \quad \tau_{k-2} \leq \tau \leq \tau_{k-1} \end{aligned} \tag{2.120}$$

The given k points τ_k are called knots. The vector $\tau = [\tau_0, \ldots, \tau_{k-1}]$ is called a knot vector for the spline. When the knots are not equidistantly distributed in the interval $[a, b]$, the spline is therefore called to be non-uniform. If the above described interpolation method is applied to a large number of waypoints comparatively high order splines have to be selected for a feasible interpolation. Consequently, oscillations between the support points may occur. Therefore, a cubic spline is applied piecewise for each segment. This way, a transition condition at each segment boundary ensures smoothness up to the second derivative. A third order spline is selected for each segment k and each degrees of freedom $i = [x, y, z]$

$$S_{i,k}(t) = a_{i,k} + b_{i,k}(\tau - \tau_k) + c_{i,k}(\tau - \tau_k)^2 + d_{i,k}(\tau - \tau_k)^3 \tag{2.121}$$

$S_{i,k}(t)$ represents the spline for the dimension i in segment k for a specific τ. The n waypoints $P_j(x, y, z)$, $j = 1, \ldots, n$ are the support points of the spline interpolation. A number of n points causes $k = (n - 1)$ segments. Each segment contains three spline functions including four parameters to be determined by four relations. The following requirements for each segment boundary are defined:

- Consecutive spline segments must be connected with each other.
- The first and second derivative must be continuous.

The quadratic Bezier curve with three control points P_0, P_1, P_2 is given by [39]:

$$Q(s) = (1 - s)^2 P_0 + 2s(1 - s)P_1 + s^2 P_2 \quad 0 \leq s \leq 1 \tag{2.122}$$

The total range is divided into N subintervals each of length ΔR. Now (N+1) data points and (N−1) interior points exist. Let a cubic spline on the ith interval $R_i \leq R \leq R_{i+1}$, be defined as

$$H_i(R) = a_i R^3 + b_i R^2 + c_i R + d_i \tag{2.123}$$

which expresses one of the degrees of freedom (x, y or z) in that interval. The values $h_1, h_2, \ldots, h_{N-1}$ at the knot points are the optimizing parameters; The values h_0 and h_N and the gradients at R_1 and R_{N-1}. Aerial robot constraints are enforced not only at the knot points but at intermediate points along the spline also. Mathematically, the spline continuity and smoothness constraints are written as:

$$H_{i-1}(R_i) = H_i(R_i) \tag{2.124}$$

$$\frac{dH_{i-1}(R)}{dR}|_{R_i} = \frac{dH_i(R)}{dR}|_{R_i} \tag{2.125}$$

$$\frac{d^2H_{i-1}(R)}{dR^2}|_{R_i} = \frac{d^2H_i(R)}{dR^2}|_{R_i} \tag{2.126}$$

Boundary conditions of the first and last splines are

$$\begin{aligned} H_0(0) &= h_0 \\ H_{N-1}(R_N) &= h_N \end{aligned} \tag{2.127}$$

$$\frac{dH_0(R)}{dR}|_{R_1} = S_1 \tag{2.128}$$

$$\frac{dH_{N-1}(R)}{dR}|_{R_{N-1}} = S_{N-1} \tag{2.129}$$

Thus a total of 4N constraints exist to compute the N splines; the splines having 4 coefficients each. The optimization problem is now to minimize

$$J = \int_0^{R_F} e^2 dR \tag{2.130}$$

where R_F is the total range value (the total distance to be travelled). The error e has to be positive. The slope is defined as:

$$slope = \frac{dh}{dR} = \frac{dh/dt}{V} \tag{2.131}$$

with the constraint $slope_{min} \leq slope \leq slope_{max}$. The curvature κ is defined as

$$\kappa = \frac{dslope}{dR} = \frac{d^2h/dt^2}{V^2} \tag{2.132}$$

with the constraint $\kappa_{min} \leq \kappa \leq \kappa_{max}$. The kink p is defined as:

$$p = \frac{d\kappa}{dR} \tag{2.133}$$

with the constraint $p_{min} \leq p \leq p_{max}$.

This is a nonlinear programming problem and different algorithms can be employed. In [129], a sequential quadratic programming (SQP) is used to solve this problem. At each major iteration, an approximation of the Hessian of the Lagrangian function is made which is then used to form a search direction for a line search procedure [61]. Each Quadratic Programming (QP) suproblem minimizes a quadratic approximation of the constraints. The approximations are carried out using Taylor series expansion. The QP subproblem is to minimize the quadratic objective function;

$$F(h) = \frac{1}{2}\bar{h}^T H\bar{h} + \bar{y}\bar{h} \tag{2.134}$$

subject to linear equality and inequality constraints:

$$\begin{aligned} A\bar{h} &= a \\ B\bar{h} &\leq b \end{aligned} \tag{2.135}$$

where $\bar{h} = (h_1, \ldots, h_{N-1})^T$ denotes the optimizing parameter vector, y is the gradient vector, H is the positive definite Hessian matrix and A, B, a, b are matrices of appropriate dimensions defining the constraints.

2.3.5.3 Pythagorean Hodograph

Curves with simple form descriptions of their parametric speed and arc length are useful. The interesting class of Pythagorean Hodograph is distinguished by having a polynomial arc length function and rational offset curves [42]. The Pythagorean Hodograph curves are introduced to provide exact solutions to a number of basic computational problems that arise in path planning. These include analytic reduction of the arc length and bending energy integrals, construction of rational offset (parallel) cuves, formulation of real time interpolator for motion control applications; determination of rotation minimizing frames for specifying orientations of rigid bodies along spatial paths. The arclength of a PH curve can be computed precisely by evaluating a polynomial. Digital motion control along curved paths, with fixed or variable speeds is an application context for PH curves. The interpolation integral which defines times reference points on a curved path in accordance with the specified speeds admits an analytic reduction for PH curves, yielding real-time interpolation [62]. A polynomial space curve is said to Double Pythagorean Hodograph (DPH) if the Frenet frame, the curvature and the torsion have all a rational dependence on the curve parameter.

2 D Pythagorean Hodograph

In 2D, a Pythagorean Hodograph $r(t) = (x(t), y(t))$ is a polynomial curve whose tangents $\dot{x}(t)$ and $\dot{y}(t)$ satisfies

$$\dot{x}^2(t) + \dot{y}^2(t) = \sigma^2(t) \tag{2.136}$$

for some polynomial $\sigma(t)$. From the principles of differential geometry, the path length s and parametric speed $\dot{s}$ of a parametric curve are given by:

$$s = \int_{t_1}^{t_2} \|\dot{r}(t)\| \, dt = \int_{t_1}^{t_2} \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} dt \tag{2.137}$$

If the sum of square of the tangents $\dot{x}(t)$, $\dot{y}(t)$ could be represented by perfect square of a single polynomial, it leads to two advantages:

- The radical form for calculating the path length is eliminated
- The parametric speed of the curve is simply a polynomial function of the parameter t.

This is achieved by forming the hodograph of the curve $r(t)$ of the form:

$$\begin{aligned} \dot{x}(t) &= w(t)\left(u^2(t) - v^2(t)\right) \\ \dot{y}(t) &= 2w(t)u(t)v(t) \end{aligned} \tag{2.138}$$

Equation (2.136) becomes

$$\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} = \sigma(t) = w(t)\left(u^2(t) + v^2(t)\right) \tag{2.139}$$

where $u(t)$, $v(t)$, $w(t)$ and $\sigma(t)$ are non zero real polynomials, satisfying $gcd(u(t), v(t)) = 1$ (greater common divisor). Now, Eq. (2.137) becomes

$$s = \int_{t_1}^{t_2} \sigma(t) dt \tag{2.140}$$

Depending on the order of the polynomials $u(t)$, $v(t)$, $w(t)$, the Pythagorean Hodograph curve can be either cubic, quartic or quintic. The expressions for Frenet-Serret frame unit tangent T, unit normal N and curvature κ of a Pythagorean Hodograph curve are:

$$T = \frac{\left(u^2 - v^2, 2uv\right)}{u^2 + v^2} \quad N = \frac{\left(2uv, v^2 - u^2\right)}{u^2 + v^2} \tag{2.141}$$

$$\kappa = \frac{1}{w\left(u^2+v^2\right)}\left(2\left(u\dot{v}-v\dot{u}\right), 2uv\right) \tag{2.142}$$

The off-set curve $r_0(t)$ of the curve $r(t)$ at a distance $\pm d$ is

$$r_0(t) = r(t) \pm dN(t) \tag{2.143}$$

3 D Pythagorean Hodograph

In 3 D, the Pythagorean Hodograph is defined by a parametric velocity $\dot{s} = \sqrt{\dot{x}^2(t)+\dot{y}^2(t)+\dot{z}^2(t)}$ polynomial in t. The arc length of the Pythagorean Hodograph curve can be computed precisely by evaluating a polynomial. For the hodograph $\dot{r} = (\dot{x}(t), \dot{y}(t), \dot{z}(t))$, it is necessary and sufficient that its components be expressible in terms of polynomials $u(t)$, $v(t)$, $p(t)$, $q(t)$ in the form

$$\begin{aligned} \dot{x}(t) &= u^2(t)+v^2(t)-p^2(t)-q^2(t) \\ \dot{y}(t) &= 2\left(u(t)q(t)+v(t)p(t)\right) \\ \dot{z}(t) &= 2\left(v(t)q(t)-u(t)p(t)\right) \\ \sigma(t) &= u^2(t)+v^2(t)+p^2(t)+q^2(t) \end{aligned} \tag{2.144}$$

If $u(t)$, $v(t)$, $p(t)$, $q(t)$ are all constants, the hodograph is a single point specifying a uniformly parameterized straight line.

The simplest non trivial Pythagorean Hodograph are cubic, they correspond to segments of non circular helices with a constant ratio $\frac{\kappa}{\tau}$. They may be characterized by certain geometrical constraints on these Bezier control polygons.

$$r(t) = \sum_{k=1}^{3} b_k \binom{3}{k} (1-t)^{3-k}\, t^k \quad t \in [0,1] \tag{2.145}$$

To guarantee sufficient shape flexibility, fifth order Pythagorean Hodograph curve may be employed. The construction of spatial Pythagorean Hodograph fifth order is described, in Bernstein-Bezier form

$$r(t) = \sum_{k=1}^{5} b_k \binom{5}{k} (1-t)^{5-k}\, t^k \qquad t \in [0,1] \tag{2.146}$$

where $b_k = (x_k, y_k, z_k)$ are control points, whose vertices define the control polygon or Bezier polygon, t is a parameter and $k = 0 \ldots 5$. Knowing that

$$\int \binom{n}{k} (1-t)^{n-k}\, t^k dt = \frac{1}{n+1} \sum_{i=k+1}^{n+1} (1-t)^{n+1-i}\, t^i \tag{2.147}$$

and

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2.148}$$

For Pythagorean Hodograph fifth orders, $\left|r' \times r''\right| = \sigma^2 \iota$ and $\iota = \left(up' - u'p + vq' - v'q\right)^2 + \left(uq' - u'q + v'p - vp'\right)^2$. For Pythagorean Hodograph fifth order, $\left(r' \times r''\right) \cdot r'''$ is of degree 6 while σ, ι are both in degree 4 in t. For a Pythagorean Hodograph curve with $\iota = \omega^2$, the Frenet-Serret frame vectors and the curvature and torsion functions are given by the rational expressions:

$$T = \frac{r'}{\sigma} \quad N = \frac{\sigma r'' - \sigma' r'}{\sigma\omega} \quad B = \frac{r' \times r''}{\sigma\omega} \tag{2.149}$$

$$\kappa = \frac{\omega}{\sigma^2} \quad \tau = \frac{\left(r' \times r''\right) \cdot r'''}{\sigma^2\omega^2} \tag{2.150}$$

Hence, the Pythagorean Hodograph curves may be regarded as the complete set of polynomial curves that have rational Frenet frame.

Let the initial and final configurations be $(x_i, y_i, z_i, \chi_i, \gamma_i)^T$ and $\left(x_f, y_f, z_f, \chi_f, \gamma_f\right)^T$. The four control points of the Bezier polygon are calculated by first order Hermite interpolation as follows:

$$\begin{gathered} b_0 = (x_i, y_i, z_i) \\ b_5 = \left(x_f, y_f, z_f\right) \\ b_1 = b_0 + \tfrac{1}{5}\left(\cos\chi_i \cos\gamma_i, \sin\chi_i \cos\gamma_i, \sin\gamma_i, \right) \\ b_4 = b_5 - \tfrac{1}{5}\left(\cos\chi_f \cos\gamma_f, \sin\chi_f \cos\gamma_f, \sin\gamma_f, \right) \end{gathered} \tag{2.151}$$

The control points b_0, b_1, b_4, b_5 are fixed. Now the problem is reduced to finding the control points b_2, b_3. Both polynomials curves are given by

$$\begin{aligned} u(t) &= u_0(1-t)^2 + 2u_1(1-t) + u_2t^2 \\ v(t) &= v_0(1-t)^2 + 2v_1(1-t) + v_2t^2 \\ p(t) &= p_0(1-t)^2 + 2p_1(1-t) + p_2t^2 \\ q(t) &= q_0(1-t)^2 + 2q_1(1-t) + q_2t^2 \end{aligned} \tag{2.152}$$

The set of equations to be solved for the control points b_2, b_3 results in four solutions. Among these four paths, only one has an acceptable shape that is without twists and cusps. This path will be used as reference and is identified by calculating the bending energy of the curve and choosing the path whose energy is minimal. When the polynomials $u(t)$, $v(t)$, $p(t)$, $q(t)$ are of degree μ at most, the spatial PH curve obtained by integration of this hodograph is of odd degree $n = 2\mu + 1$. If the polynomials $u(t)$, $v(t)$, $p(t)$, $q(t)$ are specified in terms of Bernstein coefficients on $t \in [0, 1]$, the Bezier control points of the spatial PH curves they define can be

Algorithm 1 Pythagorean Hodograph Algorithm

1. **Inputs**: Endpoints P_0, P_1 and endpoint derivatives (velocity vectors) V_0, V_1. All these data are considered as complex numbers, by identifying the plane with the Argand diagram [27]
2. **Outputs**: PH quintic $p(\tau)$ defined over the interval [0, 1] and interpolating the input
3. Transform the data to a certain canonical position

$$\tilde{V}_0 = \frac{V_0}{P_1 - P_0} \quad \tilde{V}_1 = \frac{V_1}{P_1 - P_0} \tag{2.153}$$

4. Compute the control points of the so-called preimage

$$\begin{gathered} w_0 = \sqrt{\tilde{V}_0}^{+} \\ w_2 = \sqrt{\tilde{V}_1}^{+} \\ w_1 = \tfrac{1}{4}\left[-3(w_0 + w_2) + \sqrt{120 - 15(\tilde{V}_0 + \tilde{V}_1) + 10 w_0 w_2}^{+}\right] \end{gathered} \tag{2.154}$$

   where $\sqrt{}^{+}$ denotes the square root with the positive real part.
5. Compute the control points of the hodograph i.e. the first derivative and transform it back to the original position

$$\begin{gathered} h_0 = w_0^2 (P_1 - P_0) \\ h_1 = w_0 w_1 (P_1 - P_0) \\ h_2 = (\tfrac{2}{3} w_1^2 + \tfrac{1}{3} w_0 w_2)(P_1 - P_0) \\ h_3 = w_2 w_1 (P_1 - P_0) \\ h_4 = w_2^2 (P_1 - P_0) \end{gathered} \tag{2.155}$$

6. Compute the control points of the PH interpolant

$$p_0 = P_0 \quad p_i = p_{i-1} + \frac{1}{5} h_{i-1} \tag{2.156}$$

   and return the PH curve in Bernstein-Bezier representation

$$p(\tau) = \sum_{i=0}^{5} p_i \binom{5}{i} \tau^i (1 - \tau)^{5-i} \tag{2.157}$$

expressed in terms of these coefficients. For example, if $u(t) = u_0 (1 - y) + u_1 t$, and similarly for $v(t)$, $p(t)$, $q(t)$, the control points of spatial PH cubics are of the form

$$P_1 = P_0 + \frac{1}{3} \begin{pmatrix} u_0^2 + v_0^2 - p_0^2 - q_0^2 \\ 2\,(u_0 q_0 + v_0 p_0) \\ 2\,(v_0 q_0 + u_0 p_0) \end{pmatrix} \tag{2.158}$$

$$P_2 = P_1 + \frac{1}{3} \begin{pmatrix} u_0 u_1 + v_0 v_1 - p_0 p_1 - q_0 q_1 \\ u_0 q_1 + q_0 u_1 + v_0 p_1 + p_0 v_1 \\ v_0 q_1 + q_0 v_1 - u_0 p_1 - p_0 u_1 \end{pmatrix} \tag{2.159}$$

$$P_3 = P_2 + \frac{1}{3}\begin{pmatrix} u_1^2 + v_1^2 - p_1^2 - q_1^2 \\ 2\,(u_1 q_1 + v_1 p_1) \\ 2\,(v_1 q_1 + u_1 p_1) \end{pmatrix} \tag{2.160}$$

The point corresponding to the integration constants, being freely chosen. Degree-n spatial PH curves possess $2n+5$ degrees of freedom. However, ten of these freedoms correspond to a choice of origin and orientation of the coordinate axes in $\mathbb{R}^3$. The pseudo code of this approach is presented in Algorithm 1 in page 95.

Its parametric speed is a polynomial, possibly piecewise

$$\left\| p'(\tau) \right\| = \|w(\tau)\|^2 \, |p_1 - p_0| \tag{2.161}$$

where

$$w(\tau) = w_0(1-\tau)^2 + 2w_1\tau(1-\tau) + w_2\tau^2 \tag{2.162}$$

is the so-called pre-image.

2.3.5.4 η^3 Splines

The η^3 Splines can represent any seventh order polynomial curve with third order geometric continuity denoted as G^3 continuity: continuous in position, curvature and derivative of the curvature [119].

Problem 2.3. 3D Problem Determine the minimal order polynomial curve which interpolates two given configurations $P_i = (x_i, y_i, z_i, \chi_i, \gamma_i, \kappa_i, \kappa_i')$ and $P_f = (x_f, y_f, z_f, \chi_f, \gamma_f, \kappa_f, \kappa_f')$.

The 7th order is the minimal order polynomial curve interpolating such configurations. The solution proposed for the above interpolating problem is given by a 7th order polynomial curve $P(u) = (x(u), y(u), z(u)), u \in [0, 1]$ defined below:

$$\begin{aligned} x(u) &= \alpha_0 + \alpha_1 u + \alpha_2 u^2 + \alpha_3 u^3 + \alpha_4 u^4 + \alpha_5 u^5 + \alpha_6 u^6 + \alpha_7 u^7 \\ y(u) &= \beta_0 + \beta_1 u + \beta_2 u^2 + \beta_3 u^3 + \beta_4 u^4 + \beta_5 u^5 + \beta_6 u^6 + \beta_7 u^7 \\ z(u) &= \gamma_0 + \gamma_1 u + \gamma_2 u^2 + \gamma_3 u^3 + \gamma_4 u^4 + \gamma_5 u^5 + \gamma_6 u^6 + \gamma_7 u^7 \end{aligned} \tag{2.163}$$

The polynomial coefficients are detailed by the following closed-form expressions, by solving a nonlinear equation system associated to the end point interpolation conditions:

$$\alpha_0 = x_i \tag{2.164}$$

$$\alpha_1 = \eta_1 \cos\chi_i \cos\gamma_i \tag{2.165}$$

$$\alpha_2 = \frac{1}{2}\eta_3 \cos\chi_i \cos\gamma_i - \frac{1}{2}\eta_1^2 \kappa_i \sin\chi_i \cos\gamma_i \tag{2.166}$$

$$\alpha_3 = \frac{1}{6}\eta_5 \cos\chi_i \cos\gamma_i - \frac{1}{6}\left(\eta_1^3\kappa_i' + 3\eta_1\eta_3\kappa_i\right)\sin\chi_i \cos\gamma_i \tag{2.167}$$

$$\begin{aligned}\alpha_4 = 35(x_f - x_i) &- \left(20\eta_1 + 5\eta_3 + \tfrac{2}{3}\eta_5\right)\cos\chi_i \cos\gamma_i + \\ &+ \left(5\eta_1^2\kappa_i + \tfrac{2}{3}\eta_1^3\kappa_i' + 2\eta_1\eta_3\kappa_i\right)\sin\chi_i \cos\gamma_i + \\ &- \left(15\eta_2 - \tfrac{5}{2}\eta_4 + \tfrac{1}{6}\eta_6\right)\cos\chi_f \cos\gamma_f + \\ &- \left(\tfrac{5}{2}\eta_2^2\kappa_f - \tfrac{1}{6}\eta_2^3\kappa_f' - \tfrac{1}{2}\eta_2\eta_4\kappa_f\right)\sin\chi_f \cos\gamma_f\end{aligned} \tag{2.168}$$

$$\begin{aligned}\alpha_5 = -84(x_f - x_i) &+ (45\eta_1 + 10\eta_3 + \eta_5)\cos\chi_i \cos\gamma_i + \\ &- \left(10\eta_1^2\kappa_i + \eta_1^3\kappa_i' + 3\eta_1\eta_3\kappa_i\right)\sin\chi_i \cos\gamma_i + \\ &+ \left(39\eta_2 - 7\eta_4 + \tfrac{1}{2}\eta_6\right)\cos\chi_f \cos\gamma_f + \\ &+ \left(7\eta_2^2\kappa_f - \tfrac{1}{2}\eta_2^3\kappa_f' - \tfrac{3}{2}\eta_2\eta_4\kappa_f\right)\sin\chi_f \cos\gamma_f\end{aligned} \tag{2.169}$$

$$\begin{aligned}\alpha_6 = 70(x_f - x_i) &- \left(36\eta_1 + \tfrac{15}{2}\eta_3 + \tfrac{2}{3}\eta_5\right)\cos\chi_i \cos\gamma_i \\ &+ \left(\tfrac{15}{2}\eta_1^2\kappa_i + \tfrac{2}{3}\eta_1^3\kappa_i' + 2\eta_1\eta_3\kappa_i\right)\sin\chi_i \cos\gamma_i + \\ &- \left(34\eta_2 - \tfrac{13}{2}\eta_4 + \tfrac{1}{2}\eta_6\right)\cos\chi_f \cos\gamma_f + \\ &- \left(\tfrac{13}{2}\eta_2^2\kappa_f - \tfrac{1}{2}\eta_2^3\kappa_f' - \tfrac{3}{2}\eta_2\eta_4\kappa_f\right)\sin\chi_f \cos\gamma_f\end{aligned} \tag{2.170}$$

$$\begin{aligned}\alpha_7 = -20(x_f - x_i) &+ \left(10\eta_1 + 2\eta_3 + \tfrac{1}{6}\eta_5\right)\cos\chi_i \cos\gamma_i + \\ &- \left(2\eta_1^2\kappa_i + \tfrac{1}{6}\eta_1^3\kappa_i' + \tfrac{1}{2}\eta_1\eta_3\kappa_i\right)\sin\chi_i \cos\gamma_i + \\ &+ \left(10\eta_2 - 2\eta_4 + \tfrac{1}{6}\eta_6\right)\cos\chi_f \cos\gamma_f + \\ &+ \left(2\eta_2^2\kappa_f - \tfrac{1}{6}\eta_2^3\kappa_f' - \tfrac{1}{2}\eta_2\eta_4\kappa_f\right)\sin\chi_f \cos\gamma_f\end{aligned} \tag{2.171}$$

The same kind of calculations are valid for $\beta_0, \ldots \beta_7$.

$$\beta_0 = y_i \tag{2.172}$$

$$\beta_1 = \eta_1 \sin\chi_i \cos\gamma_i \tag{2.173}$$

$$\beta_2 = \frac{1}{2}\eta_3 \sin\chi_i \cos\gamma_i - \frac{1}{2}\eta_1^2\kappa_i \cos\chi_i \cos\gamma_i \tag{2.174}$$

$$\alpha_3 = \frac{1}{6}\eta_5 \sin\chi_i \cos\gamma_i - \frac{1}{6}\left(\eta_1^3\kappa_i' + 3\eta_1\eta_3\kappa_i\right)\cos\chi_i \cos\gamma_i \tag{2.175}$$

$$\begin{aligned}\beta_4 = 35(y_f - y_i) &- \left(20\eta_1 + 5\eta_3 + \tfrac{2}{3}\eta_5\right)\sin\chi_i \cos\gamma_i + \\ &+ \left(5\eta_1^2\kappa_i + \tfrac{2}{3}\eta_1^3\kappa_i' + 2\eta_1\eta_3\kappa_i\right)\cos\chi_i \cos\gamma_i + \\ &- \left(15\eta_2 - \tfrac{5}{2}\eta_4 + \tfrac{1}{6}\eta_6\right)\sin\chi_f \cos\gamma_f + \\ &- \left(\tfrac{5}{2}\eta_2^2\kappa_f - \tfrac{1}{6}\eta_2^3\kappa_f' - \tfrac{1}{2}\eta_2\eta_4\kappa_f\right)\cos\chi_f \cos\gamma_f\end{aligned} \tag{2.176}$$

$$\begin{aligned}\beta_5 = &-84(y_f - y_i) + (45\eta_1 + 10\eta_3 + \eta_5)\sin\chi_i\cos\gamma_i + \\ &- \left(10\eta_1^2\kappa_i + \eta_1^3\kappa_i' + 3\eta_1\eta_3\kappa_i\right)\cos\chi_i\cos\gamma_i + \\ &+ \left(39\eta_2 - 7\eta_4 + \tfrac{1}{2}\eta_6\right)\sin\chi_f\cos\gamma_f + \\ &+ \left(7\eta_2^2\kappa_f - \tfrac{1}{2}\eta_2^3\kappa_f' - \tfrac{3}{2}\eta_2\eta_4\kappa_f\right)\cos\chi_f\cos\gamma_f\end{aligned} \tag{2.177}$$

$$\begin{aligned}\beta_6 = &\,70(y_f - y_i) - \left(36\eta_1 + \tfrac{15}{2}\eta_3 + \tfrac{2}{3}\eta_5\right)\sin\chi_i\cos\gamma_i + \\ &+ \left(\tfrac{15}{2}\eta_1^2\kappa_i + \tfrac{2}{3}\eta_1^3\kappa_i' + 2\eta_1\eta_3\kappa_i\right)\cos\chi_i\cos\gamma_i + \\ &- \left(34\eta_2 - \tfrac{13}{2}\eta_4 + \tfrac{1}{2}\eta_6\right)\sin\chi_f\cos\gamma_f + \\ &- \left(\tfrac{13}{2}\eta_2^2\kappa_f - \tfrac{1}{2}\eta_2^3\kappa_f' - \tfrac{3}{2}\eta_2\eta_4\kappa_f\right)\cos\chi_f\cos\gamma_f\end{aligned} \tag{2.178}$$

$$\begin{aligned}\beta_7 = &-20(y_f - y_i) + \left(10\eta_1 + 2\eta_3 + \tfrac{1}{6}\eta_5\right)\sin\chi_i\cos\gamma_i \\ &- \left(2\eta_1^2\kappa_i + \tfrac{1}{6}\eta_1^3\kappa_i' + \tfrac{1}{2}\eta_1\eta_3\kappa_i\right)\cos\chi_i\cos\gamma_i + \\ &+ \left(10\eta_2 - 2\eta_4 + \tfrac{1}{6}\eta_6\right)\sin\chi_f\cos\gamma_f + \\ &+ \left(2\eta_2^2\kappa_f - \tfrac{1}{6}\eta_2^3\kappa_f' - \tfrac{1}{2}\eta_2\eta_4\kappa_f\right)\cos\chi_f\cos\gamma_f\end{aligned} \tag{2.179}$$

and $\gamma_1 \ldots \gamma_7$.

$$\gamma_0 = z_i \tag{2.180}$$

$$\gamma_1 = \eta_1 \sin\gamma_i \tag{2.181}$$

$$\gamma_2 = \frac{1}{2}\eta_3\sin\gamma_i - \frac{1}{2}\eta_1^2\kappa_i\cos\gamma_i \tag{2.182}$$

$$\gamma_3 = \frac{1}{6}\eta_5\sin\gamma_i - \frac{1}{6}\left(\eta_1^3\kappa_i' + 3\eta_1\eta_3\kappa_i\right)\cos\gamma_i \tag{2.183}$$

$$\begin{aligned}\gamma_4 = &\,35(z_f - z_i) - \left(20\eta_1 + 5\eta_3 + \tfrac{2}{3}\eta_5\right)\sin\gamma_i + \\ &+ \left(5\eta_1^2\kappa_i + \tfrac{2}{3}\eta_1^3\kappa_i' + 2\eta_1\eta_3\kappa_i\right)\cos\gamma_i + \\ &- \left(15\eta_2 - \tfrac{5}{2}\eta_4 + \tfrac{1}{6}\eta_6\right)\sin\gamma_f + \\ &- \left(\tfrac{5}{2}\eta_2^2\kappa_f - \tfrac{1}{6}\eta_2^3\kappa_f' - \tfrac{1}{2}\eta_2\eta_4\kappa_f\right)\cos\gamma_f\end{aligned} \tag{2.184}$$

$$\begin{aligned}\gamma_5 = &-84(z_f - z_i) + (45\eta_1 + 10\eta_3 + \eta_5)\sin\gamma_i + \\ &- \left(10\eta_1^2\kappa_i + \eta_1^3\kappa_i' + 3\eta_1\eta_3\kappa_i\right)\cos\gamma_i + \\ &+ \left(39\eta_2 - 7\eta_4 + \tfrac{1}{2}\eta_6\right)\sin\gamma_f + \\ &+ \left(7\eta_2^2\kappa_f - \tfrac{1}{2}\eta_2^3\kappa_f' - \tfrac{3}{2}\eta_2\eta_4\kappa_f\right)\cos\gamma_f\end{aligned} \tag{2.185}$$

$$\begin{aligned}\gamma_6 = 70(z_f - z_i) - \left(36\eta_1 + \tfrac{15}{2}\eta_3 + \tfrac{2}{3}\eta_5\right)\sin\gamma_i + \\ + \left(\tfrac{15}{2}\eta_1^2\kappa_i + \tfrac{2}{3}\eta_1^3\kappa_i' + 2\eta_1\eta_3\kappa_i\right)\cos\gamma_i + \\ - \left(34\eta_2 - \tfrac{13}{2}\eta_4 + \tfrac{1}{2}\eta_6\right)\sin\gamma_f + \\ - \left(\tfrac{13}{2}\eta_2^2\kappa_f - \tfrac{1}{2}\eta_2^3\kappa_f' - \tfrac{3}{2}\eta_2\eta_4\kappa_f\right)\cos\gamma_f\end{aligned} \tag{2.186}$$

$$\begin{aligned}\gamma_7 = -20(z_f - z_i) + \left(10\eta_1 + 2\eta_3 + \tfrac{1}{6}\eta_5\right)\sin\gamma_i + \\ - \left(2\eta_1^2\kappa_i + \tfrac{1}{6}\eta_1^3\kappa_i' + \tfrac{1}{2}\eta_1\eta_3\kappa_i\right)\cos\gamma_i + \\ + \left(10\eta_2 - 2\eta_4 + \tfrac{1}{6}\eta_6\right)\sin\gamma_f + \left(2\eta_2^2\kappa_f - \tfrac{1}{6}\eta_2^3\kappa_f' - \tfrac{1}{2}\eta_2\eta_4\kappa_f\right)\cos\gamma_f\end{aligned} \tag{2.187}$$

The real vector $\eta_v = (\eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6)^T$ can be freely selected and influence the path shape without violating the end-point interpolating conditions. This solution represents a family of curves that depend on a symmetric parameterization induced by the chosen η vector. Specifically, parameters η_1, η_3, η_5 influence the curve at its beginning while the parameters η_2, η_4, η_6 affect the curve ending. Parameters η_1, η_2 can be interpreted as velocity parameters while parameters $\eta_3, \eta_4, \eta_5, \eta_6$ are twist parameters depending on curve accelerations and jerks at the end-points. Acting on the shaping parameters vector η, a wide variety of curves satisfying the boundary conditions can be obtained. This suggests choosing η to generate optimal curves. Different optimality criteria may be chosen depending on the mission of the aerial robot.

2.4 Nonholonomic Motion Planning

Nonholonomic motion planning relies on finding a trajectory in the state space between given initial and final configurations subject to nonholonomic constraints. The Lie algebraic method relies on a series of local planning around consecutive current states. Global trajectory results from joining local trajectories. At a current state, a direction of motion towards the goal state is established. Then, a rich enough space of controls is taken. As the driftfree system (2.2) is controllable, via some vector fields, the controls are able to generate the fields [41, 131]. The steering method for affine driftless systems exploits different properties of such a system namely differential flatness and nilpotence [37, 131, 139].

2.4.1 Differential Flatness

A flat system is a system such that there exists a finite set of variable y_i differentially independent which appear as differential function of the system variables (state variables and inputs) and a finite number of their derivatives, each system variable being

itself a function of the y_i and a finite number of their derivatives. The variables y_i are called the linearizing outputs of the system. In this case, path planning takes place in a space defined by parameters different in general from the configuration variables.

Definition 2.2. Differential flatness
A dynamic system

$$\begin{array}{c} \dot{x} = f(x,u) \\ y = h(x) \\ x \in \mathbb{R}^n \quad u, y \in \mathbb{R}^m \end{array} \tag{2.188}$$

is flat if and only if there exist variables $F \in \mathbb{R}^m$ called the flat outputs such that

$$\begin{array}{l} x = \Sigma_1\left(F, \dot{F}, \ldots, F^{(n-1)}\right) \\ y = \Sigma_2\left(F, \dot{F}, \ldots, F^{(n-1)}\right) \\ u = \Sigma_3\left(F, \dot{F}, \ldots, F^{(n-1)}\right) \end{array} \tag{2.189}$$

Σ_1, Σ_2, Σ_3 are three smooth mappings and $F^{(i)}$ is the ith derivative of F.

The parameterization of the control inputs u in function of the flat outputs F plays a key role in the trajectory planning problem: the nominal control inputs to be applied during a mission can be expressed in function of the desired trajectory, thus allowing to tune the profile of the trajectories for keeping the applied control inputs below the actuator limits. The advantage of flatness is that it allows to know a-priori if a predefined trajectory is feasible or not and it allows to modify the trajectory parameters in function of the constraints.

2.4.1.1 Helicopter Robot

The objective in [23] is to drive a quadrotor aerial robot as fast as possible from an initial position to a final position without violating kinematical and dynamical constraints. These constraints can be the actuator limits or the maximum allowable pitch and roll angles. A classical employed quad-rotor aerial robot model is:

$$\begin{array}{l} \ddot{x} = (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\, \frac{u_z}{m} \\ \ddot{y} = (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\, \frac{u_z}{m} \\ \ddot{z} = (\cos\phi \cos\theta)\, \frac{u_z}{m} - g \end{array} \tag{2.190}$$

$$\ddot{\theta} = \frac{u_\theta}{J_1} \quad \ddot{\phi} = \frac{u_\phi}{J_2} \quad \ddot{\psi} = \frac{u_\psi}{J_3} \tag{2.191}$$

where ϕ, θ, ψ are the roll, pitch and yaw Euler angles. A simplified model is employed for trajectory planning, assuming hovering conditions $u_z \approx mg$, no yawing and small roll and pitch angles

$$\begin{aligned} \ddot{x} &= \theta g \\ \ddot{y} &= -\phi g \\ \ddot{z} &= \tfrac{u_z}{m} - g \end{aligned} \tag{2.192}$$

$$\ddot{\theta} = \frac{u_\theta}{J_1} \quad \ddot{\phi} = \frac{u_\phi}{J_2} \quad \ddot{\psi} = \frac{u_\psi}{J_3} \tag{2.193}$$

The linear system (2.192) is flat with $F_1 = z, F_2 = x, F_3 = y, F_4 = \psi$ and $\theta = \frac{\ddot{F}_2}{g}, \phi = -\frac{\ddot{F}_3}{g}$. Let $F_1^*, F_2^*, F_3^*, F_4^*$ be the reference trajectories for the flat outputs F_1, F_2, F_3, F_4, the nominal control inputs to be applied along these reference trajectories are:

$$\begin{aligned} u_z^* &= m\left(\ddot{F}_1^* + g\right) \quad u_\theta^* = J_1 \tfrac{F_2^{4*}}{g} \\ u_\phi^* &= -J_2 \tfrac{F_3^{4*}}{g} \quad u_\psi^* = J_3 \ddot{F}_4^* \end{aligned} \tag{2.194}$$

The relative degree are $r_z = r_\psi = 2$ and $r_\theta = r_\phi = 4$.

When Bezier polynomials are used to design the reference trajectories F_i^*, a polynomial of degree 5 (as 5 parameters must be determined) is used for F_1, F_4 and a polynomial of degree 9 for F_2, F_3 is used (as 9 parameters must be determined). The parameters polynomials may be determined knowing the initial and final configurations. The trajectory planning to drive the aerial robot as fast as possible consists in tuning the profile of the trajectory by tuning the final time of the mission.

2.4.1.2 Airplane Robot

The kinematic model of the airplane like aerial robot exhibits also the property known as differential flatness particularly relevant in planning problems. A nonlinear system

$$\dot{\mathbf{X}} = f(\mathbf{X}) + g(\mathbf{X})\mathbf{U} \tag{2.195}$$

is differentially flat if there exists a set of outputs $\mathbf{Y}$ called flat outputs such that the state $\mathbf{X}$ and the control inputs $\mathbf{U}$ can be expressed algebraically as a function of $\mathbf{Y}$ and its time derivative up to a certain order:

$$\begin{aligned} \mathbf{X} &= \mathbf{X}\left(\mathbf{Y}, \dot{\mathbf{Y}}, \ddot{\mathbf{Y}}, \ldots, \mathbf{Y}^{(r)}\right) \\ \mathbf{U} &= \mathbf{U}\left(\mathbf{Y}, \dot{\mathbf{Y}}, \ddot{\mathbf{Y}}, \ldots, \mathbf{Y}^{(r)}\right) \end{aligned} \tag{2.196}$$

As a consequence, once an output is assigned for $\mathbf{Y}$, the associated trajectory of the state $\mathbf{X}$ and history of control inputs are uniquely determined. The kinematic model is recalled, using the East-North-Up frame, as:

$$\begin{aligned} \dot{x} &= V \sin\chi \cos\gamma + W_x \\ \dot{y} &= V \cos\chi \cos\gamma + W_y \\ \dot{z} &= V \sin\gamma + W_z \end{aligned} \tag{2.197}$$

and the dynamic model of an airplane like aerial robot is given by the following equations:

$$\begin{array}{c}\dot{V} = \frac{1}{m}\left(T\cos\alpha - D + mg\sin\gamma\right) + \\ -\left(\dot{W}_x\cos\gamma\sin\chi + \dot{W}_y\cos\gamma\cos\chi + \dot{W}_z\sin\gamma\right)\end{array} \tag{2.198}$$

with

$$\dot{\chi} = \frac{1}{mV\cos\gamma}\left(L + T\sin\alpha\right)\sin\sigma - \frac{1}{V\cos\gamma}\left(\frac{\dot{W}_x\cos\chi - \dot{W}_z\sin\chi}{V\cos\gamma}\right) \tag{2.199}$$

and

$$\begin{array}{c}\dot{\gamma} = \frac{1}{mV}\left(L\cos\sigma + T\cos\sigma\sin\alpha + mg\cos\gamma\right) + \\ -\frac{1}{V}\left(\dot{W}_x\sin\gamma\sin\chi + \dot{W}_y\sin\gamma\cos\chi - \dot{W}_z\cos\gamma\right)\end{array} \tag{2.200}$$

If the control inputs are T, σ, α or equivalently T, σ, L then the aerial robot motion is differentially flat. By determining a suitable trajectory in Cartesian coordinates x, y, and z, the required controls can be calculated.

$$V = \sqrt{(\dot{x} - W_x)^2 + (\dot{y} - W_y)^2 + (\dot{z} - W_z)^2} \tag{2.201}$$

$$\gamma = \arcsin\frac{\dot{z} - W_z}{V} \tag{2.202}$$

$$\chi = \arctan\frac{\dot{x} - W_x}{\dot{y} - W_y} \tag{2.203}$$

Using the following notations,

$$\begin{array}{c}\eta_1 = m\dot{V} - mg\sin\gamma + m\left(\dot{W}_x\cos\gamma\sin\chi + \dot{W}_y\cos\gamma\cos\chi + \dot{W}_z\sin\gamma\right) \\ \eta_2 = mV\dot{\chi}\cos\gamma + m\left(\dot{W}_x\cos\chi + m\dot{W}_y\sin\chi\right) \\ \eta_3 = mV\dot{\gamma} - mg\cos\gamma + m\left(\dot{W}_x\sin\gamma\sin\chi + \dot{W}_y\sin\gamma\cos\chi - \dot{W}_z\cos\gamma\right)\end{array} \tag{2.204}$$

The bank angle is given by:

$$\sigma = \arctan\frac{\eta_2}{\eta_3} \tag{2.205}$$

While the thrust is given by:

$$T = \sqrt{\left(\eta_1 - L\frac{C_D}{C_L}\right)^2 + (\eta_2 - L\sin\sigma)^2 + (\eta_3 - L\cos\sigma)^2} \tag{2.206}$$

The lift is given by resolving the following equation:

$$L^2 - (\eta_2 - T\sin\alpha\sin\sigma)^2 + (\eta_3 - T\sin\alpha\cos\sigma)^2 = 0 \tag{2.207}$$

where the lift force L is normal to the velocity vector of the aerial robot with respect to the air and is contained in the plane of symmetry of the aerial robot. The drag force D is parallel and in the opposite direction of the velocity vector.

$$D = L\frac{C_D}{C_L} = L\left(\frac{C_{D_0}}{C_l} + KC_L\right) \tag{2.208}$$

The constraints are first expressed in terms of thrust and velocities and then transformed into limitations on flight path and heading angles. As the aerial robot admits a set of flat outputs, these may be exploited to solve planning problems efficiently. In fact, some interpolation schemes can be used to solve the path of $\mathbf{Y}$ in such a way to satisfy the appropriate boundary conditions. The evolution of the other configuration variables, together with the associated control inputs, can then be computed algebraically from $\mathbf{Y}$. The resulting configuration space path will automatically satisfy the nonholonomic constraints [9].

2.4.2 Nilpotence

The control system

$$\begin{gathered}\dot{x} = V\cos\chi\cos\gamma \quad \dot{y} = V\sin\chi\cos\gamma \quad \dot{z} = V\sin\gamma \\ V = u_1 \quad \dot{\chi} = u_2 \quad \dot{\gamma} = u_3\end{gathered} \tag{2.209}$$

is nilpotent as the Lie brackets of the control vector fields vanish from level 3. For such system, it is possible to compute a basis of the control Lie algebra from a Philip Hall family. Because this system is nilpotent, each exponential of Lie brackets can be developed exactly as finite combination of the control vector fields and thus find piecewise constant controls steering the system exactly to the goal.

2.4.2.1 Philip Hall Basis

This system as given by

$$\dot{X} = G(X)U \tag{2.210}$$

can be steered along these Lie brackets $\{g_1, g_2, g_3, g_4, g_5\}$. Where

$$g_1 = \begin{pmatrix}\cos\gamma\cos\chi \\ \cos\gamma\sin\chi \\ \sin\gamma \\ 0 \\ 0\end{pmatrix} \quad g_2 = \begin{pmatrix}0 \\ 0 \\ 0 \\ 1 \\ 0\end{pmatrix} \quad g_3 = \begin{pmatrix}0 \\ 0 \\ 0 \\ 0 \\ 1\end{pmatrix} \tag{2.211}$$

The vectors $g_4 = [g_1, g_2], g_5 = [g_1, g_3]$ represent new motion directions that can be followed approximately. Locally generating motion in these directions is slower than following the vector fields g_1, g_2, g_3. The Philip Hall basis gives a way to choose the smallest number of Lie products that must be considered to generate a basis for this distribution [41]. The level of difficulty in steering controllable systems is proportional to the order of Lie brackets. When the dimension of the Lie algebra is 5 according to Chow's theorem and Lie algebra rank condition, the system is proved to be completely nonholonomic and controllable. The motion planning problem is concerned with finding a control that steers the system from an initial configuration X_i to the final configuration $X_f \in \mathbb{R}^5$ along a certain trajectory. The following extension Ξ_e of Eq. (2.210) is proposed by adding higher order Lie bracket motions

$$\dot{X} = g_1 u_1 + g_2 u_2 + g_3 u_3 + g_4 u_4 + g_5 u_5 \tag{2.212}$$

where u_4, u_5 are fictitious inputs that may not correspond to the actual system inputs. It has been proved above that the system is nilpotent, as all the Lie brackets vanish from level 3. Once the extended system has been selected, the solution proposed here of the motion planning problem involves two steps:

1. STEP 1: Find a control $U = (u_1, u_2, u_3, u_4, u_5)^T$ that steers Ξ_e from a point X_i to a point X_f, defining a curve $\varpi(t) : [0, T] \to \mathbb{R}^5$, then the tangent vector $\dot{\varpi}(t)$ is expressed as a linear combination of

$$g_1(\varpi(t)), g_2(\varpi(t)), g_3(\varpi(t)), g_4(\varpi(t)), g_5(\varpi(t))$$

   the corresponding coefficients being $u_i(t), i = 1 \ldots 5$
2. STEP 2: Use U to compute a control u that steers Ξ_e from the point X_i to a point X_f by computing the Philip Hall coordinates of the extended system and by finding a control u from the Philip Hall coordinates.

As $\{g_1, g_2, g_3, g_4, g_5\}$ are vector fields defined in a neighborhood N of a point X such that at each point of N, the set $\{g_1, g_2, g_3, g_4, g_5\}$ constitutes a basis of the tangent space, then there is a smaller neighborhood of X on which the maps

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) \to e^{\alpha_1 g_1 + \alpha_2 g_2 + \alpha_3 g_3 + \alpha_4 g_4 + \alpha_5 g_5} X$$

and

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) \to e^{\alpha_5 g_5} e^{\alpha_4 g_4} e^{\alpha_3 g_3} e^{\alpha_2 g_2} e^{\alpha_1 g_1} X$$

are two coordinate systems, called the first and second normal coordinate system associated to $\{g_1, g_2, g_3, g_4, g_5\}$. The Campbell Baker Hausdorff Dynkin (CBHD) formula states precisely the difference between the two systems for a sufficiently small τ, the following relation is verified:

$$e^{\tau f} \cdot e^{\tau g} = e^{\tau f + \tau g - 0.5[f,g] + \tau^2 \epsilon(t)} \tag{2.213}$$

where $\epsilon(\tau) \to 0$ when $\tau \to 0$. Actually, the whole formula gives an explicit form for the $\epsilon(t)$ function. More precisely, $\epsilon(t)$ yields a formal series whose coefficients c_k of τ^k are combinations of brackets of degree k, i.e.

$$\tau^2 \epsilon(t) = \sum_{k=3}^{\infty} \tau^k c_k \tag{2.214}$$

Roughly speaking, the Campbell Baker Hausdorff Dynkin formula tells how a small time controllable nonholonomic system can reach any point in the neighborhood of a starting point [95, 96]. The formula is the hard core of the local controllability concepts. It yields a method for explicitly computing admissible paths in a neighborhood of a point.

Philip Hall Coordinates

The Philip Hall basis of the controllability Lie Algebra generated by $\{g_1, g_2, g_3\}$ is defined as follows $\{g_1, g_2, g_3, g_4, g_5\}$. All flows can be represented in the form

$$\begin{aligned} S(t) &= e^{h_5(t)g_5} e^{h_4(t)g_4} e^{h_3(t)g_3} e^{h_2(t)g_2} e^{h_1(t)g_1} \\ \overline{S}(t) &= e^{\overline{h}_1(t)g_1} e^{\overline{h}_2(t)g_2} e^{\overline{h}_3(t)g_3} e^{\overline{h}_4(t)g_4} e^{\overline{h}_5(t)g_5} \end{aligned} \tag{2.215}$$

The map $S \to (h_1, \ldots, h_5)$ and $\overline{S} \to \left(\overline{h}_1, \ldots, \overline{h}_5\right)$ establish global diffeomorphism between Lie groups and $\mathbb{R}^5$ [128]. h_i are the backward Philip Hall coordinates of S and $\overline{h}_i$ are the forward Philip Hall coordinates of $\overline{S}$. In addition, $S(t)$ satisfies the differential equation

$$\dot{S}(t) = S(t)\,(g_1 u_1 + g_2 u_2 + g_3 u_3 + g_4 u_4 + g_5 u_5) \qquad S(0) = I \tag{2.216}$$

Now, the problem of finding the initial control U is tackled out.

Remark 2.1. The symbol $\otimes$ is used for concatenation, for example $A \otimes B$ means A followed by B.

Finding U

Now, the following initial value problem is considered:

$$\dot{S}^*(t) = S^*(t)\,(g_1 u_1 + g_2 u_2 + g_3 u_3) \qquad S^*(0) = I \tag{2.217}$$

The backward Philip Hall coordinates h_i being known, the control $u(t)$ must be found that produces $S^*(t)$ having these coordinates. First, the forward coordinates can be obtained from Eq. (2.217). Then, the question is turned into finding a control for

$$S^*(t) = e^{\overline{h}_1(t)g_1} e^{\overline{h}_2(t)g_2} e^{\overline{h}_3(t)g_3} e^{\overline{h}_4(t)g_4} e^{\overline{h}_5(t)g_5} \tag{2.218}$$

Each exponential factor must be solved separately, then the results must be concatenated. Since the first flow $e^{\overline{h}_1(t)g_1}$ is just along g_1, so the control input $u_1 = \overline{h}_1$ for the first control sequence. Similarly, for the second and third control sequences, the result input $u_2 = \overline{h}_2$ will result in the flow $e^{\overline{h}_2(t)g_2}$ and $u_3 = \overline{h}_3$ will result in the flow $e^{\overline{h}_3(t)g_3}$. So the control sequences $\overline{h}_1(t)g_1 \otimes \overline{h}_2(t)g_2 \otimes \overline{h}_3(t)g_3$ generate the flow $e^{\overline{h}_1(t)g_1} e^{\overline{h}_2(t)g_2} e^{\overline{h}_3(t)g_3}$. Note that the flow along the Lie bracket direction $e^{\overline{h}_4(t)g_4} = e^{\overline{h}_4(t)[f,g_1]}$ is needed. Using the Campbell Baker Hausdorff formula, the following control sequence (assuming $\overline{h}_4 > 0$ and $\overline{h}_5 > 0$),

$$\sqrt{\overline{h}_4(t)g_1} \otimes \sqrt{\overline{h}_4(t)g_2} \otimes \sqrt{\overline{h}_4(t)g_3} \otimes \left(-\sqrt{\overline{h}_4(t)g_3}\right) \otimes \left(-\sqrt{\overline{h}_4(t)g_2}\right) \otimes \left(-\sqrt{\overline{h}_4(t)g_1}\right),$$

idem for $e^{\overline{h}_5(t)g_5} = e^{\overline{h}_5(t)[f,g_2]}$ with

$$\sqrt{\overline{h}_5(t)g_1} \otimes \sqrt{\overline{h}_5(t)g_2} \otimes \sqrt{\overline{h}_5(t)g_3} \otimes \left(-\sqrt{\overline{h}_5(t)g_3}\right) \otimes \left(-\sqrt{\overline{h}_5(t)g_2}\right) \otimes \left(-\sqrt{\overline{h}_5(t)g_1}\right)$$

give rise to $e^{\overline{h}_4(t)g_4} e^{\overline{h}_5(t)g_5}$
so far a control U made of $1 * 1 * 1 * 6 * 6 = 36$ pieces have been obtained so $S(t)$ can be realized by 36 moves.

2.4.3 Constrained Motion Planning

This section adddresses the constrained motion planning of aerial robots represented by kinematic driftless control system with limited inputs and controls [63]. The problem consists in defining a control function driving the system output to a desirable point at a given time instant whereas state and control variables remain over the control horizon within prescribed bounds.

$$\dot{q} = G(q) = \sum_{i=1}^{m} g_i(q)u_i \tag{2.219}$$

and

$$Y = k(q) \tag{2.220}$$

where $q = (x, y, z, \chi, \gamma)^T$, $u = (V, \dot{\chi}, \dot{\gamma})^T$

$$G(q) = \begin{pmatrix} \cos\gamma\cos\chi & 0 & 0 \\ \cos\gamma\sin\chi & 0 & 0 \\ \sin\gamma & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [g_1(q) \quad g_2(q) \quad g_3(q)] \tag{2.221}$$

Let $q(t) = \varphi_{q_0,t}(u(.))$ denote the state trajectory of the basic system (2.219), initialized at q_0 and driven by a control $u(t)$. Then the end point map $K_{q_0,t} : \mathbb{U} \to \mathbb{R}^r$ of this system is defined as:

$$K_{q_0,t}(u(.)) = k(q(T)) = k\left(\varphi_{q_0,t}(u(.))\right) \tag{2.222}$$

Problem 2.4. Constrained motion planning Given an initial state q_0 and a desirable point $Y_d \in \mathbb{R}^r$, find a control function $u(.)$ such that $K_{q_0,T}(u(.)) = Y_d$ while the instantaneous values of state and control variables are bounded

$$q_i^{min} \le q_i(t) \le q_i^{max} \quad u_j^{min} \le u_j(t) \le u_j^{max} \quad \forall t \in [0, T] \tag{2.223}$$

This basic system (2.219) has strongly linear growth in controls. There exist two methods of inclusion of constraints into the motion planning problem:

1. The first recommends to multiply the control vector fields in the driftless control system by a smooth function vanishing in the impenetrable region of the state space.
2. The second resorts to exterior penalty function.

State and input bounds are incorporated into the system through extending the system by extra state variables driven by the plus function dependent on the violation of constraints. To include the constraints (2.223) into the basic system the **plus function**

$$\xi_+ = max\,(\xi, 0) \tag{2.224}$$

is included so that the constraints will be satisfied when the functions $(u_j(t) - u_j^{min})_+$, $\left(u_j^{max} - u_j(t)\right)_+$, $(q_j(t) - q_j^{min})_+$, $\left(q_j^{max} - q_j(t)\right)_+$ vanish $\forall t \in [0, T]$. Futhermore, in order to secure the smoothness of the extended system, the plus function can be approximated by a smooth function:

$$\xi_+ \approx p(\xi, \alpha) = \xi + \frac{1}{\alpha}\ln\left(1 + \exp(-\alpha\xi)\right) \tag{2.225}$$

parameterized by $\alpha > 0$. It follows that the function approaches ξ_+ when $\alpha \to +\infty$ and that the derivative $\left|\frac{dp(\xi,\alpha)}{d\xi}\right| \le 1$. In this way, the original motion planning and the problem of satisfaction of constraints are made equivalent to an unconstrained motion planning formulated in the extended system.

The state and control constraints are handled by extending the control system with a pair of state equations driven by the violation of constraints and adding regularizing

perturbation. The motion planning problem is solved by means of the continuation methods; The continuation method formulates the motion planning problem in terms of the endpoint map of the control system representation, whose derivative with respect to the control function is referred to as the system Jacobian. For the regularized system a Jacobian motion algorithm, called **imbalanced** is designed. However, an obstacle appears against using Jacobian motion algorithm in the extended system, resulting from the singularity of the Jacobian in the region, where constraints are satisfied. Suppose

$$\begin{gathered}\dot{q} = G(q)u \\ \dot{q}_{n+1} = \sum_{i\in N_1}\left(p(q_i - q_i^{min}, \alpha)\right) + \left(p(-q_i + q_i^{max}, \alpha)\right) \\ \dot{q}_{n+2} = \sum_{j\in N_2}\left(p(u_j - u_j^{min}, \alpha)\right) + \left(p(-u_j + u_j^{max}, \alpha)\right) \\ Z = (Y, q_{n+1}, q_{n+2})\end{gathered} \tag{2.226}$$

In order to cope with this singularity problem, the extended system is subject to a regularizing perturbation. The right-hand side of Eq. (2.226) is perturbed by a pair of functions $r_1(q)$ and $r_2(u)$ that depend respectively on q_i, $i \in N_1$ and u_j, $j \in N_2$, giving rise to a regularized system

$$\begin{gathered}\dot{q} = G(q)u \\ \dot{q}_{n+1} = \sum_{i\in N_1}\left(p(q_i - q_i^{min}, \alpha)\right) + \left(p(-q_i + q_i^{max}, \alpha)\right) + r_1(q) \\ \dot{q}_{n+2} = \sum_{j\in N_2}\left(p(u_j - u_j^{min}, \alpha)\right) + \left(p(-u_j + u_j^{max}, \alpha)\right) + r_2(u) \\ Z = (Y, q_{n+1}, q_{n+2})\end{gathered} \tag{2.227}$$

The perturbations should be chosen in such a way that a regularized system has still strongly linear growth in the controls. The motion planning problem in the regularized system is then solved by the imbalanced Jacobian system. The motion planning problem amounts to determining a control function $u(.)$ that drives $Z(T)$ to

$$Z_d(u(.)) = \left(Y_d, \int_0^T r_1(q)dt, \int_0^T r_2(u)dt\right) \tag{2.228}$$

A curve u_θ is chosen and an error is defined as:

$$e(\theta) = R_{x_0,T}(u_\theta) - Z_d(\theta) \tag{2.229}$$

containing the desirable output $Z_d(\theta) = Z_d(u_{\theta(.)})$ of the regularized system. The basic idea of the imbalanced Jacobian algorithm is to use the fact that the error (2.229) coincides with the error in the extended system $e(\theta) = R_{x_0,T}(u_\theta) - Z_d(\theta)$ where $Z_d = (Y_d, 0, 0)$ and to compute u_θ from the equation

$$J_{x_0,T}\left(u_\theta(.)\right)\frac{du_\theta}{d\theta} = -\gamma e(\theta) \tag{2.230}$$

and if the derivative of $Z_d(\theta)$ on the right hand side of Eq. (2.230) has been absent. Let $J'_{x_0,T}$ be the right inverse of the Jacobian, then an application of this inverse to (2.230) leads to the dynamic system:

$$\frac{du_\theta}{d\theta} = -\gamma J'_{x_0,T}\,(u_\theta(.))\,e(\theta) \tag{2.231}$$

Assume that u_θ is a solution of relation (2.231). After its substitution in (2.230), the following relation is verified

$$\frac{de_\theta}{d\theta} = -\gamma e(\theta) + \pi(\theta) \tag{2.232}$$

where $\pi(\theta) = -\frac{dZ_d(\theta)}{d\theta}$ can be regarded as a perturbation.

The system (2.231) defines the imbalanced Jacobian inverse kinematics algorithm if the perturbation $\pi(\theta)$ is such that the error (2.232)$\rightarrow$ 0 along with θ. If this is the case, solution to the constrained inverse kinematic problem can be computed as the limit to infinity of the trajectory of (2.231). The adjective **imbalanced** reflects to the presence of the perturbation on the right hand side of (2.232).

2.4.4 Motion Planning for Highly Congested Spaces

The theory deals with ϵ-approximations of non admissible paths by admissible ones in a certain optimal sense. The need of such an approximation arises in high congested configuration spaces [19]. A reasonable general strategy to obtain an admissible path that avoids obstacles and complies with constraints is

- not taking into account admissibility issues, elaborate a path that copes with the configuration constraints
- use the first path as an Ariadne thread to compute an approximating admissible path

Actually, for highly congested configuration spaces, the admissible path has to stay $\epsilon-$close to the Ariadne thread. Such a situation may arise either because a high density of obstacles in the physical space or due to constraints imposed on the aerial robot by the task or mission to be achieved. In practice ϵ may be taken as large as possible depending on the mission requirements [19]. Kinematic models of aerial robots are given as a vector distribution Δ over a n-dimensional manifold M. The rank of the distribution is p and the corank is $k = n - p$. Motion planning problems are local problems in an open neighborhood of a given finite path:

$$\dot{X} = \sum_{i=1}^{p} G_i(X)U_i \tag{2.233}$$

where for a typical aerial robot:

$$U = \begin{bmatrix} V \\ \dot{\chi} \\ \dot{\gamma} \end{bmatrix} \tag{2.234}$$

and

$$X = \begin{bmatrix} x \\ y \\ z \\ \chi \\ \gamma \end{bmatrix} \tag{2.235}$$

and

$$G = \begin{bmatrix} \cos\gamma\cos\chi & 0 & 0 \\ \cos\gamma\sin\chi & 0 & 0 \\ \sin\gamma & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.236}$$

The G_i are smooth C^∞ vector fields that span the distribution Δ. The standard controllability assumption is verified as the Lie algebra generated by the G_i spans the whole tangent space at each point of M, as long as

$$V\cos\gamma \neq 0 \tag{2.237}$$

Consequently, the distribution Δ is completely non integrable and any smooth path: $\Gamma : [0, T] \rightarrow M$ can be uniformly approximated by an admissible path: $\vartheta : [0, \tau] \rightarrow M$, i.e. a Lipschitz path, which is almost everywhere tangent to Δ. Thus it is possible to approximate uniformly non admissible paths by admissible ones. The purpose of this section is to present a general constructive theorem that solves this problem in a certain constructive way [141]. The objective function to be minimized is given in the following form:

$$J(U) = \int_0^\tau \sqrt{\sum_{i=1}^{p} (U_i^2)}dt \tag{2.238}$$

This choice is motivated by several reasons:

- The optimal curves do not depend on their parametrization
- The minimization of such a cost produces a metric space. The associated distance is called the subriemannian distance or the Carnot Theodory distance
- To minimize such a cost is equivalent to minimize the following function

$$J(U) = \int_0^\tau \sqrt{\sum_{i=1}^{p} (U_i^2)}dt \tag{2.239}$$

The distance between two points is defined as the minimum length of admissible curves connecting these two points. The length of the admissible curve corresponding to the control $u : [0, \theta] \to M$ is simply $J(u)$.

Another way to interpret the problem is as follows: the dynamics is specified by the distribution Δ (i.e. not by the vector fields F_i but their span only). The cost is then determined by an Euclidean metric g over Δ, specified by the fact that the G_i form an orthogonal frame field for the metric. In reasonable dimensions and co-dimensions, the optimal trajectories are extremely robust, and in particular, do not depend at all (modulo certain matrix transformation) on the choice of the metric but depend on the distribution Δ only. They depend only at the nilpotent approximation along Γ. For many low values of the rank p and corank k, these nilpotent approximations are universal in a certain sense that depend only on certain integer numbers, namely the dimensions of the successive bracket space generated by Δ and no functional or real parameter appears in the problem reduced to its nilpotent approximation. As a consequence, the asymptotic optimal syntheses (i.e. the phase portraits of the admissible trajectories that approximate up to a small ϵ) are also universal. Given a motion planning problem, specified by a non-admissible curve Γ, and a subriemanian structure, two different distinct concepts are considered

- The matrix complexity $MC(\epsilon)$ that measure asymptotically the length of the last $\epsilon-$approximation admissible trajectory.
- the interpolation entropy $E(\epsilon)$ that measures the length of the best admissible curves that interpolate Γ with pieces of length ϵ.

The set of motion planning problems on $\mathbb{R}^n$ is the set of couples Γ, Σ embedded with the C^∞ topology of uniform convergence over compact sets and generic problems (or problems in general position) form an open dense set in this topology. For instance, it means that the curve Γ is always transversal to Δ (except it may be at isolated points, in the case $k = 1$ only). More details on this implementation can be found in [19].

2.5 Obstacle/Collision Avoidance

The aim of this section is to formulate the problem for avoiding obstacles during a mission, while fulfilling its objective. Then some algorithms for the obstacle avoidance are presented.

2.5.1 Problem Formulation

Some assumptions are introduced in order to reduce conflict avoidance to the purely geometrical but still quite difficult problem of generating a collision free path. The canonical motion planning problem can be expressed as follows [50]:

Problem 2.5. Canonical Motion Planning: Given an initial and a final configuration of the aerial robot A, find if there exists a path, i.e. a continuous sequence of postures, that drives the aerial robot between both configurations while avoiding collisions between A and the obstacles $O_1 \ldots O_N$; report a failure if such a path does not exist.

Clearly, some of the hypotheses of this canonical problem may not be satisfied in applications. For example, the free flying hypothesis does not hold as two non-holonomic constraints exist in 3D and the aerial robot cannot move along arbitratry paths [18]. The assumption that the aerial robot is the only object in motion in the workspace rules out the relevant case of moving obstacles and multi-robot systems. Advance knowledge of obstacle geometry and placement is another strong assumption. In unstructured environments, the aerial robot is typically in charge of detecting obstacles by means of its sensors, and the planning problem must therefore be solved on-line during the motion [6].

Two variants of the canonical planning problem can also be presented. First the feasibility problem is formalized, then the optimality problem is introduced. Let X be a bounded connected open subset of $\mathbb{R}^n$ where $n \in \mathbb{N}, n = 2, 3$. Let X_{obs}, X_{goal} called the obstacle region and the goal region, respectively, be open subsets of X. The obstacle-free space X_{free} is X/X_{obs}. The initial state x_{init} is an element of X_{free}. A path in X_{free} is a collision-free path. A collision-free path that starts at x_{init} and ends in the goal region is said to be a feasible path: a collision-free path $\sigma : [0, s] \rightarrow X_{free}$ is feasible if and only if $\sigma(0) = x_{init}$ and $\sigma(s) \in X_{goal}$. The feasibility problem of path planning is to find a feasible path, if one exists and report failure otherwise [75].

Problem 2.6. Feasible Motion Planning: Given a bounded connected open set $S \subset \mathbb{R}^n$, an obstacle space $X_{obs} \subset X$, an initial state $x_{init} \in X_{free}$ and a goal region $X_{goal} \subset X_{free}$, find a path $\sigma : [0, s] \rightarrow X_{free}$ such that $\sigma(0) = x_{init}$ and $\sigma(s) \in X_{goal}$, if one exists. If no such path exists, then report failure.

Let $c : X_{free} \rightarrow \mathbb{R}^+$ be the cost function, assigning a non-negative cost to all nontrivial collision-free paths. The optimality problem asks for finding a feasible path with minimal cost.

Problem 2.7. Optimal Motion Planning: Given a bounded connected open set X, an obstacle space X_{obs}, an initial state x_{init}, find a path $\sigma^* : [0, s] \rightarrow cl\left(X_{free}\right)$ such that $\sigma^*(0) = x_{init}, \sigma^*(s) \in X_{goal}, c(\sigma^*) = min_{\sigma \in cl(X_{free})}$. If no such path exists, then report failure.

There are a variety of problem types defined in the literature. A problem is considered static if knowledge of the environment is perfect and dynamic if knowledge of the environment is imperfect or changes as the task takes place. When the obstacles are fixed in space, the problem is called time-invariant and when they are allowed to move, the problem is called time-variant. When the vehicle's equations of motion act as constraints on the path, this is Kinodynamic planning [86]. Kinodynamic motion planning problems can be formulated to handle systems with differential

constraints such as the aerial robot. The free flying robot hypothesis does not hold in nonholonomic mechanical systems, which cannot move along arbitrary paths in the workspace [98].

Problem 2.8. Kinodynamic Motion Planning: Given the domain X, obstacle region X_{obs}, goal region X_{goal} and a smooth function f that describes the system dynamics, find a control $u \in \mathcal{U}$ with domain $[0, T]$ for some $T \in \mathbb{R}^+$ such that the corresponding unique trajectory $x \in \mathcal{X}$, with $\dot{x} = f(x(t), u(t))$ for all $t \in [0, T]$ avoids the obstacles, i.e. $x(t) \in X_{free}$ for all $t \in [0, T]$ and reaches the goal region i.e $x(t) \in X_{goal}$.

Optimal kinodynamic motion planning problem can also be formulated to solve the kinodynamic motion planning while optimizing a given objective function.

Problem 2.9. Optimal Kinodynamic Motion Planning: Given the domain X, obstacle region X_{obs}, goal region X_{goal} and a smooth function f that describes the system dynamics, find a control $u \in \mathcal{U}$ with domain $[0, T]$ for some $T \in \mathbb{R}^+$ such that the corresponding unique trajectory $x \in \mathcal{X}$, with $\dot{x} = f(x(t), u(t))$ for all $t \in [0, T]$ avoids the obstacles, i.e. $x(t) \in X_{free}$ for all $t \in [0, T]$, reaches the goal region i.e $x(t) \in X_{goal}$ and minimizes the objective function $J(x) = \int_0^T g(x)dt$.

It is possible to further categorize problems based on the assumed aerial robot shape, environment type and behavior [17, 33, 38, 78, 92, 134]. The common problem types used in literature are described below:

1. **Point Robot**: In this problem, the aerial robot is modeled as a point within the world space. Thus the configuration space is the same as the world space. Often, an aerial robot is modeled by fitting it inside a bounding ball (in 2D Euclidean space this is a circle and in 3D Euclidean space a sphere), and the configuration space is simply the world space with the obstacles expanded by the radius of the aerial robot's bounding ball. Thus the ball-shaped aerial robot problem is the same as the point aerial robot problem. This is a conservative approximation to and simplification of the mover's problem. The minimum length path is the optimal path.
2. **Point robot with geometric constraints**: Planning with geometric constraints is the problem of moving an object through an obstacle field to a goal state. The aerial robot is usually modeled as a rigid body, thus the configuration space has a larger dimension than the world space. A classical problem of this case is the piano mover's problem. For this kind of problem, it is usually assumed that the object has no dynamic constraints. Mover's problems measure complexity of the aerial robot in addition to that of the obstacle field.
3. **Point Vehicle with differential constraints**: In problems with differential constraints, time and states have to satisfy the equations of motion of the aerial robot. Typically the states are constrained by limits on velocity and acceleration, and sometimes also on higher-order derivatives of position, and propulsion related to

flight envelope. For many aerial vehicle, this more realistic model is needed for the stability of the aerial robot.

4. **General vehicle with differential constraints**: The differential constraints typically arise in two forms: one is on kinematics, and this kind of problem is usually called nonholonomic problem. Another one is on dynamics, involving second-order or higher differential constraints. Now it is insufficient to model the aerial robot with only a point in the world space, since six variables are needed to indicate the position of the aerial robot in a three dimensional Euclidean space. For most cases, the configuration space is not a simple Euclidean space.

The purpose of the next two sections is to present some motion planning algorithms published in the literature [24, 40, 50, 70, 76, 77, 84, 86, 117]. Consistently keeping a safe distance from obstacles, and producing smooth paths that exhibit desirable properties (e.g. duration, energy usage) are typical requirements. Low computational complexity is therefore generally an important goal for an algorithm. A faster algorithm can allow a more rapid update of the solution. The choice of the algorithm depends on the type of problem to be solved. A commonly-used metric is obstacle complexity, or the amount of information used and stored in a computer model of the environment. It is generally measured in terms of obstacles number, edges, or vertices. Other metrics are the fill ratio (percentage of the configuration space occupied by obstacles), along with higher order characteristics, such as mean passage width or degree of clustering of obstacles.

Definition 2.3. Completeness A motion planning algorithm is considered to be complete if and only if it finds a path when one exists, and returns a variable stating no path exists when none exists. It is considered to be optimal when it returns the optimal path with respect to some criterion. Any optimal planner is also complete. A sound planner is one that always guarantees the aerial robot will enter the goal region and stop there without hitting any obstacle despite uncertainty in sensing and control.

This definition implies that the uncertainties are bounded. Tractable algorithms approach the motion planning problem by relaxing the completeness requirement to resolution completeness which amounts to finding a solution if one exists, when the resolution parameter of the algorithm is set fine enough. The complexity space PSPACE includes decision problems for which answers can be found with resources such as memory which are polynomial in the size of the input. The runtime is not constrained. The complexity class NP is a subspace of PSPACE.

Definition 2.4. Heuristics Heuristics are creative solutions of problems, both logical and mathematical, by way of experiment, trial and error method or by using analogies. Heuristic methods are applicable everywhere, where a solution of a problem requires large volumes of computation.

The literature does not provide formal proofs for the correctness of operation of heuristic algorithms, but their efficiency is confirmed by simulations made. They are widely applied in expert system, decision support system and operations research.

The purpose of this section is to introduce motion planning methods as far as they are related to aerial robots. Early approaches to the planning problem have mainly focused on the development of complete planners which find a solution if one exists and return failure otherwise. However, complete planners suffer from computational complexity.

2.5.2 Discrete Search Methods

Most planning methods that are based on gridding or cell decomposition fall into the tractable category. A graph is a collection of nodes (vertices) and edges. Typically in motion planning, a node represents a salient location and an edge connects two nodes that correspond to locations that have an important relationship. This relationship could be that the nodes are mutually accessible from each other, two nodes are within line of sight of each other, two pixels/voxels are next to each other in a grid. This relationship does not have to be mutual if the aerial robot can traverse from nodes V_1 to V_2 but not from V_2 to V_1. The edge E_{12} connecting V_1 and V_2 is directed. Such a collection of nodes and edges is called a directed graph. Typically, one searches a tree for a node with some desired properties such as the goal location for the aerial robot.

A path P is a sequence of vertices $P = (v_1, v_2, \ldots, v_m)$ such that the set of consecutive pairs [112]

$$E(P) = \{(v_i, v_{i+1}), i = 1, \ldots, (m-1)\} \tag{2.240}$$

is a subset of E. Each edge in a graph can be furthermore assigned weights $w_{ij} \in \mathbb{R}^+$ and costs $c_{ij} \in \mathbb{R}^+$

Problem 2.10. Shortest path problem Given a graph $G = (V, E)$, costs c_{ij} and start and destination vertices s and d, the shortest path problem is to find the path from s to d such that the sum of costs c_{ij} is minimized.

$$min_P \sum_{(i,j)\in E(P)} c_{ij} \tag{2.241}$$

subject to

$$s, d \in P \tag{2.242}$$

When weights are assigned to edges, the following problem can be stated

Problem 2.11. Weight Constrained Shortest path problem Given a graph $G = (V, E)$, costs c_{ij}, weights $w_{ij} \in \mathbb{R}^+$ and start and destination vertices s and d, the Weight Constrained shortest path problem is to find the path from s to d such that the sum of costs c_{ij} is minimized and the sum of weights is kept below a certain constant W

$$min_P \sum_{(i,j)\in E(P)} c_{ij} \tag{2.243}$$

subject to

$$\sum_{(i,j)\in E(P)} w_{ij} \leq W \quad s, d \in P \tag{2.244}$$

In general, sampling techniques synthesize a dense tree or road map of nodes and edges. Each node represents a particular instantaneous state of the aerial robot and an edge connects two nodes via a planned path. Construction of the trees or road maps is through deterministic or random generation of new nodes and then connecting the new nodes to the old ones via a planned path that can incorporate the aerial robot constraints. Once the construction phase is completed (i.e. a connection exists between start and finish), post processing on the resulting route is typically necessary and desired to ensure smoothness and improve optimality. Overall, these methods are excellent at quickly generating feasible solutions; however, complex paths and increased dimension of the system kinematic and dynamic equations can bog down the computational speed. These methods reduce the problem to that of a graph search by fitting a graph or a road map to the space. Searching a graph means systematically following the edges of the graph so as to visit the vertices of a graph. A graph searching algorithm can discover much about the structure of a graph. Many algorithms begin by searching their input graph to obtain this structural information. Several other graph algorithms elaborate on basic graph searching. In outdoor environments space may not be traversable omnidirectionally. The planner must take environmental information into account when planning paths. This information can be integrated within the graph representation by encoding a specific path section as the difficulty of traversing a specified edge in a particular direction and replacing the underlying undirected graph with a directed one. Many different metrics can be used to evaluate these algorithms. Four metrics are based on the effectiveness of the collision avoidance: number of conflicts, number of collisions, aircraft survival rate and average life expectancy. For these metrics, a conflict is defined as two or more aerial robots within a few designated seconds of each other and a collision is two or more aerial robots within 1 s of each other. The survival rate is a percentage of aircraft that have not been destroyed via collision and average life expectancy is the amount of time the aircrafts stay alive in a simulation. The fifth metric, number of waypoints achieved, is used to evaluate efficiency of the algorithms [56].

2.5.2.1 Simple Space Planning Approaches

The graph search can be tuned to find optimal paths with respect to metrics such as energy, time, distance traversability, safety …as well as combinations of them. There is also the efficiency: minimize the number of nodes that have to be visited to locate the goal node subject to the path optimality criteria. Depth first and breadth first are uninformed: the search just moves through the graph without any preference for or

influence on where the goal node is located. For example, if the coordinates of the goal node are known, then a graph search can use this information to help decide which nodes in the graph to visit (i.e. expand) to locate the goal node. A graph search may choose as its next node to explore one that has the shortest euclidean distance to the goal because such node has highest possibility, based on local information, of getting closest to the goal. However, there is no guarantee that this node will lead to the globally shortest path in the graph to the goal. This is just a good guess. However, these good guesses are based on the best information available to the search. Two main categories of simple space search algorithms exist:

- Uninformed methods where no information exists concerning which node should the algorithm explore next. As a result, nodes are opened one by one until a goal is reached.
- Informed methods use some form of heuristics in order to select the next node to open.

Breadth-First Algorithm

Breadth first algorithm is one of the simplest algorithms for searching a graph. Given a graph $G = (V, E)$ and a source vertex s, breadth first search systematically explores the edges of G to discover every vertex that is reachable from s [29]. It computes the distance (smallest number of edges) from s to each reachable vertex. It also produces a breath-first tree, with root s that contains all reachable vertices. For any vertex v reachable from s, the simple path in the breath-first tree from s to v corresponds to a shorthest path from s to v in G, that is a path containing the smallest number of edges. The algorithm works on both directed and undirected graphs. This search discovers all nodes at distance k from s before discovering any vertices at distance $k + 1$. Breadth first search constructs a breadth first tree, initially containing only its root, which is the source vertex s. Whenever the search discovers a white vertex v in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree, u is the predecessor or parent of v in the breadth first tree. Since a vertex is discovered at most once, it has at most one parent. The search starts at the root then visits all of the children of the root first. Next, the search then visits all of the grand children and so forth. The belief here is that the target node is near the root so this search would require less time. In 2D, Four point connectivity will only have edges to the North, South, East and West, whereas 8 point connectivity will have edges to all pixels surrounding the current pixel: North, East, South, West, North–East, North–West, South–East, South–West. In 3D, there are 26 neighbors for a 45° discrimination [34].

The pseudo-code breadth first search Algorithm 2 in page 118 assumes that the input graph $G = (V, E)$ is represented using adjacency lists. It attaches several additional attributes to each vertex in the graph. If u has no predecessor (i.e if $u = s$ or u has not been discovered), then $u.\pi = NULL$. The attribute $u.d$ holds the distance from the source s to vertex u computed by the algorithm. The algorithm

Algorithm 2 Breadth-First Algorithm: BFA

1. for each vertex $u \in G, V - \{s\}$
2. u.color = WHITE
3. u.d = ∞
4. $u.\pi$=$NULL$
5. s.color = GRAY
6. s.d = 0
7. $s.\pi$=$NULL$
8. Q = $\emptyset$
9. ENQUEUE(Q, s)
10. while $Q \neq \emptyset$
11. u = DEQUEUE (Q)
12. for each $v \in G.Adj\ [u]$
13. if v.color == WHITE
14. v.color = GRAY
15. v.d = u.d +1
16. $v.\pi$=u
17. ENQUEUE(Q, v)
18. u.color = BLACK

also uses a First-In, First-Out (FIFO)queue Q.The procedure works as follows. With the exception of the source s, lines 1–4 paint every vertex, white, set u.d to be infinity for each vertex u, and set the parents of every vertex to be NULL. Line 5 paints s gray, it is discovered as the procedure begins. line 6 initializes $s.d$ to 0, and line 7 sets the predecessor of the source to be NULL. Lines 8–9 initialize Q to the queue containing just the vertex s. The while loop of lines 10–18 iterates as long as there remain gray vertices, which are discovered vertices that have not yet had their adjacency lists fully examined. This while loop maintains the following invariant: at the test in line 10, the queue Q consists of the set of gray vertices. Line 11 determines the gray vertex u at the head of the queue Q and removes it from Q. The for loop of lines 12–17 considers each vertex v in the adjacency list of u. If v is white, then it has not yet been discovered, and the procedure discovers it by executing lines 14–17. The procedure paints vertex v gray, sets its distance $v.d$ to $u.d + 1$, records u as its parent $v.\pi$ and places it at the tail of the queue Q. Once the procedure has examined all the vertices on u's adjacency list, it blackens u in line 18. The loop invariant is maintained because whenever a vertex s is painted gray in line 14, it is also enqueueud in line 17 and whenever a vertex is dequeueud it is also painted in black.

The **wave-front planner** is an implementation of a breadth first search. Breadth First search produces the shortest path to the start node in terms of link strengths. Since the wavefront planner is a breadth-first search, a four point connectivity wave front algorithm produces the shortest path with respect to the distance function. It has an underlying graph, where each node corresponds to a pixel or a voxel and neighboring pixels or voxels have an edge length of one. In general, a breadth first search is implemented with a list in a First-In First-Out (FIFO) manner, a queue. The depth first search contrasts in that the nodes are placed in a Last-In First-Out (LIFO) manner, a stack. Another common search is called a greedy search which expands

nodes that are closest to the goal. Here the data structure is called a priority queue in that the nodes are placed into a sorted list based on a priority value. This priority value is a heuristic that measures distance to the goal node.

Depth First Algorithm

The strategy followed by depth first algorithm is to search deeper in the graph whenever possible. This search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it. A depth first search starts at the root, chooses a child then that node's child and so on until finding either the desired node or a leaf. If the search encounters a leaf, the search then backs up a level and then searches through an unvisited graph until finding the desired node is found or all nodes are visited in the Algorithm 3 in page 119 [29].

Algorithm 3 Depth First Algorithm: DFA

```
1. for each vertex u ∈ G.V
2. u.color = WHITE
3. u.π = NULL
4. time = 0
5. for each vertex u ∈ G.V
6. if u.color == WHITE
7. DFA-VISIT(G, u)
```

Procedure DFA works as follows. Lines 1–3 paint all vertices white and initializes their π attributes to NULL. Line 4 resets the global time counter. Lines 5–7 check each vertex in V in turn and, when a white vertex is found, visit it using DFA-VISIT. Every time DFA-VISIT(G, u) is called in line 7, vertex u becomes the root of a new tree in the depth first forest. When DFA returns, every vertex u has been assigned a discovery time $u.d$ and a finishing time $u.f$.

Algorithm 4 Depth First Visit Algorithm: DFA-VISIT

```
1. time = time + 1
2. u.d = time
3. u.color = GRAY
4. for each v ∈ G.ADJ [u]
5. if v.color == WHITE
6. v.π = u
7. DFA-VISIT(G, v)
8. u.color = BLACK
9. time = time +1
10. u.f = time
```

In each call DFA-VISIT (G, u), vertex u is initially white. Line 1 increments the global variable time, line 2 records the new value of time as the discovery time $u.d$ and line 3 paints u gray. Lines 4–7 examine each vertex v adjacent to u and recursively visit v if it is white. As each vertex $v \in Adj\,[u]$ is considered in line 4, edge (u, v) is explored by the depth first search. Finally, after each vertex leaving u has been explored, lines 8–10 paint u black, increment time, and record the finishing time in $u.f$.

Branch and Bound Algorithm

There are two elements to a generic brand-and-bound Algorithm 5 in page 120 [38]: Branching and Bounding.

Algorithm 5 Branch and Bound Algorithms

1. Branching: The problem is divided into subproblems by partitioning the search space. Each subproblem is further divided in the same way, and the algorithm proceeds by searching a tree of sub-problems, hence branching
2. Bounding: a lower bound on the optimal cost of each subproblem is found by solving a relaxed simpler form of that subproblem. These bounds are then used to identify if a branch requires further subdivision. If the subproblem is infeasible or the solution to the subproblem is worse than the cost of the best feasible solution found so far, then the branch is said to be fathomed and no further branching is necessary.

The following Algorithm 6 in page 120 illustrates the direct mapping of the branch and bound algorithm to the 2D avoidance problem.

Algorithm 6 Depth 2D branch Algorithms

1. Branching: the problem of avoidance obstacles 1…M can be divided into two subproblems
2. the problem of passing clockwise around obstacle 1 and avoiding obstacles 2…M or
3. the problem of passing counterclockwise around obstacle 1 and avoiding obstacles 2…M
4. Bounding: the shortest path passing clockwise (or counterclockwise) around obstacle 1 and avoiding obstacles 2…M must be longer than the shortest path passing clockwise (or counterclockwise) around obstacle 1 and ignoring obstacles 2…M

The global optimality of any solution obtained from a branch and bound method is guaranteed, assuming that the globally optimal solution to each evaluated subproblem is found. In the avoidance case, this corresponds to finding the best path passing on a given side of a set of active obstacles and ignoring all others. Because this subproblem no longer involves a choice of side, it can be solved by nonlinear programming. Although, it is difficult to express the concept of one side or another in terms of an explicit constraint, it can be achieved by initializing the search on the appropriate side

and employing a primal-dual optimization method which is then checked against the desired path.

Dijkstra Algorithm

Dijkstra's algorithm solves the single source shortest-paths problem on a weighted directed graph $G = (V, E)$ for the case in which all edge weights are nonnegative, $w(u, v) \geq 0$ for each edge $(u, v) \in E$. This algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined [29]. The algorithm repeatedly selects the vertex $u \in V - S$, with the minimum shortest path estimate, adds u to S, and relaxes all edges leaving u.

Algorithm 7 Dijkstra Algorithm: DIJKSTRA

1. INITIALIZE-SINGLE-SOURCE(G, s)
2. S = $\emptyset$
3. Q= G.V
4. while $Q \neq \emptyset$
5. u = EXTRACT-MIN(Q)
6. S = S $\cup \{u\}$
7. for each vertex $v \in G.Adj\,[u]$
8. RELAX(u, v, w)

Procedure DIJKSTRA 7 in page 121 works as follows. Line 1 initializes the d and π values and line 2 initializes the set S to the empty set. The algorithm maintains the invariant that $Q = V - S$ at the start of each iteration of the while loop of lines 4–8. Line 3 initializes the min-priority queue Q to contain all the vertices in V. Since $S = \emptyset$ at that time, the invariant is true after line 3. Each time through the while loop of lines 4–8, line 5 extracts a vertex u from $Q = V - S$ and line 6 adds it to set S, thereby maintaining the invariant. Vertex u therefore, has the smallest shortest-path estimate of any vertex in $V - S$. Then, lines 7–8 relax each edge (u, v) leaving u, thus updating the estimate $v.d$ and the predecessor $v.\pi$ if the shortest path to v can be improved.

Because Dijkstra's algorithm always chooses the lightest or closest vertex in $V - S$ to add to set S, it is said to be a greedy strategy. Greedy algorithms do not always yield optimal results in general, but Dijkstra's algorithm computes shortest paths. Dijkstra algorithm finds a minimum cost path between a start state s_{start} and a goal state s_{goal} (or a set of goal states) in a graph with non-negative edge costs. For each state s in the graph, it maintains a value g(s), which is the minimum cost proven so far to reach s from s_{start}. Initially all $g(s)$ are ∞, except for the start state, whose g-value is initilalized at 0. The algorithm maintains an OPEN queue containing all locally inconsistent states, i.e. states that may have successors s' for which $g(s') > g(s)+c(s, s')$, where $c(s, s')$ is the cost of traversing the edge between s and s'. Initially, OPEN only contains the start state s_{start}. Continually, Dijkstra

algorithm extracts the state s from OPEN with minimal g(s) value, and expands it. That is, it updates the g-values of the successors of s, and puts them on the OPEN queue if their g-values was decreased.

A* Algorithm

The A* algorithm is a generalization of Dijkstra's algorithm that improves its running time in practice if a heuristic is available, by focusing the search towards the goal. If the heuristic is admissible i.e. if $h(s) \leq c^*(s, s_{goal})$ for all s, A* is guaranteed to find the optimal solution in optimal running time. If the heuristic is also consistent, i.e. if $h(s) \leq c^*(s, s') + h(s')$, it can be proven that no state is expanded more than once by the A* algorithm. The A* algorithm is used to compute minimum cost paths in graphs in many applications ranging from map navigation to path planning. The A* algorithm searches a graph efficiently with respect to a chosen heuristic.The A* algorithm returning an optimal path in the heuristic is optimistic. An optimistic or admissible heuristic always returns a value less than or equal to the cost of the shortest path from the current node to the goal node within the graph. The A* algorithm has a priority queue which contains a list of nodes sorted by priority, determined by the sum of the distance traveled in the graph thus far from the start node and the heuristic. The first node to be put into the priority queue is naturally the start node. Next, the start node is expanded by putting all adjacent nodes to the start node into the priority queue sorted by their corresponding priorities. These nodes can naturally be embedded into the aerial robot free space and thus have values corresponding to the cost required to traverse between the adjacent nodes. The output of the A* algorithm is a back pointer path, which is a sequence of nodes starting from the goal and going back to the start. The difference from Dijkstra is that A* expands the state s in OPEN with a minimal value of $g(s) + h(s)$ where $h(s)$ is the heuristic that estimates the cost of moving from s to s_{goal}. Let $c^*(s, s')$ denote the cost of the optimal solution between s and s′. The "Open List" saves the information about the parental nodes found as a candidate solution. The 3D cells in the grid not only have elements in the neighbourhood on the same height level used but also have cell nodes with locations above and below. The output of the A* algorithm is a back pointer path, which is a sequence of nodes starting from the goal and going back to the start. Two additional structures are used, an open set O and a closed set C. The open set O is the priority queue and the closed set C contains all processed nodes [24]. The Euclidean distance between the current point and the destination goal, divided by the maximum possible nominal speed can be employed as a heuristic function. This choice ensures that the heuristic cost will always be lower than the actual cost to reach the goal from a given node and thus the optimum solution is guaranteed [35]. The pseudo code for this approach can be formulated as Algorithm 8 in page 123.

Variations of A* algorithm are:

Algorithm 8 A* Algorithm

1. Input: A graph
2. Output: A path between start and goal nodes
3. Repeat
   a. Pick n_{best} from 0 such that $f(n_{best}) < f(n)$
   b. Remove n_{best} from O and add to C
   c. If $n_{best} = q_{goal}$, EXIT
   d. expand n_{best}: for all $x \in Star(n_{best})$ that are not in C
   e. if $x \notin O$ then
   f. add x to O
   g. else if $g(n_{best}) + C(n_{best}, x) < g(x)$ then
   h. update x's back pointer to point to n_{best}
   i. end if
4. Until O is empty

1. When $f(n) = h(n)$, then the search becomes a greedy search because the search is only considering what it believes is the best path to the goal from the current node.
2. When $f(n) = g(n)$ the planner is not using any heuristic information but rather growing a path that is shortest from the start until it encounters the goal.

Weighted A* extends A* by allowing to trade-off running time and solution quality. It is similar to A*, except that it inflates the heuristic by a value $\epsilon \geq 1$ and expands the state s in OPEN with minimal $f(s) = g(s) + \epsilon C(s)$. The higher ϵ the greedier the search and the sooner a solution is typically found. The suboptimality of solutions found by weighted A* is bounded by ϵ, i.e. the solution is guaranteed to be no costlier than ϵ times the cost of the optimal solution. Weighted A* may expand states more than once (as the inflated heuristic $\epsilon.h(s)$ is typically not consistent). However, if $h(s)$ itself is consistent, it can be proven that restricting states to being expanded no more than once does not invalidate the $\epsilon-$suboptimality bound [59].

An extension of the A^* algorithm is the θ^* algorithm. This algorithm does not restrict the search to the neighboring nodes, but it allows to search through the nodes in line of sight with the expanded node. The search can also be focused by using a heuristic function of distance to the goal. A dynamic algorithm that calculates the trajectories while the weather hazards is updated is presented in [49].

Roadmap Algorithm

This section focuses on a class of topological maps called roadmaps. A roadmap is embedded in the free space and hence the nodes and edges of a roadmap also carry a physical meaning. Using a roadmap, the planner can construct between any two points in a connected component of the aerial robot free space by first adding a collision freee path onto the roadmap, traversing the roadmap to the vicinity of

the goal and then constructing a collision free path from a point on the roadmap to the goal. The bulk of the motion occurs on the roadmap and thus searching does not occur in a multi-dimensional space, whether it be the workspace or the configuration space [94, 117].

Definition 2.5. Roadmap A union of one-dimensional curves is a roadmap RM for all q_{start} and q_{goal} in Q_{free} that can be connected by a path, the following properties hold

1. **Accessibility**: There exists a path from $q_{start} \in Q_{free}$ to some $q'_{start} \in RM$
2. **Separability** : There exists a path from some $q'_{start} \in RM$ to $q_{goal} \in Q_{free}$
3. **Connectivity**: There exists a path in RM between q'_{start} and q'_{goal}

The road map algorithm applies sampling methods to the trajectory planning and dynamic planning problems. It handles high dimensionality and global constraints. Sampling methods are not based on a rigorous mathematical structure. Despite the existence of numerous distinct sampling techniques, they all share similar defining actions.

Visibility Graph

The visibility graph uses corner points of obstacles. If the environment is represented as a configuration space, the polygon description of all obstacles is already available. The list of all start and end points of obstacle border lines plus the aerial robot's start and goal position is available. A complete graph is then constructed by linking every node position to every other one. Finally, all the lines that intersect an obstacle are deleted, leaving only the lines that allow the flight from one node to another in a direct line. The characteristic of the Algorithm 9 presented below in page 125 are as follows. Let $V = \{v_1, \ldots, v_n\}$ be the set of vertices of the polygons in the configuration space as well as the start and goal configurations. To construct the visibility graph, other vertices visible to $v \in V$, must be determined. The most obvious way to make this determination is to test all line segments $vv_i, v \neq v_i$, to see if they interset an edge of any polygon. A more efficient way is the rotational sweep algorithm. For the problem of computing the set of vertices visible from v, the sweep line I is a half-line emanating from v and a rotational sweep rotating I from 0 to 2π are used.

The key of this algorithm is to incrementally maintain the set of edges that intersect I, sorted in order of increasing distance from v. If a vertex v_i is visible to v, then it should be added to the visibility graph. It is straightforward to determine if v_i is visible to v. Let S be the sorted list of edges that intersects the half line emanating from v. The set is incrementally constructed as the algorithm runs. If the line segment vv_i does not intersect the closed edge in S and if I does not lie between the two edges incident on v (the sweep line does not intersect the interior of the obstacles at v.) then v_i is visible from v [24]. Some planning problems may be solved according to the following Algorithm 10 in page 125.

Algorithm 9 Visibility Algorithm

1. **Input**: A set of vertices $\{v_i\}$ (whose edges do not interset) and a vertex v.
2. **Output**: A subset of vertices from $\{v_i\}$ that are within line of sight of v.
3. For each vertex v_i, calculate α_i, the angle from the horizontal axis to the line segment vv_i
4. Create the vertex list ϵ, containing the α_i sorted in increasing order.
5. Create the active list S, containing the sorted list of edges that intersect the horizontal half line emanating from v.
6. for all α_i do
7. if v_i is visible to v then
8. add the edge (v, v_i) to the visibility graph
9. endif
10. if v_i is the beginning of an edge E, not in S, then
11. insert the edge E into S
12. endif
13. if v_i is the end of an edge in S then
14. Delete the edge from S
15. endif
16. endfor
17. endfor

Algorithm 10 Real Time Application Algorithm

1. The 2D route planning is based on calculating a visibility graph for the circumvention of non-navigable areas, looking for the shortest path in the visibility graph and the launch of an algorithm to optimize the order of passing over all mission areas;
2. Planning for rectilinear trajectories 4D (3D coordinates over time) is calculated for a nominal feed rate and a constant altitude of between rallying points of the route;
3. The planning process calculates the operative sequence that models the trajectories and changes in direction.

Edge-Sampled Visibility Graph

This algorithm approximately solves the 3D path length minimization point aerial robot problem. This algorithm assigns multiple vertices along edges of polyhedral obstacles so that there is a minimum edge length n and builds a visibility graph from this expanded set of vertices.

Voronoi Roadmap

Given the difficulty in controlling aerial vehicles precisely enough to follow the minimum-distance path without risk of colliding with obstacles, many skeleton-based road map approaches have been taken. The Voronoi approach builds a skeleton that is maximally distant from the obstacles, and finds the minimum distance path that follows this skeleton. This algorithm is a 2D algorithm, complete but not optimal. Voronoi diagram is a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space [113]. Given a set of points S, the corresponding Voronoi diagram are generated, each point P has its own

Voronoi cell which consists of all points closer to P than any other points. The border points between polygons are the collection of the points with the distance to shared generators [5]. A Voronoi diagram is another method for extracting distance node information from a given 2D environment. The algorithm works by constructing a skeleton of points with minimal distances to obstacles. A free space F in environment (white voxels) is defined as well as an occupied space F' (black voxels), $b' \in F'$ is a basis point for $p \in F$ if and only if b has minimal distance to p, compared with all other points in F', Voronoi diagram = $\{p \in F | p$ has at least two basis points $\}$. If the logic of the algorithm requires the set of nodes to be deleted during an incremental update to form a tree, then supplying the subsequently executed portions of the code, with anything but a tree will likely cause the code to loop or crash. Similarly if a new node is to be inserted into the Voronoi Algorithm 11 in page 126, then the code may subsequently need to access the coordinates of this node [55].

Algorithm 11 VORONOI Algorithm

```
1.  ϵ = lower-bound
2.  Repeat
3.  x = ComputeData(ϵ)
4.  success = CheckConditions (x, ϵ)
5.  If (not success) then
6.  ϵ=10ϵ
7.  reset data structure appropriatedly
8.  until (success OR ϵ > upper-bound)
9.  ϵ = lower-bound
10. If (not success) then
11. illegal = CheckInput()
12. If (illegal) then
13. clean data locally
14. restart computation from scratch
15. else
16. x = DesperateMode()
```

Typical computational units are given in Algorithm 11 in page 126. Line 1 set ϵ to maximum precision while line 3 computes some data. Line 4 checks topological and conditions while line 6 relaxes the ϵ threshold and line 7 makes sure to reset ϵ. Line 10 checks locally for soundness of input and line 13 fixes the problem in the input data. Finally line 14 replaces 'correct' by 'best possible'. If the relaxation of the ϵ threshold and the heuristics of the multi-level recovery process have not helped to compute data that meets the topological conditions then the code finally enters 'desperate mode', because the 'optimum' is replaced by 'best possible'. Similarly, if some numerical data could be computed but it failed the numerical sanity checks then VORONOI accepts the data whichever meets most of the sanity checks. Finally, any violation of a topological condition is cured by forcing its validity. For instance, if the code cannot break up a cycle of nodes, during the incremental update then desperate mode will force a break-up by inserting a dummy degree two node randomly on one

of the edges of the cycle. VORONOI always first attempts to perform a computation as it would normally perform it, irrelevant of whether or not it had already entered desperate mode once before.

One problem of this approach is that it allows lines to pass very closely to an obstacle, and so this would only work for a theoretical point robot. However, this problem can be easily solved by virtually enlarging each obstacle by at least half of the aerial robot largest dimension before applying the algorithm.

Algorithm 12 VISIBILITY-CLOSE Algorithm

1. Identify each obstacle and each border with a unique label or color
2. Iteration (i=2, until no more changes, i++)
3. a: if a free voxel is neighbor to a labeled voxel or a border then label the voxel 'i' in the same color.
4. b: If a free voxel is overwritten twice or more in different colors by this point, then make it a Voronoi point.
5. c: If a voxel and its top or right neighbor, then make this voxel a Voronoi Point.

Visibility approach is an exact solution to the 2D point aerial robot problem. Its pseudo-code is presented in Algorithm 12. This approach uses the knowledge that the shortest path grazes polygonal obstacles at their vertices and builds a road map of lines connecting each vertex with all vertices visible from its position. Since the minimum-length path comes arbitrarily close to obstacles many times in a typical path, this approach offers no safety buffer to prevent collisions in the case of systems with uncertainty in their position. To avoid this problem, the obstacle space is expanded by a ball larger than the aerial robot's longest radius. More advanced versions of the visibility graph algorithm can be used to also include the aerial robot orientation for flying.

The **Delaunay triangulation** tries to construct a Voronoi diagram with much less computational effort. The **Brushfire Algorithm** 13 in page 127 is a discrete graphics algorithm for generating Voronoi diagrams on an occupancy grid (1 for occupied, 0 for free).

Algorithm 13 Brushfire Algorithm

1. Identify each obstacle and each border with a unique label or color
2. Iteration (i=2, until no more changes, i++)
3. a: if a free voxel is neighbor (8 nearest or 26 nearest neighbor) to a labeled voxel or a border then label the voxel 'i' in the same color.
4. b: If a free voxel is overwritten twice or more in different colors by this point, then make it a Voronoi point.
5. c: If a voxel and its top or right neighbor (four nearest neighbor), then make this voxel a Voronoi Point.

The border labels or colors are slowly moving in from the sides toward the center, so in that sense brushfire is similar to a flood-fill algorithm [21].

Grid Based State Space Search

This method defines an arbitrary speed-varying safety corridor, making this particular algorithm one of very few trajectory planning algorithms with a proven explicit safety guarantee. This algorithm discretizes the entire state space of the aerial robot onto a lattice, and searches the lattice for the time-optimal path that satisfies the safety corridor. Although the algorithm converges as a polynomial of the number of obstacles, it is a high-order polynomial that is exponential with the number of dimensions, making practical real-time implementation difficult due to high dimensionality of the state-space. It has also difficulty in solving planning problems for the case of under-actuated vehicles, such as an aerial robot.

State Space Navigation Function with Interpolation

The grid-based state space search method approximates the time-optimal path to a goal. Instead of returning only a single trajectory, it returns a navigation function over the state space. This navigation function can be computed by either value iteration or control policy iteration, although value iteration is more popular. For a given state, performing gradient descent on this navigation function will produce an approximately time-optimal trajectory to the goal. Interpolation between lattice points allows a continuous function which can be used for feedback. The algorithm takes on the same order of complexity as the grid-based state space search.

Reachability Graph

This approach also uses a body-centered frame of reference: for each state, the tree explores variety of states including the maximum deflection states. The combinatorial complexity of such a process is often prohibitive, and the tree quickly fills the space close to the initialization point. The basic approach has been employed for curvature constrained path problem in 2D. Another way to make this approach tractable is to use cell-based pruning: the configuration space is divided into cells, and the reachability graph is set up to be a tree that has no more than one leaf ending in each cell.

2.5.2.2 Complex Space Planning Approaches

While complete motion algorithms do exist, they are rarely used in practice since they are computationally infeasible in all but the simplest cases. Deterministic and complete algorithms for the solution of kinodynamic motion-planning problems require

at least exponential time in the dimension of the state space of the dynamical system, which is usually at least twice the dimension of the underlying configuration space, and polynomial in the number of obstacles. As a consequence, available algorithms are implementable at the current technology levels, only for systems of very small dimensions. For aerial robots, one has to resort to heuristic techniques or seek alternative formulation of the problem [47]. For this reason, probabilistic methods have been developed. The idea is to create a graph of randomly generated collision-free configurations with connections between these nodes made by a simple and fast local planning methods [44, 80, 105]. Probabilistic planners represent a class of efficient methods. They belong to the general family of sampling-based methods. A configuration that does not cause a collision is added to the current road map and connected if possible to other already stored configurations. The above strategy is quite general and may lead to different planning methods depending on the specific design choices, and mainly on the criterion for selecting the samples to be checked for collision [126].

Paths are constructed connecting randomly sampled points. The **Probabilistic RoadMap** PRM algorithm constructs a graph of feasible paths off-line and primarily aims at multiple-query applications, in which several motion planning problems need to be solved in the same environment. Incremental based algorithms have been developed for single query real time applications [136]. The **Rapidly expanding random Tree** (RRT) algorithm has been shown to be probabilistically complete, with an exponential decay of the probability of failure. The **rapidly exploring random graph** (RRG) are proposed to find feasible trajectories that can handle specifications given in the form of deterministic μ-calculus, which includes **the Linear Temporal Logic**, RRG incrementally builds a graph of trajectories, since specifications given in μ-**calculus**, in general require infinite-horizon looping trajectories, which are not included in tree. Probabilistic road-map methods operate as follows: a preprocessing phase, a set of configurations in the free space is generated by sampling configurations at random and removing those that put the aerial robot in collision with an obstacle. Those nodes are then connected into a road map graph by inserting edges between configurations if they can be connected by a simple and fast local planning method, example a straight line planner. This road map can then be queried by connecting given start and goal configurations to nodes in the road map (again using the local planner) and then searching for a path in the road map connecting these nodes. Various sampling schemes and local planners have been used. Both the PRM and RRT algorithms are **probabilistically complete** in the sense that the probability that these algorithms return a solution, if one exists, converges to one as the number of samples approaches infinity, under mild technical assumption. In most applications of aerial robotics, finding not only a feasible solution, but also one that has a good quality (in terms of a cost function) is highly desired.

Definition 2.6. Asymptotical Optimality An algorithm is said asymptotically optimal if the solution returned by the algorithm converges to an optimal solution almost surely as the number of samples approaches infinity.

Probabilistic RoadMap Methods

The roadmap methods described above are able to find a shortest path on a given path. The issue most path planning methods are dealing with is how to create such a graph. To be useful for path planning applications, the roadmap should represent the connectivity of the free configuration space well and cover the space such that any query configuration can be easily connected to the roadmap. Probabilistic RoadMap approach (PRM) is a probabilistic complete method that is able to solve complicated path planning problems in arbitrarily high dimension configuration spaces. The basic concept in PRM is that rather than attempt to sample all of C-space, one instead samples it probabilistically. This algorithm operates in two phases, a roadmap construction phase in which a roadmap is constructed within the C-space and a query phase, in which probabilistic searches are conducted using the roadmap to speed the search:

1. **Roadmap Construction Phase**: tries to capture the connectivity of free configuration space. An undirected, acyclic graph is constructed in the aerial robot C-space in which edges connect nodes if and only if a path can be found between the nodes (which correspond to way-points). The graph is grown by randomly choosing new locations in C-space and attempting to find a path from the new location to one of the nodes already in the graph while maintaining the acyclic nature of the graph. This relies on a local path planner to identify possible paths from the randomly chosen location and one or more of the nodes in the graph. The choice of when to stop building the graph and the design of the local path planner are application specific, although performance guarantees are sometimes possible.

    a. **Sample generation** (s samples): generates samples in the configuration space
    b. **Milestone construction** (m milestones): compute milestones that correspond to the samples in the free space by performing discrete collision queries. A milestone lies in the free space and does not collide with the obstacles.
    c. **Proximity computation** (m milestones, $m-k$ neighbors): For each milestone, find other milestones that are nearest to it. For each milestone computed, k-nearest neighbors are sought. In general, there are two types of k-nearest neighbors: exact and approximated (which is faster by allowing a small relaxation).
    d. **Local planning** (m milestones, e edges): connect nearby milestones using local planner and form a roadmap. Local planning checks whether there is a local path between two milestones, which corresponds to an edge on the roadmap. Many methods are available for local planning. The most common way is to discretize the path between two milestones into n_i steps and the local path exists when all the intermediate samples are collision free, performing discrete collision queries at those steps. Local planning can also be performed by continous collision detection, a local RRT algorithm or computing distance bounds. It is the most expensive part of the PRM algorithm.

2. **Query Phase**: When a path is required between two configurations s and g, paths are first found from s to node $\bar{s}$ in the roadmap and from g to some node $\bar{g}$ in the roadmap. The roadmap is then used to navigate between $\bar{g}$ and $\bar{s}$. After every query, the nodes s and g and the edges connecting them to the graph can be added to the roadmap. As in the learning phase, the query phase relies on a heuristic path planner to find local paths in the configuration space.

   a. **Query connection** (a graph): Connect initial and goal configurations of query to the roadmap. For both of these configurations, the k-nearest milestones are found on the roadmap and edges are added between query and milestones that can be connected by local planning.
   b. **Graph search**: execute a graph search algorithm on the roadmap and find collision free path. The search algorithm tries to find a path on the roadmap connecting initial and goal configurations. Many solutions for this, such as Depth First Search, Best First Search, A* can find the shortest path, which is not necessary for the basic motion planning problem.

The local planner should be able to find a path between two configurations in simple cases in a small amount of time. Given a configuration and a local planner, one can define the set of configurations to which a local planning attempt will succeed. This set is called the visibility region of a node under a certain local planner. The larger the visibility region is, the more powerful the local planner. The most straightforward sampling scheme shown in page Algorithm 14 in 137 is to sample configurations uniformly and randomly over the configuration space.

Algorithm 14 Sampling Scheme Algorithms

1. nodes ← sample N nodes random configuration
2. for all nodes
3. find $k_{nearest}$ nearest neighbors
4. if collision check and $\gamma \leq \gamma_{max}$ then roadmap ← edge
5. end

For every node, a nearest neighbor search is conducted. Several constraints have to be satisfied during the construction phase before an edge connection between two nodes is possible.

Algorithm 15 in page 132 can be applied to an aerial robot flying at a constant altitude and considered as a Dubins Vehicle.

Algorithm 15 Dubins PRM Algorithms

1. PRM roadmap construction (q_0, q_g)
2. shortest path (q_0, q_g) ← modified A^* between q_0 and q_g with Dubins metric.

The above algorithm depicts the necessary steps in the query phase for flight planning. The initial configuration $q_0 = (x_0, y_0, z_0, \gamma_0, \chi_0)^T$ and the final configuration $q_f = (x_f, y_f, z_f, \gamma_f, \chi_f)^T$ with a given position and arbitrary γ_f, χ_f have to be connected to their k nearest neighbors in the roadmap. As a second step, a modified A^* graph search algorithm is applied to find the shortest path from q_0 to q_f in the roadmap. The modifications made for this algorithm further adds flight path constraints limiting

- the maximum allowed heading $\Delta\chi_{max}$ due to a limited turning radius
- the minimum length of a straight line segment L_{min} by approximating it as the length of an arc piece by $L_{min} = 2\Delta\chi_{max} R_{min} C_{safety}$ where C_{safety} is a safety factor and R_{min} is the turning radius determined by the motion primitive with the smallest turning radius.

Only if these constraints are satisfied, the specific node is included into the A^* graph search. PRM waypoints serve as intermediate goal regions which are represented by a sphere with a certain radius around x_g. A goal region of the size less than the minimal turning radius has been found acceptable. The planner performs an informed search based on a partially greedy cost functional. It is desired to obtain depth first behavior in the free space that results in fast search space exploration and breadth first exploration when close to the obstacles or in the proximity of a goal region to minimize the probability to end up in a local minimum [83, 87, 116].

Rapidly Expanding Random Tree: RRT

The Rapidly expanding Random Tree approach is suited for quickly searching high-dimensional spaces that have both algebraic and differential constraints. The key idea is to bias the exploration toward unexplored portions of the space by sampling points in the state space and incrementally pulling the search tree toward them, leading to quick and uniform exploration of even high-dimensional state spaces. A graph structure must be built with nodes at explored positions and with edges describing the control inputs needed to move from node to node. Since a vertex with a larger Voronoi region has a higher probability to be chosen as (x_{near}) and it is pulled to the randomly chosen state as close as possible, the size of larger Voronoi regions is reduced as the tree grows. Therefore, the graph explores the state space uniformly and quickly. To improve the performance of the RRT, several techniques have been proposed such as biased sampling and reducing metric sensitivity. The basic RRT Algorithm 16 in page 133 operates as follows: the overall strategy is to incrementally grow a tree from the initial state to the goal state. The root of this tree is the initial state; at each iteration, a random sample is taken and its nearest neighbor in the tree computed. A new node is then created by growing the nearest neighbor toward the random sample.

For each step, a random state (x_{rand}) is chosen in the state space. Then (x_{near}) in the tree that is the closest to the (x_{rand}) in metric ρ is selected. Inputs $u \in \mathbf{U}$, the input set are applied for Δt, making motions toward (x_{rand}) from (x_{near}). Among

Algorithm 16 RRT Basic Algorithm

1. Build RRT (x_{init})
2. G_{sub}, init(x_{init})
3. for k = 1 to maxIterations do
4. $x_{rand} \leftarrow$ RANDOM-STATE()
5. $x_{near} \leftarrow$ NEAREST-NEIGHBOR(x_{rand}, G_{sub})
6. u_{best}, x_{new}, success $\leftarrow$ CONTROL($x_{near}, x_{rand}, G_{sub}$);
7. if success
8. G_{sub}.add-vertex x_{new}
9. G_{sub}.add-edge $x_{near}, x_{new}, u_{best}$
10. Return G_{sub}
11. RRT-EXTEND
12. $V \leftarrow \{x_{init}\}, E \leftarrow \emptyset, i \leftarrow 0$;
13. While $i < N$, do
14. $G \leftarrow (V, E)$
15. $x_{rand} \leftarrow$ Sample(i); $i \leftarrow i + 1$)
16. (V, E) $\leftarrow$ Extend(G, x_{rand})
17. end

the potential new states, the state that is as close as possible to (x_{rand}) is selected as a new state (x_{new}). The new state is added to the tree as a new vertex. This process is continued until (x_{new}) reaches (x_{goal}).

Sampling: The function $Sample : N \rightarrow X_{free}$ returns independent identically distributed (i.i.d) samples from X_{free}.

Steering: Given two points $x, y \in X$, the function Steer: $(x, y) \mapsto z$ returns a point $z \in \mathbb{R}^d$ such that z 'is closer' to y than x is. The point z returned by the function Steer is such that z minimizes $\|z - y\|$ while at the same time maintaining $\|z - x\| \leq \eta$ for a prescribed $\eta > 0$, i.e. Steer(x,y)=$argmin_{z \in \mathbb{R}^d, \|z-x\| \leq \eta} \|z - y\|$

The pseudo-code for the procedure RRT-EXTEND is given in Algorithm 17 in page 133.

Algorithm 17 RRT-EXTEND Algorithm

1. $V' \leftarrow V, E' \rightarrow E$
2. $x_{nearest} \leftarrow$ Nearest(G, x);
3. $x_{new} \leftarrow Steer(x_{nearest}), x)$;
4. If ObstacleFree($x_{nearest}, x_{free}$) then
5. $V' \leftarrow V' \cup \{x_{new}\}$
6. $E' \leftarrow E' \cup \{x_{nearest}, x_{new}\}$
7. Endif
8. return $G' = (V', E')$

Nearest Neighbor: Given a graph $G = (V, E)$ and a point $x \in X_{free}$ and a number $n \in N$, the function Nearest Neighbor: $G(x) \mapsto v$ returns a vertex $v \in V$ that is closest to x in terms of a given distance function. Euclidean distance can be used such as:

$$Nearest\,(G=(V,E),x)=argmin_{v\in V}\|x-v\| \tag{2.245}$$

Near Vertices: given a graph $G=(V,E)$, a point $x\in X_{free}$ and a number $n\in N$, the function Near Vertices: $G(x,n)\mapsto V'$ returns a set $V'\subset V$. The Near Vertices procedure can be thought of as a generalization of the Nearest Neighbor procedure in the sense that the former returns a collection of vertices that are close to x, whereas the latter returns only one such vertex that is the closest. Near (G(x,n)) is the set of all vertices with the closed ball of radius r_n centered at x, where $r_n=min\left\{\left(\frac{\gamma\log n}{\chi_d^n}\right)^{1/d},\eta\right\}$, γ is a constant. Hence the volume of this ball is $min\left\{\left(\frac{\gamma\log n}{n}\right)^{1/d},\chi_d\eta_d\right\}$.

Collision Test: Given two points $x,x'\in X_{free}$, the Boolean function Obstacle-Free (x,x') returns True if and only if the line segment between x and x′ lies in X_{free}, i.e. $\left[x,x'\right]\subset X_{free}$.

The RRT grow in a Voronoi biased manner due to the way they process the random samples drawn from the configuration space. If, instead of taking a single sample, k samples are taken, then the nodes in the tree can be sorted according to how many samples they were the nearest neighbor for, and grow from the nodes which collected the most samples. This is the multi-sample RRT (MS-RRT) pseudo-code given in Algorithm 18 in page 134.

Algorithm 18 MS-RRT Algorithm

1. G_{sub}, init(x_{init})
2. for x = 1 to maxIterations do
3. for i = 1 to k do
4. $x_{rand}\leftarrow$ RANDOM-STATE();
5. $x_{near}\leftarrow NEAREST-NEIGHBOR(x_{rand},G_{sub})$;
6. $x_{near}.sampleCount+=1$
7. $x_{near}.sampleAverage+=x_{rand}$
8. $x_{best}\leftarrow max(x.SampleCount, x\in G_{sub}$
9. $x_{next}\leftarrow x_{best}.SampleAverage/x_{best}.SampleCount$
10. u_{best}, x_{new}, success $\leftarrow$ CONTROL($x_{best}, x_{next}, G_{sub}$);
11. if success then
12. $G_{sub}.add_vertex x_{new}$
13. $G_{sub}.add_edge x_{near}, x_{new}, u_{best}$
14. CLEAR-SAMPLE-INFO(G_{sub})
15. Return G_{sub}

During a particular iteration, a node grows toward the average of the samples if collected; this way be viewed as an estimate of the centroid of that node's Voronoi region. As k approaches infinity, the exact Voronoi volumes and Voronoi regions are probabilistically obtained. k may be viewed as a knob which changes the behavior of the algorithm from randomized to deterministic. Starting from the initialconditions,

Algorithm 19 RRG Algorithm

1. $V \leftarrow \{x_{init}\}, E \leftarrow \emptyset, i \leftarrow 0$;
2. While $i < N$, do
3. $G \leftarrow (V, E)$
4. $x_{rand} \leftarrow$ Sample(i); $i \leftarrow i + 1$)
5. (V, E) $\leftarrow$ RRG-EXTEND(G, x_{rand})
6. end

a tree of feasible trajectories is incrementally built, trying to explore efficiently the reachable set. At each iteration, the new candidate is connected to each other of the nodes currently in the trees before discarding it as unreachable from the current tree (in the original RRT, only the closest node is tested for reachability). The RRT criterion of testing the closest node translates into the heuristics of testing the nodes in ascending distance order. The second main difference is that the optimal cost function in the obstacle free case is used as a measure of distance, both in the selection of nodes to expand and in the computation of the optimal control. Once a feasible trajectory has been found, the focus of the search focus from exploration to the optimization of the computed trajectory. To enhance the quality of the new addition to the tree, the nodes can be sorted in ascending order of total cost to reach the final configuration. This optimization heuristics is appropriate when a feasible solution is already available [47]. This method uses a stochastic search over the body-centered frame of reference and expands a tree through a random sampling of the configuration space. This algorithm is proven to be complete in the probabilistic sense, and to produce a trajectory that is feasible given the dynamic constraints of the aerial robot. However, there is no proof of the convergence rate or of optimality.

Rapidly Exploring Random Graphs: RRG

The Rapidly exploring Random Graphs (RRG) Algorithm 19 first extends the nearest vertex and if such extension is successful, it also extends all the vertices returned by the Near Procedure, producing a graph in general. All the extensions resulting in collision-free trajectories are added to the graph as edges and their terminal points as new vertices.

Initially, the algorithm starts with the graph that includes the initial state as its single vertex and no edges; Then they incrementally grow a graph on X_{free} by sampling a state x_{rand} from X_{free} at random and extending the graph towards x_{rand}. In the sequel, every such step of sampling followed by extensions (lines 2–5 of algorithm 19 in page 135) is called a single iteration of the incremental sampling based algorithm [22]. Sets of vertices and edges of the graph maintained by the RRG algorithm can be defined as functions from the sample space Ω to appropriate sets.

Pseudo-code for procedure RRG-EXTEND is given in Algorithm 20 in page 136.

The RRG algorithm inherits the probabilistic completeness as well as the exponential decay of failure, as the number of samples increase, from the RRT.

Algorithm 20 RRG-EXTEND Algorithm

1. $V' \leftarrow V, E' \rightarrow E$
2. $x_{nearest} \leftarrow$ Nearest(G, x);
3. $x_{new} \leftarrow Steer(x_{nearest}, x)$;
4. If ObstacleFree($x_{nearest}, x_{free}$) then
5. $V' \leftarrow V' \cup x_{new}$
6. $E' \leftarrow E' \cup \{(x_{nearest}, x_{new}), (x_{new}, x_{nearest})\}$
7. $X_{near} \leftarrow$ Near $(G, x_{new}, |V|)$
8. For all $x_{near} \in X_{near}$ do
9. if ObstacleFree $(x_{new}, x_{nearest})$ then
10. $E' \leftarrow E' \cup \{(x_{nearest}, x_{new}), (x_{new}, x_{nearest})\}$
11. Endif
12. return $G' = (V', E')$

RRT*

Maintaining a tree structure rather than a graph may be advantageous when differential constraints exist or to cope with modeling errors [154]. The RRG algorithm can be be slightly modified to maintain a tree structure while preserving the asymptotic optimality properties as well the computational efficiency. A tree version of the RRG algorithm, called the RRT* is introduced in this paragraph. Given two points $x, x' \in X_{free}$, Line(x,x'): $[0, s] \rightarrow X_{free}$ denotes the path defined by $\sigma(\tau) = \tau x + (s - \tau)x'$ for all $\tau \in [0, s]$ where $s = \|x' - x\|$. Given a tree G=(V, E), and a vertex $v \in V$, let Parent be a function that maps v to the unique vertex $v' \in V$ such that $(v, v') \in E$.

The RRT* algorithm differs from the RRT and the RRG algorithms only in the way that it handles the Extend procedure. Its pseudo-code is given in Algorithm 21 in page 137. In the description of the RRT* algorithm, the cost of the unique path from x_{init} to a vertex $v \in V$ is denoted by Cost(v). Initially, Cost(x_{init}) is set to zero. Similar to the RRT and RRG, the RRT* algorithm first extends the nearest neighbor towards the sample. However, it connects the new vertex x_{new} to the vertex that incurs the minimum accumulated cost up until x_{new} and lies within the set X_{near} of vertices returned by the Near procedure. RRT* also extends the new vertex to the vertices in X_{near} in order to rewire the vertices that can be accessed through x_{new} with smaller costs. The RRT* inherits the asymptotic optimality of the RRG while maintaining a tree structure rather than a graph [120].

RRT and RRT* are difficult to apply in problems with complicated or underactuated dynamics because they require the design of a two domain specific extension heuristics: a distance metric and node extension metric. A method was proposed

Algorithm 21 RRT*-EXTEND Algorithm

1. $V' \leftarrow V, E' \rightarrow E$
2. $x_{nearest} \leftarrow Nearest(G, x)$;
3. $x_{new} \leftarrow Steer(x_{nearest}, x)$;
4. If $ObstacleFree(x_{nearest}, x_{new})$ then
5. $V' \leftarrow V' \cup x_{new}$
6. $x_{min} \leftarrow x_{nearest}$
7. $X_{near} \leftarrow Near(G, x_{new}, |V|)$
8. For all $x_{near} \in X_{near}$ do
9. if $ObstacleFree(x_{near}, x_{new})$ then
10. $c' \leftarrow Cost(x_{near}) + c(Line(x_{near}, x_{new}))$
11. if c' < Cost(x_{new}) then
12. $x_{min} \leftarrow x_{near}$
13. $E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$,
14. For all $x_{near} \in X_{near}\ x_{min}$ do
15. if $ObstacleFree(x_{new}, x_{near})$ and $Cost(x_{near}) > Cost(x_{new}) + c(Line(x_{near}, x_{new})$ then
16. $x_{parent}) \leftarrow Parent(x_{near})$
17. If ObstacleFree($x_{nearest}, x_{free}$) then
18. $E' \leftarrow E' \left\{(x_{parent}, x_{near})\right\}$
19. $E' \leftarrow E' \cup \{(x_{new}, x_{near}), (x_{new}, x_{nearest})\}$
20. return G'=(V', E')

in [115] to derive these two heuristics for RRT* by locally linearizing the domain dynamics and applying linear quadratic regulator.

Guided Expansive Search Trees: GEST

Guided Expansive Search Trees is a variation of conventional probabilistic path planning strategies: PRM, RRT and most similarly Expansive Search Trees (EST). Guided Expansive Search Trees have advantages over the conventional techniques when the state space is governed by higher dimensions, Kinodynamic constraints and when minimizing the control cost of the entire path is important. Guided Expansive Search Trees borrow from conventional expansion techniques for robustness in finding paths and also use path cost statistics to guide the tree in the window of acceptably low-cost paths [118]. Conventional probabilistic path planning algorithm rely on a metric to determine whether two configurations are 'close'. But in kinodynamic state spaces, it is not obvious when two configurations should be considered 'close'. These techniques do not necessarily accurately represent the reachability of the configuration. For instance, two configurations which are close based on a 1-norm will likely not be able to reach one another with a reasonable cost.

The pseudo-code for this GEST method is given in Algorithm 22 in page 138.

The weighting function of the Expansive Search Trees algorithm only looks at the number of neighbors n_{neigh} within a range $weight = \frac{1}{n_{neigh}}$. The guided expansive search tree algorithm differs in that it additionally takes into account the out-degree, number of out-going edges from the way point; the order of the way point; how

Algorithm 22 Guided Expansive Search Trees Algorithm

1. For i=0 to N do
2. p = choose-waypoint ()
3. n = expand-waypoint (p)
4. (if n is not valid) continue
5. add-to-tree (p,n)
6. assign-weigh (n)
7. if n connects to goal: return add-to-tree (n, goal)
8. end for.

recently it was created and the A* cost, estimated total cost to the goal computed as the sum of the control required to reach the way point from the root and the estimated control cost to reach the goal. These statistics are taken to the power $\alpha, \beta, \gamma, \delta$ respectively:

$$weight = \frac{(order)^{\gamma}}{n_{neigh}^{\alpha}(out - degree)^{\beta}(A * cost)^{\delta}} \tag{2.246}$$

The out-degree term prevents a highly weighted waypoints from being expanded too many times. This is incremented even if the expanded to waypoint is not valid. The A* cost term focuses the search towards the goal and prevents the tree from often expanding high-cost waypoints which violate a velocity or rate constraints. The order term, like the number of neighbors term, tends to keep the tree expanding on the frontier. The purpose of using the number of neighboring waypoints in the weighting function is to bias the search towards expanding on the frontier. So, whether two waypoints are close should be defined based on the control cost between the two configurations. This indicate how likely one way-point is to expand into the region of the other.

Randomized motion planning techniques such as Probabilistic road map (PRM), Rapidly exploring Random Trees (RRT), rapidly exploring random graphs (RRG) or guided expansive search trees (GEST) have been successful in dealing with the high dimensional configuration space arising in many real world applications. However, they rely on accurate models of the environment. Missiuro in [105] proposed an extension of PRM that computes motion plans that are robust to environment uncertainty by incorporating uncertainty in PRM sampling as well as modeling the cost of collision in traveling through uncertain regions of the map. In the aforementioned approaches, the uncertainty area of the map is globally known and the aerial robot has to determine a plan to reach the goal as safely as possible.

2.5.3 Continuous Search Methods

2.5.3.1 Reactive Planning

The term **reactive planning** refers in general to a broad class of algorithms that use only local knowledge of the obstacle field to plan the trajectory. Reactive algorithms are important in dealing with uncertainty, and run very quickly since no elaborate couplings are involved. In the case where a global obstacle map is not available and obstacle positions are known within a small radius, a reactive algorithm prevents last-minute collisions by swerving the aerial robot when an obstacle is known to be in the trajectory, which has been planned by a different algorithm. This type of approach is important in many existing practical implementations in order to 'patch' an unsound algorithm to ensure that it is sound, as well as to deal with obstacle fields that may change suddenly. However, reactive planners, due to their inability to take the global planning problem into consideration, are seldom used as the sole trajectory generation process. If only the reactive planner is used, the aerial robot may never find a trajectory that will lead to the goal.

Algorithm 23 Bug Algorithm

1. Visualize a direct path from the start S to the goal G.
2. While the goal G is not achieved, do:
3. while the path SG to the goal is not obstructed, do:
4. Move towards the goal along the path SG
5. If the path is obstructed then
6. Mark the current location as P circumnavigates the object until the aerial robot either
7. (1) hits the line SG at a point closer to G than P and can move towards G, in which case the robot follows SG
8. (2) returns to where P in which case G is unreachable

The pseudo-code of the Bug method given in Algorithm 23 in page 139 is guaranteed to find a path to the goal location if it is accessible. The start configuration of the aerial robot is termed S and its goal is G. Then, if a direct straight line path exists in free space from S to G, the aerial robot should use this path. If the path is obstructed, then the robot should move along the straight line from S to G until it encounters the obstacle. This point is termed P. Then the aerial robot should circumnavigate the obstacle until it continues to move to the goal location G along the line SG. It circumnavigates the obstacle until it finds a point on the obstacle on the line SG which it can leave the obstacle in the direction of G that is closer to G than the point P at which it started circumnavigating the obstacle. If no such point is found, the robot determines that no path exists from S to G.

2.5.3.2 Artificial Potential Methods

In many applications such as surveillance and monitoring, the aerial robot must be able to plan its motion on-line, i.e. using partial information on the workspace gathered during the motion on the basis of sensor measurements. An effective method for on-line planning relies on the use of artificial potential fields. Essentially, the point that represents the aerial robot in configuration space moves under the influence of a potential field U obtained as the superposition of an attractive potential to the goal and a repulsive potential from the C obstacle region. Planning takes place in an incremental fashion: at each aerial robot configuration Q, the artificial force generated by the potential is defined as the negative gradient $-\nabla U(Q)$ of the potential, which indicates the most promising direction of local motion [93, 156].

Since their initial publication [77], potential field methods have been generally known for being of low computational complexity but incomplete. However, a potential field which has the properties of a navigation function makes a complete path planner.

Definition 2.7. Navigation function A function Φ: $Q_{free} \rightarrow [0, 1]$ is called a navigation function if it

- is smooth (or at least C^k for $k \geq 2$)
- has a unique minimum at Q_g in the connected component of the free space that contains Q_g
- is uniformly maximal on the boundary of the free space
- is Morse (A Morse function is one whose critical points are all non degenerate, critical points are isolated).

The value of a potential function can be viewed as energy and hence the gradient of the potential is a force, which points in the direction that locally maximally increases the potential. The combination of repulsive and attractive forces should direct the aerial robot from the start location to the goal location while avoiding obstacles. The aerial robot terminates motion when it reaches a point where the gradient vanishes. Such a point is called a critical point. When the Hessian is non singular, the critical point is non degenerate, isolated. This section is based mainly on [40, 78, 135].

Attractive Potential

The attractive potential is designed so as to guide the aerial robot to the goal configuration Q_g. To this end, a paraboloid function may be used:

$$U_{a1}(Q) = \frac{1}{2} k_a \|e(Q)\|^2 \tag{2.247}$$

where $k_a > 0$ and $e = Q_g - Q$ is the error vector with respect to the goal configuration Q_g. This function is always positive and has a global minimum in Q_g, where it is

zero. The resulting attractive force is defined as:

$$f_{a1}(Q) = -\nabla U_{a1}(Q) = -k_a e(Q) \tag{2.248}$$

Hence, f_{a1} converges linearly to zero when the aerial robot configuration q tends to the goal configuration Q_g.

Alternatively, a conical attractive potential may be defined as:

$$U_{a2}(Q) = k_a \|e(Q)\| \tag{2.249}$$

Also, U_{a2} is always positive and zero in Q_g. The corresponding attractive force is

$$f_{a2}(Q) = -\nabla U_{a2}(Q) = -k_a \frac{e(Q)}{\|e(Q)\|} \tag{2.250}$$

that is constant in modulus. This represents an advantage with respect to the force f_{a1} generated by the paraboloid attractive potential, which tends to grow indefinitely as the error vector increases in norm. On the other hand, f_{a2} is indefinite in Q_g. A choice that combines the advantages of the above two potentials is to define the attractive potential as a conical surface away from the goal and as a paraboloid in the vicinity where $\|e(Q)\| = 1$ (i.e. on the surface of the sphere of unit radius centered in Q_g), an attractive force is obtained that is continuous for any Q.

Repulsive Potential

The repulsive potential U_r is added to the attractive potential U_a to prevent the aerial robot from colliding with obstacles as it moves under the influence of the attractive force f_a. In particular, the idea is to build a barrier potential in the vicinity of the C obstacle region, so as to repel the point that represents the aerial robot. In the following, the C obstacle region has been partitioned in convex components $CO_i, i = 1 \ldots p$. These components may coincide with the C obstacles themselves. In the presence of non convex C obstacles, it is necessary to perform the decomposition in convex components before building the repulsive potential. For each convex component CO_i, an associated repulsive potential is defined as:

$$U_{r,i} = \begin{cases} 0 & \text{if } \eta_i(Q) > \eta_{0,i} \\ \frac{k_{r,i}}{2}\left(\frac{1}{\eta_i(Q)} - \frac{1}{\eta_{0,i}}\right)^2 & \text{if } \eta_i(Q) \leq \eta_{0,i} \end{cases} \tag{2.251}$$

where $\eta_{0,i}$ is the range of influence of CO_i, $k_{r,i} > 0$, $\eta_i(q)$ is the distance of q from CO_i

$$\eta_i(Q) = min_{Q' \in CO_i} \left\| Q - Q' \right\| \tag{2.252}$$

The potential $U_{r,i}$ is zero outside CO_i and positive inside the range of influence $\eta_{0,i}$ and tends to infinity as the boundary of CO_i is approached.

- When $C = \mathbb{R}^2$ and the convex component CO_i is polygonal, an equipotential contour of $U_{r,i}$ (i.e. the locus of configurations q such that $U_{r,i}$ has a certain constant value) consists of rectilinear tracts that are parallel to the sides of the polygon, connected by arcs of circle in correspondence of the vertices. The contours get closer to each other in the proximity of the C obstacle boundary, due to the hyperboloidic profile of the potential.
- When $C = \mathbb{R}^3$ and the convex component CO_i is polyhedral, the equipotential surfaces of $U_{r,i}$ are copies of the faces of CO_i, connected by patches of cylindrical surfaces in correspondence of the edges and spherical surfaces in correspondence of the vertices of CO_i.

The repulsive force resulting from $U_{r,i}$ is:

$$f_{r,i} = -\nabla U_{r,i}(Q) = \left\{ \begin{array}{ll} 0 & \text{if } \eta_i(Q) > \eta_{0,i} \\ \frac{k_{r,i}}{\eta_i^2(Q)} \left(\frac{1}{\eta_i(Q)} - \frac{1}{\eta_{0,i}}(Q) \right) \nabla \eta_i(Q) & \text{if } \eta_i(Q) \leq \eta_{0,i} \end{array} \right\} \tag{2.253}$$

Denote by Q_m the configuration of CO_i that is closer to Q (Q_m is uniquely determined in view of the convexity of CO_i). The gradient vector $\nabla \eta_i(Q)$ is orthogonal to the equi-potential contour (or surface) passing through q. If the boundary of CO_i is piecewise differentiable, the function η_i is differentiable everywhere in C_{free} and $f_{r,i}$ is continuous in the same space. The aggregate repulsive potential is obtained for the *on* obstacles by:

$$U_r(Q) = \sum_{i=1}^{on} U_{r,i}(Q) \tag{2.254}$$

If $\eta_i(Q_g) > \eta_{0,i}$ for $i = 1 \ldots on$ (i.e. if the goal is placed outside the range of influence of each obstacle component CO_i), the value of the aggregate repulsive field U_r is zero in Q_g. In the following, it will be assumed that this is the case.

Total Potential

The total potential U_r is obtained by adding the attractive and the aggregate repulsive potentials:

$$U_t(Q) = U_a(Q) + U_r(Q) \tag{2.255}$$

This results in the force field

$$f_t(Q) = -\nabla U_t(Q) = f_a(Q) + \sum f_{r,i}(Q) \tag{2.256}$$

$U_t(Q)$ clearly has a global minimum in q_g, but there may also exist some local minima where the force field is zero. This happens in the 'shadow zone' when the

repulsive potential $U_{r,i}$ has equi-potential contours with lower curvature than the attractive potential in the same area.

Planning Techniques

Different approaches for planning collision-free motions are briefly discussed below [135]:

1. The first possibility is to let
$$\tau = f_t \tag{2.257}$$
hence considering $f_t(q)$ as a vector of generalized forces that induce a motion of the aerial robot in accordance with its dynamic model.
2. The second method regards the aerial robot as a unit point mass moving under the influence of $f_t(q)$ as in
$$\ddot{Q} = f_t \tag{2.258}$$
3. The third possibility is to interpret the force field $f_t(q)$ as a desired velocity for the aerial robot by letting
$$\dot{Q} = f_t \tag{2.259}$$

In principle, one could use these three approaches for on-line as well as off-line motion planning. In the first case, Eq. (2.257) directly represents control inputs for the aerial robot, whereas the implementation of Eq. (2.258) requires the resolution of the inverse dynamics problem. Equation (2.259) can instead be used on-line in a kinematic control scheme, in particular to provide the reference inputs for the low-level controllers that are in charge of reproducing such generalized velocities as accurately as possible. In any case, the artificial force field f_t represents, either directly or indirectly, a true feedback control that guides the aerial robot towards the goal, that have been detected by the sensory system. To emphasize this aspect, on-line motion generation based on artificial potentials is also referred to as reactive planning.

In off-line motion planning, configuration space paths are generated by simulation. In general, the use of Eq. (2.257), generates smoother paths, because with this scheme, the reactions to the presence of obstacles are naturally filtered through the aerial robot dynamics. On the other hand, the strategy represented by Eq. (2.259) is faster in executing the motion corrections suggested by the force field f_t and thus be considered safer. The characteristics in Eq. (2.258) are intermediate between the other two. Another aspect to be considered is that using Eq. (2.259) guarantees (in the absence of local minima) the asymptotic stability of Q_g (i.e. the aerial robot reaches the goal with zero velocity) whereas this is not true for the other two motion generation strategies. To achieve asymptotic stability with Eqs. (2.257), (2.258), a damping term proportional to the aerial robot velocity $\dot{Q}$ must be added to f_t.

The most common choice is the simple numerical integration of Eq. (2.259) via the Euler method:

$$Q_{k+1} = Q_k + T f_t(Q) \tag{2.260}$$

where Q_{k+1}, Q_k represent respectively the current and the next aerial robot configuration, and T is the integration step. To improve the quality of the generated path, it is also possible to use a temporal variable T, smaller when the modulus of the force field f_t is larger in the vicinity of obstacles) or larger (close to the destination Q_g). Equation (2.260) may be interpreted as a numerical implementation of the gradient method for the minimization of $U_t(Q)$, often referred to as the algorithm of steepest descent.

Local Minima Problem

Although the methodology is appealing due to its intuitive nature and computationally efficient implementation (controls are typically available in analytic form) there is often no guarantee that local minima are not present which may trap the aerial robot in some configuration other than the desired one. The possibility of the existence of local minima in the artificial potential field could be one of the drawbacks of the potential field method. A local minimum can attract and trap the aerial robot, preventing it from reaching its final goal. This problem can be overcome by generating the potential field as a numerical solution to the Laplace equation or by various heuristics such as adding noise to escape from any local minima.

There are two classes of potential fields known to satisfy properties of a navigation function: those based on a harmonic function and those based on solving the optimal distance-to-go function. These methods require, however, discretizing the configuration space into a grid with M points. The added advantage of a navigation function is that it can be used to provide direct feedback control, rather than relying on feed-forward control, as traditional trajectory planners do. A single navigation function produces a trajectory for every possible starting point in the configuration space. Rather than use gradient descent, which is easily trapped in local minima, a search that is complete in the resolution or probabilistic sense is used. This can be considered as being similar to an A* search with the simple heuristic replaced by a potential field. The variational planning approach uses the potential as a cost functional and attempts to find a path to the goal point that minimizes this cost. The artificial potential field methodology is based on the assumption of the existence of a virtual potential field which attracts the aerial robot towards a goal, while repelling it away from obstacles and other flying vehicles.

Search methods have been introduced to address this problem of local minima at a high computational cost. One method for avoiding the generation of local minima is adding multiple auxiliary attraction potentials, whose positions are determined by a genetic algorithm. Also a set of analytical guidelines have been given for designing potential functions to avoid local minima for a number of representative scenarios. Another approach is based on the use of navigation functions, i.e. artificial potentials

that have no local minima. A way to define a navigation function consists of building first a diffeomorphism that maps the C obstacle region to a collection of spheres, then generating a classical total potential in the transformed space, and finally mapping it back to the original configuration space so as to obtain a potential free of local minima. If the C obstacles are star-shaped, such a diffeomorphism actually exists, and the procedure outlined above provides in fact a navigation function. Another possibility is to build the potential using harmonic functions, that are the solutions of a particular differential approach that describes the physical process of heat transmission or fluid dynamics.It has been shown that harmonic potential do not suffer from local minima and lead to unique solutions. This property of harmonic potential functions allows the potential to be defined in Euclidean space rather than in the configuration space. One important limitation from the point of view of application to aerial robot collision avoidance is that the non holonomic motion constraints might prevent the agent from being able to move immediately in the direction of the resultant force. The following heuristic can be used to alleviate the problem: Positioning is decomposed into fore-aft corrections (done by adjusting speed only) and side-side corrections (done by adjusting heading only) which can be applied independently [147]. Another limitation is that it is difficult to fully consider wide range of aircraft dynamics when they are applied to aircraft collision avoidance. Large virtual forces are necessary for repelling fast incoming traffic, but with slower intruders, this will cause unnecessary deviation from planned flight trajectories.

2.5.3.3 Geometric Optimization

Mathematical programming methods treat the trajectory planning problem as a numerical optimization problem [111, 143]. Some popular methods include Mixed Integer Linear Programming (MILP), non linear programming and other constrained optimization approaches. These methods are also known as trajectory optimization methods, since they find a trajectory to a goal point that is optimal in the resolution sense. However, the cost functions typically have a number of local minima thus finding the global solution strongly depends on the initial guess (the general formulation is NP hard, although given an initial guess sufficiently close to the global solution, the optimization converges in polynomial time). For this type of problem, one standard strategy is to enforce the equations of motion as constraints. Another strategy is to discretize the variational principles underlying the systems dynamics, such as Hamilton's principle or Lagrange-D'Alembert principles, and then these discrete equations can serve as constraints. This kind of strategy is called Discrete Mechanics and Optimal Control (DMOC). Several approaches have been used to break this into simpler problems. A 2D conflict resolution algorithm in horizontal plane using geometric computations was introduced in [16]. Conflict predictions are based on straight line projections using positions and assuming constant velocities. Computed resolutions consist of minimal changes in velocity to avoid a predefined circular protected airspace around intruder aircraft. Geometric solutions to collision avoidance have the unique advantage of being extremely fast and easily verifiable

but precautions such as adjusting the protected airspace sizes and breaking the constant velocity assumptions should be taken in order to account for uncertainties in sensing and intruder intent, and unexpected intruder dynamics. An initial trajectory for the mathematic programming methods, such as a constant-speed trajectory, is then used as an initial point in the mathematical programming search. If this initial point falls within the basin of attraction of the global solution, then the mathematical programming approach can find the optimal solution in polynomial time. However, unless care is taken in finding proper initial points, the solution could fall into a local minimum, and general global optimization approaches guaranteed to find the global minimum are prohibitively expensive.

For trajectory planning in a geographical environment, the cost to go function must account for the effects of the environment. The most common technique used so far to compute cost to go function for receding horizon trajectory planning is based on visibility graph decompositions. The graph is constructed based on the goal location and a polygonal obstacle configuration. Cost to go values for the graph's vertices can be computed using a shortest path algorithm. In a minimum time problem, the cost to go is determined heuristically based on the edge length and a characteristic aerial robot speed. A method based on multi-resolution of the 3D environment can also be proposed. Heuristics are then used to incorporate aerial robot state information: aerial robot state is related to the cell's dimension via the minimum turn radius and climb rate based on vertical cell. The optimal heading was computed from the direction of the cost to go gradient. Terminal constraints (heading and speed at the goal) and simple maneuvering limitations have been handled for planar problems using shortest path computations constrained to kino-dynamically feasible path. Most heuristic cost to go are limited to distance or pseudo-duration. A more recent approach based on the wavelets has been proposed in [148]. The algorithm uses the wavelet transform to construct an approximation of the environment at different levels of resolution. A graph whose dimension is commensurate to the on board computational resources of the aerial robot is associated with this multi resolution representation of the environment. The adjacency list of the graph can be efficiently constructed directly [4].

Mixed Integer Linear Programming: MILP

Aerial robot trajectory optimization including collision avoidance can be expressed as a list of linear constraints, involving integer and continuous variables known as mixed integer-linear program. The MILP mixed integer linear programming approach in [125] uses indirect branch-and-bound optimization, reformulating the problem in a linearized form and using commercial software to solve the MILP problem. A single aircraft collision avoidance application was demonstrated then this approach was generalized to allow for visiting a set of waypoints in a given order. Mixed Integer Linear Programming can extend continuous linear programming to include binary or integer decision variables to encode logical constraints and discrete decisions together with the continuous aerial robot dynamics. The approach to optimal path

planning based on MILP was introduced in [13, 91, 132] for robotic helicopters. The aerial robot trajectory generation is formulated as a 3D optimization problem under certain conditions in the Euclidean space, characterized by a set of decision variables, a set of constraints and the objective function. The decision variables are the aerial robot state variables, i.e. position and speed. The constraints are derived from a simplified model of the aerial robot and its environment. These constraints include:

- Dynamics constraints, such as a maximum turning force which causes a minimum turning radius, as well as a maximum climbing rate.
- Obstacles avoidance constraints like no-flight zones
- Target reaching constraints of a specific way point or target.

The objective function includes different measures of the quality in the solution of this problem, although the most important criterion is the minimization of the total flying time to reach the target.

As MILP can be considered as a geometric optimization approach, there is usually a protected airspace set up around the aerial robot in the MILP formulation. The stochasticity that stems from uncertainties in observations and unexpected aircraft dynamics could be handled by increasing the size of protected airspaces. An advantage of the MILP formulation is its ability to plan with non-uniform timesteps between waypoints. A disadvantage of this approach is that it requires all aspects of the problem (dynamics, ordering of all waypoints in time and collision avoidance geometry) to be specified as a carefully designed and a usually long list of many linear constraints, and then the solver's taks is basically to find a solution that satisfies all of those constraints simultaneously [147].

Conditions for infinite horizon safety in the presence of multiple unpredictable obstacle have seldom been directly addressed in the literature. Using the original formulation of the velocity obstacle, one can find single velocity trajectories that are guaranteed collision-free, given the exact trajectories of the obstacles for some time scale. This time-scale could be infinite but that would unrealistically require that all obstacles trajectories be perfectly known for all future time. In [157], a method is presented for finding velocity space constraints for a motion planner that guarantees infinite horizon safety of a host disc robot in an unbounded environment with multiple disc obstacles that have unicycle dynamics but can move unpredictably. The obstacles current poses are observed but no explicit predictions about their future trajectories are made. The safety guarantee is achieved by combining the obstacle's **reachable sets** as functions of time to their dynamics with the velocity obstacle concept.

Reach-Avoid Games

As a branch of applied mathematics, the game theory has been applied widely to different areas. Game theory is a decision making method between players who interact with and depend on each other. In Game theory, the individuals or teams chose possible strategies synchronously or successively achieve high payoff income according

to the known information in certain circumstances or constrained conditions [150]. A decision making process is called a game if there are more than one decision makers pursuing their own profits at the same time. The game theory analyses the conflict and cooperation problems between rational decision-makers. It has five factors:

1. player
2. strategy
3. decision-making order
4. payoff function
5. information

A reach-avoid game is one in which an agent attempts to reach a predefined goal, while avoiding some adversarial circumstance induced by an opposing agent or disturbance. Reach-Avoid Games analysis plays an important role in safe motion planning and obstacle avoidance, yet computing solutions is often computationally expensive due to the need to consider adversarial input. In [160], an open-loop formulation of a two player reach avoid game is presented whereby the players define their control inputs prior to the start of the game. Two open-loop games are defined, each is conservative towards one player and the solutions to these games are related to the optimal feedback strategy for the closed loop games.

2.6 Replanning Approaches

Planning for aerial robots in the real world involves dealing with a number of challenges not faced in many simpler domains. Firstly, the real world is an inherently uncertain and dynamic place; accurate models for planning are difficult to obtain and quickly becomes out of date. Secondly, when operating in the real world, time for deliberation is usually very limited. Aerial robots need to make decisions and act upon these decisions quickly. To cope with imperfect information and dynamic environments, efficient replanning algorithms have been developed that correct previous solutions based on updated information. These algorithms maintain optimal solutions for a fraction of the computation required to generate such solutions from scratch [89]. In [124], a dynamical reference generator equiped with an augmented transient replanning subsystem that modulates a feedback controller's effort to force a mechanical plant to track the reference signal is presented. The replanner alters the reference generator's output in the face of unanticipated disturbances that drive up the tracking error. The new reference generator cannot destabilize the tracker. Tracking errors convergence in the absence of disturbances and the overall coupled reference-tracker system cannot be destabilized by disturbances of bounded energy. Often the available information of an aerial robot is imperfect or incomplete. As a result any solution generated using its initial information may turn out to be invalid or suboptimal as it receives updated information through onboard and off board sensors. It is thus important that the aerial robot is able to replan optimal paths, when new

information arrives. A number of algorithms exist for performing this replanning. Two main types exist: incremental and anytime algorithms.

2.6.1 Incremental Replanning

Many applications require the aerial robot to have sufficient on board situational awareness to avoid collision with unanticipated obstacles in the immediate environment while fulfilling the global planning requirements. This can be achieved by environment sensing, mapping and fast replanning in real time. To accommodate plan updates in partially known or unknown environments, mainly incremental graph search algorithm have been proposed. An incremental algorithm can be proposed for a generalization of the shortest path problem in a graph with an arbitrary edge insertion, edge deletion and edge length changes.

2.6.1.1 D* Algorithm

Rather than require all obstacles to be static, the obstacles are permitted to move. If the motion of the objects is known a priori, D* (Dynamic A*) is an extension of the A* algorithm to enable efficient path replanning as new information becomes available. D* produces an initial plan using A* based on the information known initially. As the plan is executed, discrepancies between the modeled and sensed world update the environment map and the plan is repaired. D* has been shown to be optimal and complete. In addition, the process of repairing the plan can be considerably more efficient than replanning from scratch.

In dynamic environments, there are three types of dynamic obstacles.

- those that move significantly slower than the aerial robot,
- those that move at the same speed,
- those that move much faster than the aerial robot.

The super-fast obstacle case is easy to ignore because the obstacles will be moving so fast, that there probably is no need to plan for them because they will either move too fast for the planner or they will be in and out of the aerial robot's path so quickly that it does not require any consideration. In this paragraph, dynamic environments where the world change at a speed much slower than the aerial robot, are considered. The A* algorithm can be run to determine a path from start to goal and then follow that path until an unexpected change occurs. The D* algorithm is devised to locally repair the graph allowing efficient updated searching in dynamic environments, hence the term D*. D* initially determines a path starting with the goal and working back to the start using a slightly modified Dijkstra's search as shown in algorithm 24 in page 150. The modification involves updating a heuristic function. Each cell contains a heuristic cost h which for D* is an estimate of path length from the particular cell to the goal, not necessarily the shortest path length to the goal as it was for A*. These

h values will be updated during the initial Dijkstra search to reflect the existence of obstacles. The minimum heuristic values h are the estimate of the shortest path length to the goal. Both the h and the heuristic values will vary as the D* search runs, but they are equal upon initialization [24].

Algorithm 24 D* Algorithms

1. **Input**: List of all states L
2. **Output**: The goal state, if it is reachable, and the list of states L are updated so that back pointer list describes a path from the start to the goal. If the goal state is not reachable, **return NULL**
3. For each $X \in L$ do
4. $t(X)$=New
5. $endfor$
6. $h(G) = 0; 0 = \{G\}; X_c = S$
7. The following loop is Dijkstra's search for an initial path.
8. repeat
9. k_{min} = process-state (0, L)
10. until ($k_{min} > h(x_c)$) or ($k_{min} = -1$)
11. P=Get-Pointer-list (L, X_c, G)
12. If P = Null then
13. return (Null)
14. end if
15. end repeat
16. $endfor$
17. X_c is the second element of P Move to the next state in P
18. P= Get-Back-Pointer-List(L, X_c, G)
19. until X_c=G
20. return (X_c)

Notation

1. X represents a state
2. O is the priority queue
3. L is the list of all states
4. S is the start state
5. t(x) is the value of state with respect to priority queue
   a. t(x): New if x has never been in O
   b. t(x): Open if x is currently in O
   c. t(x): Closed if x was in O but currently is not

Pseudo-code of the D* method is presented in Algorithm 24 in page 150.

C(X,Y) is the estimated path length between adjacent states X, Y. h(X) is the estimated cost of a path from X to Goal (heuristic). k(X) is the estimated cost of a shortest path from X to Goal (minimum heuristic = $minh(X)$ before X is put on O, values h(X) takes after X is put on O). $b(X) = Y$ is the measured distance adjacent states with X, Y.

2.6.1.2 D* Lite Algorithm

Dynamic A* (D*) and D* Lite are currently used, due to their efficient use of heuristics and incremental updates. Both algorithms guarantee optimal paths over graphs [56, 59, 89]. In this approach, rather than recalculating the optimal path for the entire map when changes in the map are detected, a reduced set of cells are checked and the optimal path to the aerial robot pose is updated incrementally, in partially known environment. Focused $D*$ focuses the repairs using heuristics to reduce the total time for replanning. The method $D*$ lite implements the same navigation strategy but is algorithmically different from Focused $D*$. These incremental planners which make use of the results of the previous plans to generate a new plan, can substantially speed up the planning cycles. Both D* and D* Lite maintain least-cost paths between a start state and any number of goal states as the cost of arcs between state change. Both algorithms can handle increasing or decreasing arc costs and dynamic start states. They are both suited to solving the goal-directed aerial robot navigation problem, which entails moving from some initial state to one of a number of goal states while updating its map information through an onboard/offboard sensors. D* Lite maintains a least-cost path from a start state $s_{start} \in S$ to a goal state $s_{goal} \in S$, where S is the set of states in some finite state space. To do this, it stores an estimate g(s) of the cost from each state s to the goal. It also stores a one-step ahead cost $rhs(s)$ which satisfies

$$rhs(s) = \left\{ \begin{array}{ll} 0 & \text{if } s = s_{goal} \\ min_{s' \in Succ(s)} \left(c(s, s') + g(s') \right) & \text{otherwise} \end{array} \right\} \tag{2.261}$$

where $Succ(s) \subset S$ denotes the set of successors of s and $c(s, s')$ denotes the cost of moving from s to s', the arc cost [121].

Definition 2.8. Consistency A state is called consistent if and only if its g-value equals its rhs-value, otherwise it is either overconsistent if $g(s) > rhs(s)$ or underconsistent if $g(s) < rhs(s)$.

As with A*, D* lite uses a heuristic and a priority queue to focus its search and to order its cost updates efficiently. The heuristic $h(s, s')$ estimates the cost of an optimal path from state s to s' and needs to satisfy $h(s, s') \leq c^*(s, s')$ and $h(s, s'') \leq h(s, s') + c^*(s', s'')$ for all states $s, s', s'' \in S$ where $c^*(s, s')$ is the cost associated with a least cost path from s to s'. The priority queue OPEN always holds exactly the inconsistent states, states that need to be updated and made consistent. The priority, or keyvalue, of a state s in the queue is

$$\begin{array}{c} key(s) = [k_1(s), k_2(s)] = \\ = [min\left(g(s), rhs(s) + h(s_{start}, s)\right), min(g(s) + rhs(s))] \end{array} \tag{2.262}$$

A lexicographic ordering is used on the priorities so that priority $key(s)$ is less than priority $key(s')$ denoted $key(s) < key(s')$ if and only if $k_1(s) < k_1(s')$ or

both $k_1(s) = k_1(s')$ and $k_2(s) \leq k_2(s')$. D* Lite expands states from the queue in increasing priority, updating their g-values and the rhs-values of their predecessors until there is no state in the queue with a key value less than of the start state. Thus during its generation of an initial solution path, it performs in exactly the same manner as a backwards A* search.

If arc costs change after this initial solution has been generated, D^* Lite updates the rhs-values of each state immediately affected by the changed arc costs and places those states that have been made inconsistent onto the queue. As before, it then expands the states on the queue in order of increasing priority until there is no state in the queue with a key value less than that of the start state. By incorporating the value $k_2(s)$ into the priority for state s, D* Lite ensures that states that are along the current path and on the queue are processed in the most efficient order. Combined with the termination condition, this ordering also ensures that a least-cost path will have been found from the start state to the goal state when processing is finished. The basic version of the Algorithm 25, is given in page 152.

Algorithm 25 D* Lite Algorithm

1. **Key (s)**
2. return $[min\,(g(s), rhs(s) + h(s_{start}, s))\,, min(g(s) + rhs(s))]$
3. **UpdateState(s)**
4. If s was not visited before
5. $g(s) = \infty$
6. if $(s \neq s_{goal}), rhs(s) = min_{s' \in Succ(s)}\left(g(s') + c(s, s')\right)$
7. if $s \in$ OPEN, remove s from OPEN
8. if $(g(s) \neq rhs(s))$ insert s into OPEN with $key(s)$
9. **ComputeShortestPath()**
10. while $(min_{s \in > OPEN}(keys(s_{start}$ OR $rhs(s_{start}) \neq g(s_{start}))$
11. remove state s with the minimum key from OPEN
12. if g(s) > rhs(s)
13. g(s) = rhs(s)
14. for all $s' \in Pred(s)$ UpdateState(s')
15. **Main**
16. $g(s_{start}) = rhs(s_{start}) = \infty$; $g(s_{goal}) = \infty$
17. $rhs(s_{goal}) = 0$; $OPEN = \emptyset$
18. insert s_{goal} into OPEN with $key(s_{goal})$
19. forever
20. ComputeShortestPath()
21. wait for changes in edge costs
22. for all directed edges (u, v) with changed costs
23. Update the edge cost c(u,v)
24. UpdateState(u);

D* Lite is efficient because it uses a heuristic to restrict attention to only those states that could possibly be relevant to repairing the current solution path from a given start state to the goal state. When arc costs decrease, the incorporation of the

heuristic in the key value k_1 ensures that only those newly-consistent states that could potentially invalidate the current cost of the start state are processed.

2.6.1.3 Replanning with 3D A* Algorithm

The A* method is a graph-based algorithm for the aerial robot's trajectory generation. This algorithm can be used when the environment is filled with different kind of obstacles, their motion is not known in advance and the manoeuvres have to be executed nearby them. The computational cost in 3D path planning problems can be lowered with an appropriate scaling. The A* algorithm can be modified to adapt and improve its capabilities to optimize 3D trajectories under events such as no-flight zones, turbulence, storm clouds etc., which might appear during the flight. Some modifications done to the original A* algorithm are restrictions to the successors of every possible node taking into account a maximum heading rate and flight path angle limitations while avoiding physical obstacles and atmospheric threats. The motion of the obstacles is not known a priori. The easiest way is to use the data from the coordinates of the nodes and apply a tree-point moving average filter [57]. The aerial robot follows the route with imprecise sensor data or external disturbances which could be very significant. Many constraints have to be considered in the path planning: geometric, kinematic, dynamic. In the beginning, the representation of the geometrical limitations of the aerial robot has to take place in the formation of the cost maps. In the initialization of the algorithm search space, each waypoint, possible solution, is chosen to avoid impossible aerial robot's motions. The allowable connections between the grid points are designed to capture the aircraft's kinematics. The intervals are planned as a function of R_{min} (minimum turning radius) and R_{max} (maximum flight path angle between two consecutive nodes). R_{min} and R_{max} are the most important kinematics motion constraints. By satisfying these constraints, the motion of the aerial robot stays within its maximum acceleration bounds. Horizontally, the minimum turning radius corresponds to a maximum yaw rate turn (curvature) and vertically, the maximum flight path angle is equivalent to a maximal roll rate turn (torsion). These constraints are equivalent to load factor and payload limitations [34]. The generation of successor points in the neighborhood has to allow a sufficient flexibility in the trajectory planning. The range of current configuration is an integer parameter named **depth** that represents the number of points in the neighborhood from which successors are selected. In three-dimensional space with 26 moving styles the depth is one, because there is only one point in each direction. A two-component planning architecture with a coarse global planner and a fine local planner is proposed. The global planner takes into account kinematic constraints of the aircraft but neglects the dynamic constraints considered in the local planner. The local planner also called path-finder selects a sub goal from the optimal path found by the global planner over its workspace. Both the global and local planners work in 3D grid cells.

2.6.1.4 Generalized Voronoi Diagram

The Voronoi region of an obstacle O is the set of points whose distance to O is less or equal to their distance to every other obstacle in the environment. The **Generalized Voronoi Diagram** (GVD) of an environment is the intersection of two or more Voronoi regions. Each of these intersection points is equidistant from its two (or more) closest obstacles. Several methods exist for computing the Generalized Voronoi Diagram, in continous or discrete space. Here, the voronoi regions and the GVD are computed over the finite set of grid cells. Generalized Voronoi Diagrams are very useful for extracting environment topologies [71]. Generalized Voronoi Diagrams are roadmaps that provide all possible path homotopies in an environment containing obstacle regions. They provide maximum clearance from these regions. Such a representation has practical applications to surveillance and area coverage. Given a Generalized Voronoi Diagram, planning from a start position to a goal position consists of three steps:

1. Plan from the start to its closest point on the GVD (the access point).
2. Plan along the GVD until the point closest to the goal is reached (the departure point).
3. Plan from the departure point to the goal

Since the Generalized Voronoi Diagram (GVD) is a graph any graph search algorithm can be used to plan between the access and the departure points, often at a fraction of the computational expense required to plan over the complete environment.

In environments for which prior information is unavailable, incomplete or erroneous, the aerial robot must update its map when it receives sensor information during execution, reconstruct the Generalized Voronoi Diagram and generate a new plan. The Generalized Voronoi Diagram reconstruction and replanning must occur frequently because new information is received almost continually. However, because new information is usually localized around the aerial robot, much of the previous Generalized Voronoi Diagram remains correct and only portions require repair.

The **Dynamic Brushfire algorithm** is presented in this section for incremental reconstruction of the Generalized Voronoi Diagram on grids. Brushfire algorithm is analogous to Dijkstra algorithm in that it processes cells in an OPEN priority queue, where decreasing priority maps to increasing distance from an obstacle. Initially, each obstacle cell in the environment is placed on the queue with a priority of 0 (the cell's distance to the nearest obstacle). Then, until the queue is empty, the highest priority cell is removed, its neighboring cell's distances are computed, and any cell c whose distance $dist_c$ of each cell is approximated from the distances of its neighbors:

$$dist_c = min_{a \in Adj(c)} \left[distance(c, a) + dist_a \right] \tag{2.263}$$

where $Adj(c)$ is the set of cells adjacent to c usually (for a 45° discrimination, 8 cells are connected in 2D space or 26 cells connected in 3D space) and $distance(c, a)$ is the distance between c and a. Brushfire only makes one pass through the grid but

has the added expense of keeping a priority queue which usually requires $O(log(x))$ time for operations, where x is the number of elements in the queue.

Just as Brushfire is analogous to Dijkstra algorithm for planning, **Dynamic brushfire** is analogous to the unfocused D* family of efficient replanning algorithms. When new information is received concerning the environment, these algorithms only propagate the new information to portions of the map that could be affected.Thus, they avoid unnecessarily reprocessing the entire state space. In the grid based Generalized Voronoi Diagram context, new information consists of obstacles being asynchronously added and removed from the environment, in whole or in part. When an obstacle cell is removed, the distances increase for exactly those cells that were closer to it than to any other obstacle cell. When an obstacle cell is added, the distances decrease for exactly those cells that are closer to it now than any other obstacle cell. Dynamic brushfire is efficient because it determines and limits processing to only cells within these two sets. Dynamic brushfire requires a grid and a mapping from obstacle cell to obstacle. For each cell s it maintains the obstacle $obst_s$ to which it is currently closest. It also maintains a distance $dist_s$ to its closest obstacle given the changes that have occurred since that construction. A cell is consistent if $dist_s = dist_s^{new}$, overconsistent if $dist_s > dist_s^{new}$ and underconsistent if $dist_s < dist_s^{new}$. Like A*, D*, brushfire, Dynamic Brushfire keeps a priority queue OPEN of the inconsistent cells to propagate changes. A cell's priority is always $min(dist_s, dist_s^{new})$ and cells are popped with increasing priority values. When a cell is popped from the OPEN queue, its new distance is propagated to its adjacent cells, and any newly inconsistent cells are put on the OPEN queue. Thus the propagation emanates from the source of the change and terminates when the change does not affect any more cells.

When a cell is set to an obstacle, it reduces the new minimum distance of adjacent cells which propagate this reduction to their adjacent cells. This creates an overconsistent sweep emanating from the original obstacle. An overconsistent sweep terminates when cells are encountered that are equally close or closer to another obstacle and thus cannot be made overconsistent. These cells lie on the boundary between Voronoi regions and will be part of the new GVD. When an obstacle cell is removed, all cells that previously used it to compute their distances are invalid and must recompute their distances. This propagation occurs in two sweeps.

1. An underconsistent propagation sweeps out from the original cell and resets the affected cells. This sweep terminates when cells are encountered that are closer to another obstacle and cannot be made underconsistent. Thus, at most those cells in the removed obstacle cell's Voronoi region are invalidated.
2. Then, an overconsistent propagation sweeps inwards and uses the valid cells beyong this region to recompute new values for the invalide cells.

2.6.1.5 Replanning with RRT Algorithm

The RRT methods have been shown to be effective to solving single shot path planning problems in complex configuration spaces. By combining random sampling of

the configuration space with biased sampling around the goal configuration, RRT efficiently provide solutions to problems involving vast, high dimensional configuration spaces that would be intractable using deterministic approaches. In real world aerial robotic scenarios, the initial information available is incomplete and the environment itself is dynamic. The initial solution may become invalid as new information is received, through on-board sensors or exterior informations. When this occurs, typically the current RRT is abandoned and a new RRT is grown from scratch. A replanning algorithm presented in [43] repairs RRT when new information concerning the configuration space is received. Instead of abandoning the current RRT entirely, this approach mimics deterministic replanning algorithms by efficiently removing just the newly-invalid parts and maintaining the rest. It then grows the remaining tree until a new solution is found. The resulting algorithm, **Dynamic RRT** (DRRT) is a probabilistic analog to D* family of deterministic replanning algorithm. The proposal is a variation of the goal biased RRT algorithm called **Homotopic RRT**. It allows a constrained growing of the tree only in those directions that satisfy a given homotopy class. Before adding a new node into the tree, the topological path traversed is checked to ensure it belongs to the homotopy class by computing the intersections of the path with the reference frame.

Algorithm 26 Replanning RRT Algorithm

1. **RegrowRRT**
2. TrimRRT()
3. GrowRRT()
4. **TrimRRT()**
5. $S = \emptyset, i = 1$
6. while $(i < T.size())$
7. $q_i = T.node(i); q_p = Parent(q_i)$
8. if $q_p.flag =$ INVALID
9. $q_i.flag =$ INVALID
10. if$q_i.flag \neq$ INVALID
11. $S = S \cup \{q_i\}$
12. $i = i + 1$
13. T = CreateTreeFromNodes(S)
14. **InvalidateNodes**(obstacles)
15. E = FindAffectedEdges(obstacle)
16. for each edge $e \in E$
17. q_e =ChildEndPOIntNode(e)
18. $q_e.flag$ = INVALID

The pseudo-code of the method is detailed in Algorithm 26 in page 156. It begins with an RRT generated from an initial configuration to a goal configuration. When changes occur to the configuration space (e.g. through receiving new information), all the parts of the RRT that go through obstacles, are invalidated by these changes. The tree is then trimed to remove all the invalidated parts. At this point, all the nodes and edges remaining in the tree are guaranteed to be valid, but the tree may no longer reach the goal. Finally, the tree is grown out until the goal is reached once more. When

an obstacle is added to the configuration space, first the edges in the current tree that intersect this obstacle are found. Each of these edges will have two endpoint nodes in the tree: one will be the parent of the other in the RRT. One of these nodes (the parent) will have added the other (the child) to the tree through an Extend operation. The child endpoint node of each edge is then marked as invalid. After all the child endpoint nodes of the affected edges have been marked, the solution path is checked for invalid nodes. If any are found, the RRT needs to be regrown. This involves trimming the tree and growing the trimmed tree out to the goal (RegrowRRT). Trimming the tree involves stepping through the RRT in the order in which nodes were added and marking all child nodes as invalid whose parent are invalid. This effectively breaks off branches where they directly collide with new obstacles and removes all nodes on these branches. Once the tree has been trimmed, it can be grown out to the goal. This can be performed in the same manner as the basic RRT algorithm for initial construction. However, depending on how the configuration space has changed, it may be more efficient to focus the growth towards areas that have been affected [43].

2.6.2 Anytime Algorithms

Anytime algorithms try to find the best plan within the given available time. When the planning problem is complex, it may not be possible to obtain optimal solutions within the deliberation time available to an aerial robot. Anytime algorithms are appropriate in such settings, as they usually provide an initial, possibly highly suboptimal solution quickly, then concentrate on improving this solution until the time available for planning runs out. The aerial robot must be satisfied with the best solution that can be generated within the available computation time. A useful set of algorithms for generating such solutions are known as anytime algorithms. Typically these start out by computing an initial, potentially highly suboptimal solution, then improve this solution as time allows. The interaction between replanning algorithms and anytime algorithms began a decade ago [89]. Replanning algorithms have concentrated on finding a single solution with a fixed suboptimality bound and anytime algorithms have concentrated on static environments. But the most interesting problems are those that are both dynamic (requiring replanning) and complex (requiring anytime approaches). Heuristic based anytime replanning algorithms are presented in this section, to bridge the gap between these two areas of research.

2.6.2.1 Anytime Repairing A*: ARA* Algorithm

This section presents a graph-based planning and replanning algorithm able to produce bounded suboptimal solutions in an anytime fashion. This algorithm tunes the quality of its solution based on available search time at every step reusing previous search efforts. When updated information regarding the underlying graph is received, the algorithm incrementally repairs its previous solution. The result is an approach

that combines the benefits of anytime and incremental planners to provide efficient solutions to complex, dynamic search problems. A* based anytime algorithms uses the fact that inflating the heuristic values used by A* (resulting in the **weighted A*** search) often provides substantial speed-ups at the cost of solution optimality. A* has the nice property that if the heuristic used is consistent and the heuristic values are multiplied by an inflation factor $\epsilon > 1$, then the cost of the generated solution is guaranteed to be within ϵ times the cost of an optimal solution.

Anytime Heuristic Search (AHS) is an anytime version of weighted A*. It finds an initial solution for a given value of ϵ and continues the search after an initial solution is found with the same ϵ. Each time the goal state is extracted from OPEN, an improved solution is found. Eventually, Anytime Heuristic algorithm expands the state in OPEN with minimal $f(s) = g(s) + \epsilon h(s)$ where ϵ is a parameter of the algorithm. The suboptimality of intermediate solutions can be bounded by $G/min_{s \in OPEN} g(s) + h(s)$, as G, the cost of the current best solution and $min_{s \in OPEN} g(s) + h(s)$ is a lower bound of the cost of the optimal solution.

The algorithm **Anytime Repairing A*** (ARA*) uses the notion of consistency introduced above to limit the processing performed during each search by only considering those states whose costs at the previous search may not be valid given the new ϵ value. ARA* is also based on weighted A*. It finds an initial solution for a given initial value of ϵ and continues the search with progressively smaller values of ϵ to improve the solution and reduce its suboptimality bound. The value of ϵ is decreased by a fixed amount each time an improved solution is found or the current-best solution is proven to be $\epsilon-$suboptimal. The $f(s)$ values of the states $s \in OPEN$ are then updated to account for the new value of ϵ. The initial value of ϵ and the amount by which it is decreased in each iteration are parameters of the algorithm. It begins by performing an A* search with an inflation factor ϵ_0 but during this search it only expands each state at most once. Once a state has been expanded during a particular search, if it becomes inconsistent due to a cost change associated with a neighboring state then it is not reinserted into the queue of states to be expanded. Then, when the current search terminates, the states in the INCONS list are inserted into a fresh priority queue (with new priorities based on the new inflation faction ϵ), which is used by the next search.

1. by only expanding each state at most once a solution is reached much more quickly.
2. by only reconsidering states from the previous search that were inconsistent, much of the previous search effort can be reused

Thus, when the inflation factor is reduced between successive searches, a relatively minor amount of computation is required to generate a new solution. The backwards version of the ARA* is shown in algorithm 27 in page 159. Here, the priority of each state s in the OPEN queue is computed as the sum of its cost g(s) and its inflated heuristic value $\epsilon h(s_{start}, s)$. CLOSED contains all states already expanded once in the current search and INCONS contains all states that have already been expanded and are inconsistent.

Algorithm 27 ARA* Algorithm

,

1. **Key (s)**
2. return $g(s) + \epsilon h(s_{start}, s)$
3. **ImprovePath()** while $(min_{s \in > OPEN}(keys(s_{start})$
4. remove state s with the minimum key from OPEN
5. CLOSED = CLOSED $\cup \{s\}$
6. for all $s' \in Pred(s)$
7. if s' was not visited before
8. $g(s') = \infty$
9. if $(g(s') > c(s', s) + g(s))$
10. $(g(s') = c(s', s) + g(s))$
11. if $s' \notin$ CLOSED
12. insert s' into OPEN with key(s')
13. else
14. insert s' into INCONS;
15. **Main**
16. $g(s_{start}) = \infty$; $g(s_{goal}) = 0$
17. $\epsilon = \epsilon_0$
18. OPEN = CLOSED = INCONS = $\emptyset$
19. insert s_{goal} into OPEN with $key(s_{goal})$
20. while $\epsilon > 1$
21. decrease ϵ
22. Move states from INCONS into OPEN
23. Update the priorities for all $s \in$ OPEN according to key (s)
24. CLOSED = $\emptyset$
25. ImprovePath();
26. publish current $\epsilon-$suboptimal solution;

Restarting Weighting A* (RWA*) is similar to ARA* but it restarts the search each time ϵ is decreased. That is, each search is started with only the start date on the OPEN queue. It reuses the effort of previous searches by putting the states explored in previous iterations on a SEEN list. Each time, the search encounters a seen state, it is put back on the OPEN queue regardless of whether its g-value was decreased. Restarting has proven to be effective in situations where the quality of the heuristic varies substantially across the search space. As with ARA* the initial value of ϵ and the amount by which it is decreased in each iteration are parameters of the algorithm.

Anytime Window A* (AWA*) is based on unweighted A*, but expands states within an active window that slides along with the deepest state expanded so far. Only states with a g-value inside the window (i.e. the g-value is larger than the largest g-value among the states expanded so far minus the window size) are put on the OPEN queue. The other states are put on an auxiliary SUSPEND list. Iteratively, the window size is increased to broaden the search, which will eventually become equivalent to A*. A problem of this algorithm is that if it misses the goal state in its initial search (i.e. the goal state is outside the window), it exhausts the entire search space before the initial search terminates, which may take prohibitively long.

2.6.2.2 Anytime Dynamic A*: ADA* Algorithm

The algorithm, **Anytime Dynamic A*** (ADA*) continually improves its solution while deliberation time allows and corrects its solution when updated information is received. As shown in the previous sections, there exist efficient algorithms for coping with dynamic environments (e.g. D* and D* Lite) and complex planning problems (ARA*). ADA* copes with complex planning problems in dynamic environments. ADA* performs a series of searches using decreasing inflation factors to generate a series of solutions with improved bounds, as with ARA*. When there are changes in the environment affecting the cost of edges in the graph, locally affected states are placed on the OPEN queue with priorities equal to the minimum of their previous key value and their new key value, as with D* Lite. States on the queue are then processed until the current solution is guaranteed to be $\epsilon-$ suboptimal. The pseudo code of the ADA* method is presented in Algorithm 28 in page 161.

The main function first sets the inflation factor ϵ to a sufficiently high value ϵ_0, so that an initial suboptimal plan can be generated quickly. Then, unless changes in edges costs are detected, the Main function decreases ϵ and improves the quality of its solution until it is guaranteed to be optimal, that is $\epsilon = 1$. This phase is exactly the same as for ARA*: each time ϵ is decreased, all inconsistent states are moved from INCONS to OPEN and CLOSED is made empty. When changes in edge costs are detected, there is a chance that the current solution will no longer be $\epsilon-$suboptimal. If the changes are substantial, then it may be computationally expensive to repair the current solution to regain $\epsilon-$suboptimality. In such a case, the algorithm increases ϵ so that a less optimal solution can be produced quickly. Because edge cost increases may cause some states to become underconsistent, a possibility not present in ARA*, states need to be inserted into the OPEN queue with a key value reflecting the minimum of their old cost and their new cost. Further, in order to guarantee that underconsistent states propagate their new costs to their affected neighbors, their key values must use uninflated heuristic values. This means that different key values must be computed for underconsistent states than for overconsistent states. By incorporating these considerations, ADA* is able to handle both changes in edge costs and changes to the inflation factor ϵ. ADA* has two parameters the initial value of ϵ and the amount $\Delta\epsilon$ by which ϵ is decreased in each iteration. Setting these parameters requires trial-and-error and domain expertise.

2.6.2.3 Anytime Nonparametric A*: ANA* Algorithm

In many applications, real-time constraints require interactive performance and A* might not be able to compute the optimal solution within the available amount of time. In such cases, anytime algorithms produce an initial, suboptimal solution quickly, and then improve upon this solution. At a given time, the current best solution is available, and eventually the optimal solution will be found [149]. **Anytime A* algorithms** are based on weighted A*, which inflates the heuristic node values by a factor of $\epsilon \geq 1$ to trade off running time versus solution quality [149]. These algorithms repeatedly

Algorithm 28 ADA* Algorithm

Key (s)
if $g(s) > rhs(s)$
return $rhs(s) + \epsilon h(s_{start}, s), rhs(s)$
else
return $g(s) + \epsilon h(s_{start}, s), g(s)$
UpdateState(s)
If s was not visited before
$g(s) = \infty$
if $(s \neq s_{goal}), rhs(s) = min_{s' \in Succ(s)} \left(g(s') + c(s, s')\right)$
if $s \in$ OPEN, remove s from OPEN
if $(g(s) \neq rhs(s))$
if $s \notin$ CLOSED
insert s into OPEN with $key(s)$
else
insert s into INCONS
ComputeorImprovePath()
while $(min_{s \in OPEN}(key(s)) < keys(s_{start} ORrhs(s_{start}) \neq g(s_{start}))$
remove state s with the minimum key from OPEN
if $g(s) \geq rhs(s)$
g(s) = rhs(s)
CLOSED = CLOSED $\cup \{s\}$
for all $s' \in Pred(s)$ UpdateState(s')
else
$g(s) = \infty$
for all $s' \in Pred(s) \cup \{s\}$ UpdateState(s')
Main
$g(s_{start}) = rhs(s_{start}) = \infty$; $g(s_{goal}) = \infty$
$rhs(s_{goal}) = 0$; $\epsilon = \epsilon_0$;
OPEN = CLOSED = INCONS = $\emptyset$
insert s_{goal} into OPEN with $key(s_{goal})$
ComputeorImprovePath();
publish current $\epsilon-$suboptimal solution;
forever
if changes in edge costs are detected
for all directed edges (u,v) with changed edge costs
Update the edge cost c(u,v);
UpdateState(u)
if significant edge cost changes were observed.
increase ϵ or replan from scratch
else if $\epsilon > 1$
decrease ϵ
Move states from INCONS into OPEN
Update the priorities for all $s \in$ OPEN according to key (s)
CLOSED = $\emptyset$
ComputeorImprovePath()
publish current $\epsilon-$suboptimal solution;
if $\epsilon = 1$
wait for changes in edge costs;

expand the 'open' state s that has minimal value of

$$f(s) = g(s) + \epsilon h(s) \tag{2.264}$$

where $g(s)$ is the current best cost to move from the start state to s, and $h(s)$ is the heuristic function, an estimate of the cost to go from s to a goal state. The higher ϵ, the greedier the search and the sooner an initial solution is found. If the heuristic is admissible, a lower bound on the true distance to the goal, the suboptimality of solutions is bounded by ϵ, i.e. the solution is guaranteed to be no costlier than ϵ times the cost of the optimal path. These observations can be used in an anytime algorithm, for instance as in ARA*, which initially runs weighted A* with a large value of ϵ to quickly find an initial solution, and continues the search progressively decreasing ϵ to improve the solution and reduce its suboptimality bound.

The Anytime Nonparametric A* (ANA*) algorithm does not require parameters. Instead of minimizing f(s), ANA* expands the open state s with a maximal value of

$$e(s) = \frac{G - g(s)}{h(s)} \tag{2.265}$$

Algorithm 29 ANA* Algorithm

1. **ImproveSolution()**
2. While OPEN $\neq \emptyset$
3. $s \rightarrow argmax_{s \in OPEN}\{e(s)\}$
4. $OPEN \leftarrow OPEN\{s\}$
5. If $e(s) < E$ then
6. E $\leftarrow$ e(s)
7. if IsGoal(s) then
8. $G \leftarrow g(s)$
9. return
10. for each successor s' of s do
11. if $g(s) + c(s, s') < g(s')$ then
12. $g(s') \leftarrow g(s) + c(s, s')$
13. pred(s') $\leftarrow$ s
14. if $g(s') + h(s') < G$ then
15. Insert or update s' in OPEN with key e(s')
16. **Main**
17. E $\leftarrow \infty$, OPEN $\leftarrow \emptyset$, $g(s_{start}) \leftarrow 0 G \leftarrow \infty$
18. Insert s_{start} into OPEN with key $e(s_{start})$
19. While OPEN $\neq \emptyset$ do
20. ImproveSolution()
21. Report current E-suboptimal solution
22. Update keys e(s) in OPEN and prune states if g(s) + h(s) $\geq G$

where G is the cost of the current best solution, initially an arbitrarily large value. The value of e(s) is equal to the maximal value of ϵ such that $f(s) \leq G$. Hence,

continually expanding the node s with maximal e(s) corresponds to the greediest possible search to improve the current solution that in effect automatically adapts the value of ϵ as the algorithm progresses and path quality improves. The maximal value of e(s) provides an upper bound on the suboptimality of the current best solution, which is hence gradually reduced while ANA* searches for an improved solution. The algorithm ANA*, algorithm 29 in page 162 in contrast, lets the value of ϵ follow a linear trajectory, resulting in highly unpredictable search times between the fixed decrements of ϵ.

Throughout the algorithm, a global variable G is maintained storing the cost of the current best solution. Initially, $G = \infty$, as no solution has yet been found. ImproveSolution implements a version of algorithm A* that is adapted such that it expands the state $s \in OPEN$ with the maximal value of e(s). Each time a state s is expanded, it is checked whether the g-value of the successors s' of s can be decreased. If so, $g(s')$ is updated and the predecessor of s' is set to s such that the solution can be reconstructed once one is found. Subsequently, s' is inserted into the OPEN queue with key $e(s')$, or if it was already on the OPEN queue, its key $e(s')$ is updated. States for which $g(s)+h(s) \geq G$, or equivalently $e(s) \leq 1$, are not put on the OPEN queue, though, as such states will never contribute to improving the current best solution. As a result, when a goal state is extracted from OPEN, it is guaranteed that a solution has been found with lower cost than the current best solution, so G is updated and ImproveSolution terminates. The main function iteratively calls ImproveSolution to improve the current solution. It starts by initializing the g-value of the start state s_{start} to zero and putting it on OPEN. In the first iteration $G \leftarrow \infty$ as no solution has yet been found, in which case ImproveSolution expands the state in OPEN with the smallest $h(s)$ value, and in case of ties the one with the smallest $g(s)$ value. This is equivalent to executing **Weighted A* algorithm** minimizing $f(s)$ with $\epsilon = \infty$, so the search for an initial solution is maximally greedy [138].

In [142], the complete algorithm is refined into an anytime algorithm that first quickly finds a solution and then uses any remaining time to incrementally improve that solution until it is optimal or the algorithm is terminated.

In [108], an anytime planner that builds off **Safe Interval Path Planning** (SIPP) which is a fast variant of A* for planning in dynamic environment is presented. It uses intervals instead of timesteps to represent the time dimension of the problem. In addition, an optional time horizon is introduced, after which the planner drops time as a dimension. On the theoretical side, it is shown that in the absence of time horizon, this planner can provide guarantees on completeness as well as bounds on the sub-optimality of the solution with respect to the original space-time graph.

2.7 Conclusions

After a brief introduction, controllability of an aerial robot represented by its translational kinematic model is studied. Then trajectory planning methods such as trim trajectories or parametric curves are presented. Then, nonholonomic motion

planning has been presented in the third section of this chapter. Then, Obstacle/conflict avoidance Planning was formulated as either a discrete or continuous task. Methods available for both approaches were detailed. Graph search algorithms generate the path neglecting aerial robot's characteristics. This approach allows planning of the path for any aerial robot but does not guarantee match of the path with turn and climb limitations. This drawback should be faced using smoothing algorithms to adapt locally the path to the platform characteristics through single waypoint reallocation. Finally, replanning methods have been presented.

References

1. Ambrosino G, Ariola M, Ciniglio U, Carraro F, Delellis E, Pironti A (2009) Path Generation and Tracking in 3D for UAVs. IEEE Trans Control Syst Technol 17:980–988
2. Anderson R, Bakolas E, Milutinovic D, Tsiotras P (2013) Optimal feedback guidance of a small aerial vehicle in a stochastic wind. AIAA J Guidance Control Dyn 36:975–985
3. Anderson R, Bakolas E, Milutinovic D, Tsiotras, P (2012) The Markov-Dubins problem in the presence of a stochastic drift field. In: Proceedings of the IEEE conference on decision control, pp 130–135
4. Arvanitakis I, Tzes A (2012) Trajectory optimization satisfying the robot's kinodynamic constraints for obstacle avoidance. In: Proceedings of the 20th Mediterranean conference on control and automation, pp 128–133
5. Aurenhammer F (1991) Voronoi diagrams, a survey of fundamental geometric data structure. ACM Comput Surv 23:345–405
6. Avanzini G (2004) Frenet based algorithm for trajectory prediction. AIAA J Guidance Control Dyn 27:127–135
7. Bakolas E, Tsiotras P (2012) Feedback navigation in an uncertain flowfield and connections with pursuit strategies. AIAA J Guidance Control Dyn 35:1268–1279
8. Ben Asher JZ (2010) Optimal control theory with aerospace applications. AIAA Press, Reston
9. Bestaoui Y (2012) 3 D flyable curve for an autonomous aircraft. In: ICNPAA world congress on mathematical problems in engineering, sciences and aerospace, Vienna, AIP Conference Proceedings, 1493, pp 132–139. doi:http://dx.doi.org/10.1063/1.4765481
10. Bestaoui Y (2011) 3D curves with a prescribed curvature and torsion for an aerial robot. Int J Comput Appl 41:269–274
11. Bestaoui Y, Kahale E (2013) Time optimal trajectories of a lighter than air robot with second order constraints and a piecewise constant velocity wind. AIAA J Inf Syst 10:155–171. doi:10.2514/1.55643
12. Bestaoui Y, Hima S (2007) Modeling and trajectory generation of lighter than air aerial robot. In: Kozlowski K (ed) Robot motion and control 2007. Springer, LNCIS 360, pp 3–28
13. Bethke B, Valenti M, How JP (2008) UAV task assignment, an experimental demonstration with integrated health monitoring. IEEE J Rob Autom 33:39–44
14. Bhat S, Venkatraman A (2009) Optimal planar turns under acceleration constraints. IEEE Trans Autom Control 54:1654–1660
15. Biggs J, Holderbaum W, Jurdjevic V (2007) Singularities of optimal control problems on some 6D Lie groups. IEEE Trans Autom Control 52:1027–1038
16. Bilimoria KD (2000) A geometric optimization approach to aircraft conflict resolution. In: AIAA guidance, navigation, and control conference, Denver, Co., paper AIAA-2000-4265
17. Blackmore L, Ono M, Bektassov A, Williams B (2010) A probabilistic particle control approximation of chance constrained stochastic predictive control. IEEE Trans Rob 26:502–517
18. Bloch AM (2003) Non holonomics mechanics and control. Springer, Berlin

19. Boizot N, Gauthier JP (2013) Motion planning for kinematic systems. IEEE Trans Autom Control 58:1430–1442
20. Boukraa D, Bestaoui Y, Azouz N (2008) Three dimensional trajectory generation for an autonomous plane. Inter Rev Aerosp Eng 4:355–365
21. Braunl T (2008) Embedded robotics. Springer
22. Chakravorty S, Kumar S (2011) Generalized sampling based motion planners. IEEE Trans Syst Man Cybern part B: Cybernetics, 41:855–866
23. Chamseddine A, Li T, Zhang Y, Rabbath C, Theilliol D (2012) Flatness based trajectory planning for a quadrotor UAV test based considering actuator and system constraint. In: American control conference, Montreal, pp 920–925
24. Choset H, Lynch K, Hutchinson S, Kantor G, Burgard W, Kavraki L, Thrum S (2005) Principles of robot motion. The MIT Press, Theory, Algorithms and implementation
25. Conte G, Moog CH, Perdon AM (2006) Algebraic methods for nonlinear control systems: theory and applications. Springer, New York
26. Conway BA (2010) Spacecraft trajectory optimization. Cambridge Press, New York
27. Corbets JB, Langelaan JW (2010) Real time trajectory generation for target localization using micro air vehicles. AIAA J Aerosp Comput Inf Commun 7:223–240
28. Corke P (2011) Robotics, vision and control: Fundamentals algorithms in Matlab. Springer, Berlin
29. Cormen TH (2009) Introduction to algorithms. The MIT Press, Cambridge
30. Coron J-M (1998) On the stabilization of some nonlinear control systems: results, tools, and applications. NATO Advanced Study Institute, Montreal
31. Dadkhah N, Mettler B (2012) Survey of motion planning literature in the presence of uncertainty: considerations for UAV guidance. J Intell Rob Syst 65:233–246
32. Dai R, Cochran J (2010) Path planning and state estimation for UAV in hostile environments. AIAA J Guidance Control Dyn 33:595–601
33. Davis JD, Chakravorty S (2007) Motion planning under uncertainty: application to an unmanned helicopter. AIAA J Guidance Control Dyn 30:1268–1276
34. Dicheva S, Bestaoui Y (2011) Route finding for an autonomous aircraft. AIAA aerospace sciences meeting, Orlando, Fl. doi:10.2514/6.2011-79
35. Dicheva S, Bestaoui Y (2012) 3D waypoint generation in a dynamic environment for an airborne launch mission. Proc Inst Mech Eng [G] J Aerosp Eng 226:1283–1297. doi:10.1177/0954410011419565
36. Dubins LE (1957) On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. Am J Math 79:497–517
37. Duleba I, Sasiadek J (2003) Nonholonomic motion planning based on Newton algorithm with energy optimization. IEEE Trans Control Syst Technol 11:355–363
38. Eele A, Richards A (2009) Path planning with avoidance using nonlinear branch and bound optimization. AIAA J Guidance Control Dyn 32:384–394
39. Egerstedt M, Martin C (2004) A note on the connection between Bezier curves and linear optimal control. IEEE Trans Autom Control 49:1728–1731
40. Fahimi F (2009) Autonomous robots: modeling, path planning and control. Springer, New York
41. Farault J (2008) Analysis on Lie groups: an introduction. Cambridge studies in advanced mathematics, Springer, New York
42. Farouki RT (2008) Pythagorean hodograph curves. Springer, Berlin
43. Ferguson D, Kalva N, Stentz A (2006) Replanning with RRT. In: IEEE International Conference on Robotics and Automation, pp 1243–1248. doi:1.01109/ROBOT.2006/641879
44. De Filippis L, Guglieri G (2012) Path planning strategies for UAV in 3D environments. J Int Rob Syst 65:247–264
45. Fraichard T, Scheuer A (2008) From reeds and Shepp's to continuous curvature paths. IEEE Trans Rob 20:1025–10355
46. Frazzoli E, Dahleh MA, Feron E (2008) Maneuver based motion planning for nonlinear systems with symmetries. IEEE Trans Rob 4:355–365

47. Frazzoli E, Dahleh MA, Feron E (2002) Real time motion planning for agile autonomous vehicles. AIAA J Guidance Control Dyn 25:116–129
48. Funabiki K, Ijima T, Nojima T (2013) Method of trajectory generation for perspective flight path display in estimated wind condition. J Aerosp Inf Syst 10:240–249. doi:10.2514/1.37527
49. Garcia M, Viguria A, Ollero A (2012) Dynamic graph search algorithm for global path planning in presence of hazardous weather. J Intell Rob Syst. doi:10.1007/s10846-012-9704-7
50. Goerzen C, Kong Z, Mettler B (2010) A survey of motion planning algorithms from the perspective of autonomous UAV guidance. J Intell Rob Syst 20:65–100
51. Gong Q, Lewis LR, Ross IM (1991) Pseudospectral motion planning for autonomous vehicles. AIAA J Guidance Control Dyn 32:1039–1045
52. Guerrero JA, Bestaoui Y (2013) UAV path planning for structure inspection in windy environments. J Intell Rob Syst 69, 297–311
53. Guerrero JA, Bestaoui Y, Lozano R (2012) Optimal guidance for rotorcraft platoon formation flying in wind fields. In: Guerrero JA, R. Lozano R (ed) Unmanned aerial vehicles formation. Wiley-ISTE, ISBN: 978-1-84821-323-4
54. Habib Z, Sakai M (2003) Family of G^2 cubic transition curves. In: International conference on geometric modelling and graphics, pp 117–122. doi:10.1109/GMAG.2003.1219675
55. Held M (2001) VRONI: an engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput Geom 18:95–123
56. Holt J, Biaz S, Affane AC (2013) Comparison of unmanned aerial system collision avoidance algorithm in a simulated environment. AIAA J Guidance Control Dyn 36:881–883
57. Horwood J, Aragon N, Poore A (2011) Gaussian sum filters for space surveillance: theory and simulation. AIAA J Guidance Control Dyn 34:1839–1851
58. Hota S, Ghose D (2009) A modified Dubins method for optimal path planning for a miniature air vehicle converging to a straight line path. In: American control conference, pp 2397–2402
59. Hsu D, Isler V, Latombe JC, Lin MC (2011) Algorithmic foundations of robotic. Springer, Berlin
60. Igarashi H, Loi K (2001) Path-planning and navigation of a mobile robot as a discrete optimisation problems. Art Life Rob 5:72–76
61. Jaakoola TS (2001) Tutorial on variational approximation methods. In: Opper M, Saad D (eds) Advanced Mean Field Methods, MIT Press, Cambridge
62. Jaklic G, Kozak J, Krajnc M, Vitrih V, Zagar E (2008) Geometric lagrange interpolation by planar cubic pythagorean hodograph curves. Compu Aided Geom Des, 720–728
63. Janiak M, Tchon K (2011) Constrained motion planning of nonholonomic systems. Syst Control Lett 60:625–631
64. Jardin MR, Bryson AE (2001) Neighboring optimal aircraft guidance in winds. AIAA J Guidance Control Dyn 24:710–715
65. Jardin MR, Bryson AE (2012) Methods for computing minimum time paths in strong winds. AIAA J Guidance Control Dyn 35:165–171
66. Jennings AL, Ordonez R, Ceccarelli N (2008) Dynamic programming applied to UAV way point path planning in wind. IEEE international symposium on computer-aided control system design. San Antonio, TX, pp 215–220
67. Jiang Z, Ordonez R (2008) Robust approach and landing trajectory generation for reusable launch vehicles in winds. 17th IEEE international conference on control applications. San Antonio, TX, pp 930–935
68. Jurdjevic V (2008) Geometric control theory. Cambridge studies in advanced mathematics, Cambridge University Press, Cambridge
69. Kaelbling L, Lozano-Perez T (2012) Integrated robot task and motion planning in belief space. Technical report MIT-CSAIL-TR-2012-09, MIT
70. Kala R, Shukla A, Tiwari R (2010) Fusion of probabilistic A* algorithm and fuzzy inference system for robotic path planning. Artificial Intell Rev 307–327, Springer
71. Kalra N, Ferguson D, Stentz A (2009) Incremental reconstruction of generalized voronoi diagrams on grid. Rob Auton Syst 57:123–128

72. Kalyanam K, Chandler P, Pachter M, Darbha S (2012) Optimization of perimeter patrol operations using UAV. J Guidance Control Dyn 35:434–441
73. Kaminer I, Khargonekar PP, Robel G (1990) Design of a localizer capture and track modes for a lateral autopilot using H_∞ synthesis. IEEE Control Syst Mag 10:13–21
74. Kampke T, Elfes A (2003) Optimal aerobot trajectory planning for wind based opportunity flight control. IEEE/RSJ International conference on intelligent robots and systems. Las Vegas, NV, pp 67–74
75. Karaman S, Frazzoli E (2010) Incremental sampling-based algorithms for optimal motion planning. In: Matsuoka Y, Durrant-White H, Neira J (eds) Robotics, science and systems. The MIT Press, pp 267–274
76. Kavraki L, Latombe JC (1994) Randomized preprocessing of configuration space for fast path planning. IEEE Inter Conf Rob Autom 3:2138–2145
77. Khatib O (1985) Real time obstacle avoidance for manipulators and mobile robots. In: IEEE international conference on robotics and automation
78. Kim J, Khosla PK (1992) Real time obstacle avoidance using harmonic potential functions. IEEE Trans Rob Autom 8:338–349
79. Kong Z, Mettler B (2009) On the general characteristics of 2D optimal obstacle field guidance solution. In: IEEE conference on decision and control, pp 3448–3453
80. Kozlowski K (ed) (2006) Robot motion and control: recent developments. Springer, Heidelberg
81. Krozel J, Penny S, Prete J, Mitchell JSB (2007) Automated route generation for avoiding deterministic weather in transition airspace. AIAA J Guidance Control Dyn 30:144–153
82. Kuwata Y, Schouwenaars T, Richards A, How J (2005) Robust constrained receding horizon control for trajectory planning. In: AIAA conference on guidance, navigation and control
83. Langelan JW, Chakrabarty A, Deng A, Miles K (2013) Green flight challenge: aircraft design and flight planning for extreme fuel efficiency. AIAA J Aircr 50:832–846
84. Laugier C, Chatila R (eds) (2007) Autonomous navigation in dynamic environment. Springer, Heidelberg
85. Laumond JP (1998) Robot motion planning and control, LNCIS 229. Springer, New York
86. Lavalle SM (2006) Planning algorithms. Cambridge University Press, Cambridge
87. LeNy J, Feron E, Frazzoli E (2012) On the dubins traveling salesman problem. IEEE Trans Autom Control 57:265–270
88. Li Z, Canny JF (1992) Non holonomic motion planning. Kluwer Academic Press, Berlin
89. Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S (2005) Anytime dynamic A*: an anytime replanning algorithm. In: Proceedings of the international conference on automated planning and scheduling: ICAPS, American Association for Artificial Intelligence, http://www.aaai.org
90. Lorenz RD (2001) Flight power scaling of airplanes, airships and helicopters: application to planetary exploration. AIAA J Aircr 38:208–214
91. Ludington B, Johnson E, Vachtsevanos A (2006) Augmenting UAV autonomy (GTMAX). IEEE Rob Autom Mag 21:67–71
92. Macharet D, Neto AA, Campos M (2009) On the generation of feasible paths for aerial robots in environments with obstacles. In: IEEE/RSJ international conference on intelligent robots and systems, pp 3380–3385
93. Mantegh I, Jenkin M, Goldenberg A (2010) Path planning for autonomous mobile robots using the boundary integral equation method. J Intell Rob Syst. doi:10/1007
94. Marble JD, Bekris K (2013) Asymptotically near-optimal planning with probabilistic roadmap spanners. IEEE Trans Rob 29:432–444
95. Marigo A, Bichi A (1998) Steering driftless nonholonomic systems by control quanta. IEEE Inter Conf Decis Control 4:466–478
96. Marsden J, Ratiu TS (1999) Introduction to mechanics and symmetry. Springer, New York
97. Martin P, Egerstedt M (2008) Expanding motion programs under input constraints. 9th international workshop on discrete event systems. Goteborg, Sweden, pp 156–161

98. Masoud AA (2010) Kinodynamic motion planning. In: IEEE robotics and automation magazine, pp 85–99
99. Mattei M, Blasi L (2010) Smooth flight trajectory planning in the presence of no-fly zones and obstacles. AIAA J Guidance Control Dyn 33(2), 454–462
100. Matveev AS, Teimoori H, Savkin A (2010) Navigation of a nonholonomic vehicle for gradient climbing and source seeking without gradient estimation. In: American control conference, pp 219–223
101. McGee T, Hedrick JK (2007) Optimal path planning with a kinematic airplane model. AIAA J Guidance Control Dyn 30, 629–633
102. Mettler B, Dakhah N, Kong Z (2010) Agile autonomous guidance using spatial value functions. Control Eng Pract 18:773–788
103. Miele A, Wang T, Melvin W (1986) Optimal take-off trajectories in the presence of windshear. J Optim Theory Appl 49:1–45
104. Miele A, Wang T, Melvin W (1989) Penetration landing guidance trajectories in the presence of windshear. AIAA J Guidance 12:806–814
105. Missiuro P, Roy N (2006) Adaptive probabilistic roadmaps to handle uncertain maps. IEEE international conference on robotics and automation. Orlando, Fl, pp 1261–1267
106. Mujumdar A, Padhi R (2011) Evolving philosophies for autonomous obstacle/collision avoidance of UAV. AIAA J Aerosp Inf Commun 28:17–41
107. Naldi R, Marconi L (2011) Optimal transition maneuvers for a class of V/STOL aircraft. Automatica 47:870–879
108. Narayaman V, Phillips M, Likhachev M (2012) Anytime safe interval path planning for dynamic environments. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems: IROS
109. N'doye I, Zasadzinski M, Darouach M, Radhy N (2011) Exponential stabilization of a class of nonlinear systesm: a generalized Gronwall Bellman lemma approach. Nonlinear Anal 74:7333–7341
110. Nelson R, Barber B, McLain T, Beard R (2007) Vector field path following for miniature air vehicle. IEEE Trans Rob 23:519–529
111. Neri F, Cotta C (2011) Memetic algorithms and memetic optimization: a literature review. Swarm Evol Comput. doi:10.1016/j.swewo.2011.11.003
112. Ogren P, Winstrand M (2008) Minimizing mission risks in fuel constrained UAV path planning. AIAA J Guidance Control Dyn 31:1497–1500
113. Okabe AR, Boots B, Sugihara K, Chiu SN (2000) Spatial tessalations: concepts and applications of Voronoi diagrams. Wiley, Chichester
114. Patsko VS, Botkin ND, Kein VM, Turova VL, Zarkh MA (1994) Control of an aircraft landing in windshear. J Optim Theory Appl 83:237–267
115. Perez A, Platt R, Konidanis G, Kaelbling L, Lozano-Perez T (2012) LQG-RRT* optimal sampling based motion Planning with automatically derived extension heuristics. In: IEEE international conference on robotics and automation, 2537–2542
116. Perk B, Slotine JJ (2006) Motion primitives for robotic flight control. arXiv preprint cs/0609140, 2006 - arxiv.org
117. Pettersson PO, Doherty P (2004) Probabilistic road map based path planning for an autonomous unmanned aerial vehicle. In: Workshop on connecting planning theory with practice
118. Phillips JM, Bedrossian N, Kavraki LE (2004) Guided expansive space trees: a search strategy for motion and cost constrained state spaces. IEEE Int Conf Rob Autom 5:3968–3973
119. Piazzi A, Guarino Lo Bianco C, Romano M (2007) η^3 splines for the smooth path generation of wheeled mobile robot. IEEE Trans Rob 5:1089–1095
120. Plaku E, Hager GD (2010) Sampling based motion and symbolic action planning with geometric and differential constraints. In: IEEE international conference on robotics and automation, pp 5002–5008
121. Prats X, Puig V, Quevedo J, Nejjari F (2010) Lexicographic optimization for optimal departure aircraft trajectories. Aerosp Sci Technol 14, 26–37

122. Prats X, Santamaria E, Delgado L, Juillo N (2013) Enabling leg-based guidance on top of way-points based autopilots for UAS. Aerosp Sci Technol 24, 95–100
123. Rabier PJ, Rheinboldt WC (2000) Nonholonomic motion of rigid mechanical systems from a DAE viewpoint. SIAM press, Philadelphia
124. Revzen S, Ilhen D, Koditscheck D (20112) Dynamical trajectory replanning for uncertain environments. In: 51st IEEE Conference on decision and control, Hawaii, pp 3476–3483
125. Richards A, Schouwenaars T, How J, Feron E (2002) Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. AIAA J Guidance Control Dyn 25:755–764
126. Ross S, Melik-Barkhudarov N, Shankar KS, Wendel A (2012) Learning monocular reactive UAV control in cluttered natural, environments. arXiv:1211.1690 [cs.RO]
127. Rysdyk R (2007) Course and heading changes in significant wind. AIAA J Guidance Control Dyn 30:1168–1171
128. Saccon A, Hauser J, Aguiar A (2010) Optimal control on Lie groups: the projection operator approach. IEEE Trans Autom Control 58:2230–2245
129. Samar R, Rehman A (2011) Autonomous terrain following for unmanned air vehicles. Mechatronics 21:844–860
130. Santamaria E, Pastor E, Barrado C, Pratts X, Royo P, Perez M (2012) Flight plan specification and management for unmanned aircraft system. J Intell Rob Syst 67:155–181
131. Sastry S (1999) Nonlinear systems, analysis, stability and control. Springer
132. Schouwenaars T, Mettler B, Feron E (2004) Hybrid model for trajectory planning of agile autonomous vehicles. AIAA J Aeronaut Comput Inf Commun 12:466–478
133. Selig JM (1996) Geometric methods in robotics. Springer, New York
134. Shaffer PJ, Ross IM, Oppenheimer MW, Doman DB (2007) Fault tolerant optimal trajectory generator for reusable launch vehicles. AIAA J Guidance Control Dyn 30:1794–1802
135. Siciliano B, Sciavicco L, Villani L, Oriolo G (2009) Robotics, modelling, planning, and control. Springer, London
136. Simonin E, Diard J (2008) BBPRM: a behavior based probabilistic roadmap method. IEEE Int Conf Syst Man Cybern 1719–1724, doi:10.1109/CSMC.2008.4811536
137. Slegers N, Kyle J, Costello M (2006) Nonlinear model predictive control technique for unmanned air vehicles. AIAA J Guidance Control Dyn 29:5
138. Smith OJ, Boland N, Waterer H (2012) Solving shortest path problem with a weight constraint and replishnement arcs. Comput Oper Res 39:964–984
139. Sontag ED (1998) Mathematical control theory: Deterministic finite dimensional systems. Springer, New York
140. Soueres P, Laumond JP (1996) Shortest paths synthesis for a car-like robot. IEEE Trans Rob 41:672–688
141. Spooner JT, Maggiore M, Ordonez R, Passino KM (2002) Stable adaptive control and estimation for nonlinear systems: neural and fuzzy approximator techniques. Wiley, New York
142. Standley T, Korf R (2011) Complete algorithms for cooperative path finding problems. In: 22nd international joint conference on artificial intelligence, pp 668–673
143. Subchan S, Zbikowski R (2009) Computational optimal control: Tools and practice. Wiley, Chichester, UK
144. Sussmann HJ, Tang G (1991) Shortest paths for the reeds-shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. In: SYCON, rutgers center for systems and control
145. Sussmann J (1995) Shortest 3-dimensional paths with a prescribed curvature bound. In: 34th IEEE conference on decision and control, pp 3306–3312
146. Sussmann HJ (1987) A general theorem on local controllability. SIAM J Control Optim 25:158–195
147. Temizer S (2011) Planning under uncertainty for dynamic collision avoidance. PhD Thesis, MIT
148. Tsiotras P, Jung D, Bakolas E (2012) Multiresolution hierarchical path planning with small UAV using wavelet decomposition. J Intell Rob Syst 66:505–522

149. VanderBerg JP, Shah R, Huang A, Goldberg K (2011) ANA*: anytime Nonparametric A*. AAAI, paper presented at the" 25th AAAI conference on Artificial Intelligence
150. Virtanen K, Hamalainen RP, Mattika V (2006) Team optimal Signaling strategies in air combat. IEEE Trans Syst Man Cybern 36:643–660
151. Wang X, Wei G, Sun J (2011) Free knot recursive B spline for compensation of nonlinear smart sensors. Measurement 44:888–894
152. Watkins AS (2007) Vision based map building and trajectory planning to enable autonomous flight through urban environments. In: PhD Thesis, University of Florida, Gainesville
153. Watts R, Claus Christmann H, Johnson E, Fengl K (2012) Development and evaluation of an automated path planning aid. AIAA J Aircr 49:1774–1785
154. Webb D, VandenBerg J (2012) Kinodynamic RRT*: optimal motion planning for systems with liner differential constraints. arXiv preprint arXiv:1205.5088, 2012 - arxiv.org
155. Wie B (1998) Space vehicle dynamics and control. AIAA education series, AIAA Press, Reston, Va
156. Wilton D, Rao S, Glisson A (1984) Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains. IEEE Trans Antennas Propag 32:276–281
157. Wu A, How J (2012) Guaranteed infinite horizon avoidance of impredictable dynamically constrained obstacles. Auton Rob 32:227–242
158. Yakimenko OA (2000) Direct method for rapid prototyping of near optimal aircraft trajectory. AIAA J Guidance Control Dyn 23:865–875
159. Yokoyama N (2013) Path generation algorithm for turbulence avoidance using real-time optimization. AIAA J Guidance Control Dyn, 36, 2500–262
160. Zhou Z, Takei R, Huang H, Tomlin C (2012) A general open loop formulation for reach-avoid games. In: 51st IEEE Conference on decision and control, Hawaii, pp 6501–6506

http://www.springer.com/978-3-319-03706-6

Planning and Decision Making for Aerial Robots

Bestaoui Sebbane, Y.

2014, XX, 406 p. 17 illus., 14 illus. in color., Hardcover
ISBN: 978-3-319-03706-6