Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2020

Robust autonomous landing of fixed-wing UAVs in wind

Tobias Fridén

Master of Science Thesis in Electrical Engineering

Robust autonomous landing of fixed-wing UAVs in wind:

Tobias Fridén

LiTH-ISY-EX--YY/NNNN--SE

Supervisor: **Jonatan Olofsson**
ISY, Linköpings universitet
Kristoffer Bergman
ISY, Linköpings universitet
Fredrik Falkman
Airpelago

Examiner: **Daniel Axehill**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

If your thesis is written in English, the primary abstract would go here while the Swedish abstract would be optional.

Contents

Notation

Notation

Symbol	Meaning
$\mathbb{R}$	The real numbers
$\boldsymbol{w}$	Wind velocity
(W, ψ_w)	Wind speed and direction
$\boldsymbol{v}$	Velocity relative to the earth
$\boldsymbol{v}_a$	Velocity relative to the air
V_a	Speed relative to the air

1 Introduction

1.1 Background

Unmanned Aerial Vehicles (UAVs) have many different applications, both in commercial usecases such as construction and agriculture, but also in emergency response and personal use. UAVs are often primarily divided into two subclasses, multirotors and fixed-wing UAVs. While both types of UAVs are becoming more and more autonomous through various research efforts, landing a fixed-wing UAVs remains a challenging task which normally requires manual input from an experienced pilot. For small and light-weight UAVs, the presence of wind also acts as a major disturbance which needs to be taken into account when planning the landing sequence.

1.2 Scope

This section describes the overall scope of this thesis, the method used as well as where limitations have been made.

1.2.1 Problem formulation

The aim of this thesis is to develop a method for automatically generating feasible landing procedures for fixed-wing UAVs, in the presence of wind. The landing sequence should be able to take the UAV from an arbitrary initial position and land safely in a predefined landing area while fulfilling physical constraints of the system. This thesis aims to answer the following questions:

1. How can sampling-based motion planning techniques be used to generate landing sequences for fixed-wing UAVs?

2. How can wind effects be taken into account when computing safe landing sequences?

1.2.2 Method

The method proposed in this thesis consists of two main components, the *landing sequence calculation* and *motion planner*. The goal of the landing sequence calculation is to calculate a landing sequence which lands the UAV safely in a predefined landing area $\mathcal{A}$. When the landing sequence is calculated, the goal of the motion planner is to calculate a sequence of waypoints $\mathcal{M}$ which, when executed with the tracking controller of the UAV, takes it from a starting location x_0 to a position where the landing sequence can be executed.

Both these components must take the current wind $\boldsymbol{w}$ as well as any obstacles around the landing area into account. The wind is estimated using a wind estimation system onboard the UAV, while obstacles are stored in an obstacle database $\mathcal{X}_{\text{obst}}$. Finally, a positioning system onboard the UAV is used to determine the starting position sent to the motion planner. An overview of the different components and their relationship is shown in Figure 1.1.

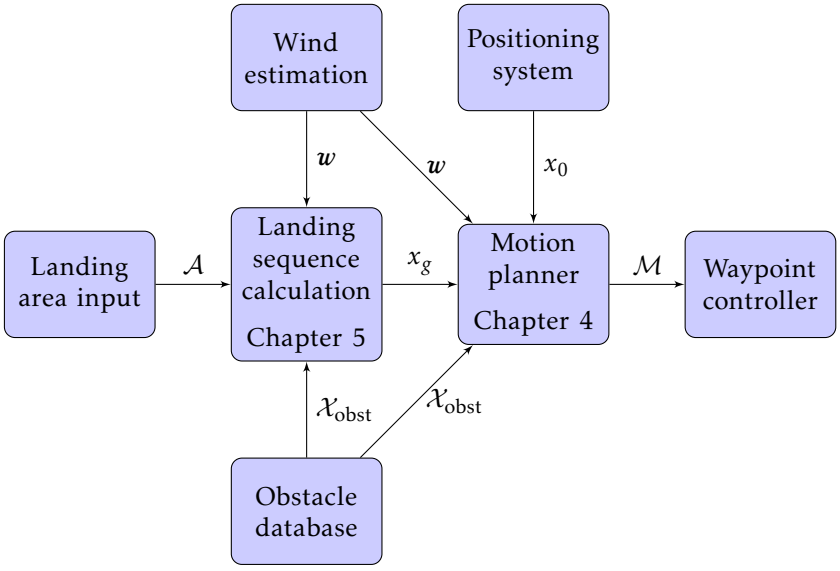


Figure 1.1: *An overview of the different components in the proposed method.*

1.2.3 Limitations

The components of a general autonomous UAV are illustrated in Figure 1.2. The work in this thesis is mainly focused on the motion planning component. However, the tracking controllers and properties of the actual UAV have to be taken

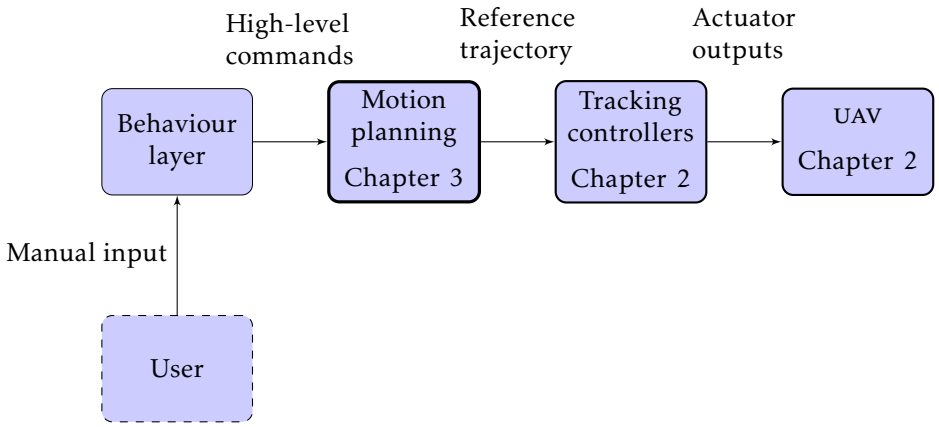


Figure 1.2: *Components of a general autonomous UAV. The main focus of this thesis is motion planning, but modeling of the tracking controllers and UAV is also discussed.*

into account to ensure feasibility of the generated path. Furthermore, the main focus of this thesis is UAVs using the ArduPilot open source autopilot [33], since the implementation relies on the trajectory controller implemented in ArduPilot.

A large part of this thesis is concerned with the analysis of wind, which is assumed to be constant in space and time. The wind is defined as a vector with magnitude W and direction ψ_w. The wind magnitude W will also be referred to as wind speed and ψ_w as wind direction.

1.3 Related work

The following section presents previous work that is relevant to the subject of this thesis.

1.3.1 Motion planning

Motion planning refers to the task of finding a feasible path between an initial state and a goal state for a given system. Since this is an important component of autonomous systems it has received increasing research interest during the past decades, with an array of different algorithms and methods available.

Sampling-based motion planning

Many motion planning techniques are based on discrete sampling of the continuous state and action space. These methods are either based on random sampling, such as in Probabilistic Roadmaps [21] or Rapidly Exploring Random Trees [25], while others such as Hybrid A^* [14] use deterministic sampling.

In [20] the A^* algorithm [19] is used to find kinematically feasible trajectories for fixed-wing UAVs with a maximum turn rate while avoiding obstacles. The feasibility of the resulting path is ensured by aligning the dimensions of the sampled grid with model parameters of the given UAV.

In [41] the results in [42] are used together with A^* to generate time-optimal trajectories in the presence of wind, while also avoiding obstacles. They further use the results in [29] to define a modified heuristic function which takes wind into account.

An RRT-based motion planning framework for fixed-wing UAVs, with constraints on both arrival time and final direction is proposed in [27].

Optimal control approach

The problem of finding time-optimal paths for fixed wing UAVs in uniform winds is often used to formulate an optimal control problem which has been studied by different authors. In these works the following kinematic model is used:

$$\dot{x} = f(x, u) = \begin{bmatrix} V_a \cos\psi + W \cos\psi_w \\ V_a \sin\psi + W \sin\psi_w \\ u \end{bmatrix} \tag{1.1}$$

where the input u is the rate of turn which is constrained such as $|u| \leq \dot{\psi}_{\max}$ [29][42].

An important result used in [29] is that earth-fixed goal states become non-stationary in a coordinate frame relative to the air. This is usedto reformulate the problem as finding a path which intersects a virtual target moving from x_g with the same speed as the wind but in the opposite direction. It is shown that in most cases, the shortest path relative to the earth corresponds to an air-relative shortest path which can be found analytically. In some cases however, a non-optimal path relative to the air is required to intercept the target. A general solution which uses root-finding techniques to cover both cases is also presented.

The approach in [42] is based on the observation from [40] that constant rate of turn paths in the air-relative frame correspond to *trochoidal* paths in the inertial frame. A trochoid is the path followed by a fix point on a circle which rolls along a line. They further show that there exists an analytical solution to compute some of the optimal-path candidates, but to find all possible optimal paths a non-polynomial equation has to be solved on a two dimensional grid which is computationally expensive.

1.3.2 Landing approaches

The problem of autonomously landing fixed-wing UAVs in different settings has been studied by several authors. In many of these works, the problem is defined as landing the UAV on a runway. A survey of different landing techniques is given in [17]. In [44] a framework is proposed for emergency landing of fixed-wing UAVs during thrust-lost and uniform wind. The motion planner in this work is

based on the trochoidal paths discussed in [42]. The problem of landing fixed-wing UAVs on a moving ground vehicle is studied in *e.g.* [36].

1.4 Outline

Chapter 2 introduces general concepts regarding fixed-wing UAVs and wind, as well as the kinematic models and controllers studied in this thesis. Chapter 3 gives an overview of motion planning theory. In Chapter 4 a motion planning method for fixed-wing UAVs flying in wind is proposed. Chapter 5 describes the landing sequence of a fixed-wing UAV and how landing parameters can be calculated while fulfilling a set of given constraints. Chapter 6 presents the implementation of the proposed method on a real UAV platform, as well as experimental results from both simulated and real flight experiments. Finally, these results are discussed and summarized in Chapter 7.

2

Modeling and control of fixed-wing UAVs in uniform wind

2.1 Definitions and terminology

Fixed-wind UAVs are receiving increasing commercial and research interest, and offer a number of advantages in many usecases. In the following sections a thorough description of general fixed-wing kinematics and control as well as a description of the specific platform used in this work will be presented. First some common definitions and terminology which will be used throughout this thesis are established. These definitions are used in many other works related to fixed-wing aircraft, such as [11].

2.1.1 Coordinate reference frames

For UAV applications in wind, there are mainly four different coordinate frames that are relevant to consider: the inertial frame which is fixed in the earth, the air-relative frame, the body frame and the wind reference frame. The body frame and wind reference frames are related through the *angle of attack* α and *sideslip* β as shown in Figure 2.1.

Definition 2.1 (Inertial frame). The earth-fixed frame, which for the purposes in this thesis can be considered inertial, is denoted with subscript I. A position vector in the inertial frame is defined in the North East Down (NED) order as

$$\boldsymbol{p}_I = (x_N, y_E, -h) \tag{2.1}$$

where x_N points in the north direction, y_E points east and h is the altitude above the ground, in order to form a right-handed positive coordinate system.

Definition 2.2 (Air frame). The air frame, denoted with subscript A, is fixed at any point in the air and aligned with the current direction of wind. In the case

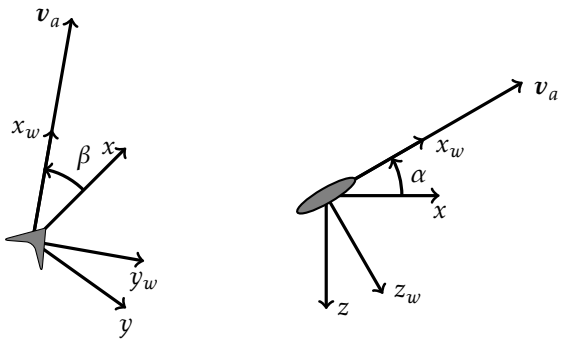


Figure 2.1: *Relation between body and wind frames*

of non-zero wind, this coordinate frame moves with the same speed as the earth-relative wind at the frames origin. This means that inertial frame coordinates become time-dependent in the air frame, and are given by

$$x_{N,A}(t) = \cos\psi_w x_{N,I} + \sin\psi_w y_{E,I} - Wt \tag{2.2}$$

$$y_{E,A}(t) = -\sin\psi_w x_{N,I} + \cos\psi_w y_{E,I} \tag{2.3}$$

where W is the wind speed and ψ_w the wind direction.

Definition 2.3 (Body frame). The body frame, denoted with subscript B is fixed in the UAV center of gravity. A position vector in the body frame is defined as

$$\boldsymbol{p}_B = (x, y, z) \tag{2.4}$$

where x points forward through the UAV, y points to the right and z points down.

Definition 2.4 (Wind reference frame). The wind reference frame, denoted with subscript W is related to the current direction of motion through the air. A position vector in the wind reference frame is defined as

$$\boldsymbol{p}_W = (x_w, y_w, z_w) \tag{2.5}$$

where x_w points in the same direction as the velocity through the air $\boldsymbol{v}_a$, y_w points to the right of x_w and z points down relative x_w and y_w.

2.1.2 Attitude representation

The attitude of the UAV is represented by the *Euler angles*.

Definition 2.5 (Euler angles). The Euler angle vector is defined as

$$\boldsymbol{\Phi} = (\phi, \theta, \psi) \tag{2.6}$$

where the *roll angle* ϕ is rotation around the north inertial axis, the *pitch angle* θ is rotation around the east inertial axis and the *yaw angle* ψ is rotation around the downwards inertial axis. The yaw angle is often denoted *heading* in this thesis.

Figure 2.2: *UAV platform used for real flight experiments*

The relationship between coordinates in the body frame and inertial frame is given in [18] as the rotation matrix

$$\mathcal{R}_B^I = \mathcal{R}_\phi^x \mathcal{R}_\theta^y \mathcal{R}_\psi^z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

This attitude representation is not uniquely defined for $\theta = \pm\pi/2$. However, this is not an issue since such angles never occur in the situations studied in this thesis.

2.1.3 Fixed-wing UAV

A fixed-wing UAV is equipped with two horizontal wings that are fixed in the body frame. In order to stay in the air, the forward velocity must be above a certain threshold, *i.e.*

$$V > V_s \tag{2.8}$$

where V_s is the airframe-dependent *stall speed*. In order to navigate through the air, it is equipped with some or all of the following control surfaces:

- *Ailerons* to primarily control ϕ
- *Elevators* to primarily control θ
- *Rudders* to primarily control ψ

The UAV is also equipped with one or several engines that are used to create the thrust which increases the total energy of the system. In the case of a propeller-equipped aircraft, these might be facing towards or against the direction of motion. The UAV platform used during real flight experiments in this work consists of a modified Parrot Disco airframe which is shown in Figure 2.2.

2.2 Wind field definition

The wind field is commonly defined as a time and spatially dependent vector field [24]

$$\boldsymbol{w}(x_N, y_E, h, t) = \begin{bmatrix} w_N(x_N, y_E, h, t) \\ w_E(x_N, y_E, h, t) \\ w_H(x_N, y_E, h, t) \end{bmatrix} \tag{2.9}$$

In this thesis, the vertical component w_H will be neglected and the wind velocity is written as

$$\boldsymbol{w} = W \begin{bmatrix} \cos \psi_w \\ \sin \psi_w \end{bmatrix} \tag{2.10}$$

where W is the wind magnitude and ψ_w is the wind direction. As mentioned in Section 1.2.3 the wind is assumed constant and the dependencies on time and location are thus removed. The wind field can be decomposed as

$$\boldsymbol{w} = \bar{\boldsymbol{w}} + \boldsymbol{w}_s \tag{2.11}$$

where $\bar{\boldsymbol{w}}$ is the mean wind field and $\boldsymbol{w}_s$ is described by some stochastic process. The random component is not considered in this thesis, *i.e.* it is assumed that $\boldsymbol{w} = \bar{\boldsymbol{w}}$.

2.3 Wind estimation

Wind field estimation techniques are important in order to handle the effects of winds on both planning and control of UAVs.

2.3.1 Direct computation of wind field

When flying in non-zero wind, the resulting velocity relative to the earth is dependent on the velocity of the UAV relative to the air $\boldsymbol{v}_a$ as well as the wind velocity $\boldsymbol{w}$. This relationship, which is sometimes denoted the *wind triangle* is illustrated in Figure 2.3. The velocity relative to the earth is given by

$$\boldsymbol{v} = \boldsymbol{v}_a + \boldsymbol{w} = V_a \begin{bmatrix} \cos \psi \\ \sin \psi \end{bmatrix} + W \begin{bmatrix} \cos \psi_w \\ \sin \psi_w \end{bmatrix} \tag{2.12}$$

and if it can be measured, *e.g.* with the Global Positioning System (GPS) of the UAV, the wind vector can be computed directly as

$$\boldsymbol{w} = \boldsymbol{v} - \boldsymbol{v}_a \tag{2.13}$$

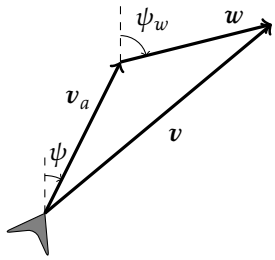


Figure 2.3: *Relationship between velocities relative to the air and the earth*

where $\boldsymbol{v}_a$ can be measured as described in Section 2.3.3. Assuming level flight, *i.e.* $\theta \approx 0$, it is shown in [24] that the measurement error is

$$e^2 = \sigma_{\dot{x}_N}^2 + \sigma_{\dot{y}_E}^2 + \sigma_{\dot{z}_H}^2 + \sigma_{V_a}^2 + V_a^2(\sigma_\theta^2 + \sigma_\alpha^2 + \sigma_\beta^2 + \sigma_\psi^2) \tag{2.14}$$

if this method is used. In standard unaided GPS systems, the measurement error is approximately 0.1 m/s. Assuming the measurement error of V_a is 0.2 m/s and angles can be measured up to 1° precision, the error becomes $e^2 = 0.07 + 0.0012V_a^2$. If $V_a = 16$ m/s this corresponds to a measurement error of $e = 0.61$ m/s.

2.3.2 Estimation using Extended Kalman Filter

A more robust approach is to use an Extended Kalman Filter (EKF) to measure vehicle states. These are commonly used in autonomous systems to fuse measurements from many different sensors such as a GPS, Inertial Measurement Unit (IMU) and barometer. A thorough reference on the underlying theory of EKFs is given in [18].

2.3.3 Airspeed measurement

Fixed-wing UAVs are often equipped with a *pitot-tube* sensor to measure $V_a = \|\boldsymbol{v}_a\|$. Such a sensor consists of a metallic tube together with a sensor which measures the dynamic pressure ΔP of the air which flows through the tube. This measurement is related to the velocity of the air flow V_{pitot} through Bernoulli's Equation as

$$V_{\text{pitot}}^2 = K\frac{2\Delta P}{\rho} \tag{2.15}$$

where ρ is the density of the air. K is a correction factor which compensates for calibration errors and generalizations such as assuming a perfect gas and constant temperature. Assuming that the sensor is mounted along the x axis in the body frame, the relationship between V_a and V_{pitot} is given in [12] as

$$V_a^2 = \frac{V_{\text{pitot}}^2}{\cos\alpha\cos\beta} = \frac{2K\Delta P}{\rho\cos\alpha\cos\beta} \tag{2.16}$$

2.4 Trajectory following

To allow autonomous operation of an UAV, it needs to be equipped with a trajectory following controller. The goal of this controller is to follow a pre-defined trajectory, which is defined by a set of linear or curved segments in the inertial frame. This goal can be formulated as calculating the control signal in each time-step which minimizes the *cross-track error*

$$d(t) = \min \|\boldsymbol{p}_{I,\text{UAV}}(t) - \boldsymbol{p}_{I,\text{traj}}\| \tag{2.17}$$

where $\boldsymbol{p}_{I,\text{traj}}$ is any point on the trajectory.

2.4.1 Kinematic model

A kinematic model for fixed-wing UAV trajectory following in wind is introduced in [11] as:

$$\dot{x}_N = V_a \cos\psi + W \cos\psi_w \tag{2.18}$$
$$\dot{y}_E = V_a \sin\psi + W \sin\psi_w \tag{2.19}$$
$$\dot{\psi} = \frac{g}{V_a} \tan\phi \tag{2.20}$$

where V_a is the speed relative to the air, ψ is the heading and ϕ is the roll angle, both relative to the inertial frame. Dynamics in the roll angle ϕ can be included as

$$\dot{\phi} = f_\phi(\phi - \phi_{\text{cmd}}) \tag{2.21}$$

where f_ϕ is defined by the inner loop roll controller of the UAV and ϕ_{cmd} is the roll-angle command by the trajectory following controller.

2.4.2 Straight path following in wind

To study the problem of accurately following a straight path segment, it is helpful to introduce another coordinate frame which is aligned with the path segment to follow, as shown in Figure 2.4. The velocity vector relative to this frame $\boldsymbol{v}_s$ is related to $\boldsymbol{v}$ as

$$\boldsymbol{v}_s = \begin{bmatrix} \cos\psi_s & \sin\psi_s \\ -\sin\psi_s & \cos\psi_s \end{bmatrix} \boldsymbol{v} \tag{2.22}$$

which implies that

$$\dot{d} \equiv \dot{y}_{E,s} = V_a \sin(\psi - \psi_s) + W \sin(\psi_w - \psi_s) \tag{2.23}$$

Assuming that $d = 0$ and $\dot{d} = 0$,

$$V_a \sin(\psi - \psi_s) + W \sin(\psi_w - \psi_s) = 0 \tag{2.24}$$

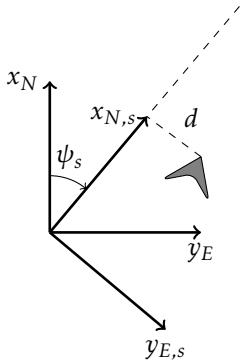


Figure 2.4: *Coordinate frame for straight path following*

The cross-track error is thus minimized when

$$\psi = \psi_{\text{wca}} \equiv -\arcsin\left(\frac{W}{V_a}\sin(\psi_w - \psi_s)\right) + \psi_s \tag{2.25}$$

In the case of $W = 0$, this simplifies to $\psi_{\text{wca}} = \psi_s$. In windy conditions however, the wind has to be compensated with a constant offset which depends on wind speed, direction and desired heading ψ_s. The angle ψ_{wca} is called the *wind correction angle* [11].

2.4.3 Relationship between course and heading

Since the direction of travel relative to the air and the ground generally differ, the concept of course can instead be introduced.

Definition 2.6 (Course). The Course Over Ground (COG) is defined as

$$\psi_{\text{cog}} = \text{atan2}(V_E, V_N) \tag{2.26}$$

where

$$V_N = V_a \cos\psi + W\cos\psi_w \tag{2.27}$$

and

$$V_E = V_a \sin\psi + W\sin\psi_w \tag{2.28}$$

i.e. the north and east components of $\boldsymbol{v}$.

2.5 ArduPlane autopilot

The ArduPlane autopilot is an open source autopilot for fixed-wing UAVs [33]. It contains high-level controllers for navigation, velocity and altitude control as well as low level logic to command the attitude and throttle of the vehicle. In the following section the underlying theory of the components relevant for this thesis will be presented.

2.5.1 Wind estimation

The ArduPlane autopilot uses an EKF to estimate $\boldsymbol{w}$. The implementation estimates 24 different states such as attitude, velocity, position, sensor biases and wind. The different process models and measurement equations are presented in [39].

2.5.2 Trajectory controller

The ArduPlane autopilot uses the L_1 controller described in [34] for trajectory following. The goal of this controller is to follow a straight line from a start coordinate $\boldsymbol{p}_0$ to a goal coordinate $\boldsymbol{p}_1$. This is obtained by aiming towards a point P which is located at a fixed distance L_1 from the UAV. The logic behind the controller is illustrated in Figure 2.5, where $\boldsymbol{p}$ is the UAV position and ψ is the UAV heading.

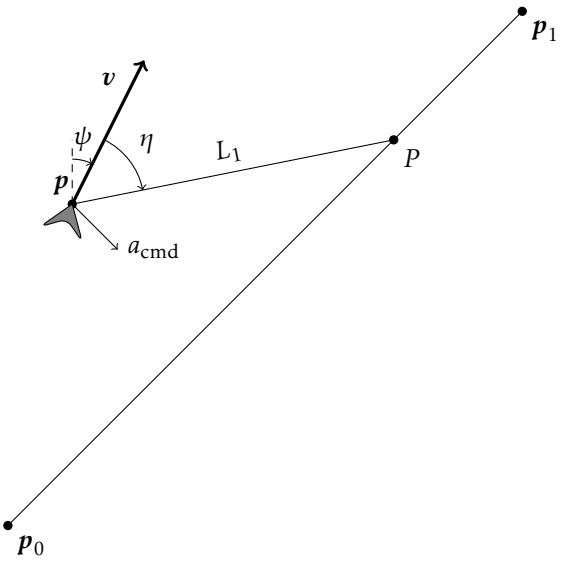


Figure 2.5: *L_1 controller logic*

In the ArduPilot implementation, the distance L_1 is calculated as

$$L_1 = \begin{cases} \frac{1}{\pi}\zeta \Delta T V & \text{if} \quad |\frac{1}{\pi}\zeta \Delta T V| > |\boldsymbol{p}_1 - \boldsymbol{p}| \\ |\boldsymbol{p}_1 - \boldsymbol{p}| & \text{otherwise} \end{cases} \tag{2.29}$$

where $V = |\boldsymbol{v}|$, ζ is the damping factor and ΔT is the update period of the controller [34]. Using the speed relative to the inertial frame compensates for wind effects. In each time step, this controller corresponds to following a circular seg-

ment with radius

$$R = \frac{L_1}{2 \sin \eta} \tag{2.30}$$

which is tangent to $\boldsymbol{v}$ in $\boldsymbol{p}$. η is defined as the angle between the UAV velocity vector $\boldsymbol{v}$ and the line from the UAV to P. By introducing a line parallel to the line-segment from $\boldsymbol{p}_0$ to $\boldsymbol{p}_1$ the angle η can be decomposed as

$$\eta = \eta_1 + \eta_2 \tag{2.31}$$

where η_2 is defined as the angle from the velocity vector $\boldsymbol{v}$ to this line. The angle components are given by

$$\eta_2 = \text{atan2}(V_E \cos \psi_s - V_N \sin \psi_s, V_N \cos \psi_s + V_E \sin \psi_s) \tag{2.32}$$

and

$$\eta_1 = \arcsin \left(\frac{x_N \sin \psi_s - y_E \cos \psi_s}{L_1} \right) \tag{2.33}$$

where ψ_s is the direction defined by the line from $\boldsymbol{p}_0$ to $\boldsymbol{p}_1$. Finally, the circular segment is followed by issuing a lateral acceleration command

$$a_{\text{cmd}} = 2 \frac{V^2}{L_1} \sin \eta \tag{2.34}$$

The lateral acceleration command is translated to a roll command

$$\phi_{\text{cmd}} = \arctan(a_{\text{cmd}}/g) \tag{2.35}$$

where g is the gravitational constant. The low-level attitude controller is then used to track the desired roll.

In the case of a straight reference trajectory, it is shown in [35] that (2.34) is well approximated by its linearization

$$a_{\text{cmd}} \approx 2 \frac{V}{L_1} \left(\dot{d} + \frac{V}{L_1} d \right) \tag{2.36}$$

which is a PD-controller. Furthermore, if inner-loop dynamics are neglected and $\boldsymbol{v}$ is assumed to be parallel to the reference line, $a_{\text{cmd}} \approx \ddot{d}$ and

$$\ddot{d} + 2\zeta \omega_n \dot{d} + \omega_n^2 d = 0 \tag{2.37}$$

with $\zeta = 1/\sqrt{2}$ and $\omega_n = \sqrt{2} V / L_1$. This is a simple second-order system where the damping is constant, and the speed depends on the ratio between V and L_1.

2.5.3 Mission representation and flight modes

A *mission* $\mathcal{M}$ is defined as

$$\mathcal{M} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\} \tag{2.38}$$

i.e. an ordered sequence of *n waypoints* represented as

$$\boldsymbol{p} = (x_N, y_E, h_{\text{rel}}, c_{\text{wp}}) \tag{2.39}$$

where h_{rel} is the altitude relative to the takeoff position and c_{wp} represents the waypoint command. There are many different waypoint commands available in ArduPlane, but this work will be focused on

$$c_{\text{wp}} \in \{\text{Waypoint}, \text{Takeoff}, \text{Land}\} \tag{2.40}$$

Waypoint mode

In *waypoint* mode the trajectory following controller is used to navigate along the line from $\boldsymbol{p}_0$ to $\boldsymbol{p}_1$. When $\boldsymbol{p}_1$ is reached, the flight mode is updated depending on the next c_{wp}. A waypoint is assumed to have been reached when

$$\|\boldsymbol{p}_{\text{rel}} - \boldsymbol{p}_{\text{wp}}\| < R_{\text{wp}} \tag{2.41}$$

where R_{wp} is defined by the user, or passed when

$$\frac{\|\boldsymbol{p}_{\text{rel}} \cdot \boldsymbol{p}_{\text{wp}}\|}{\|\boldsymbol{p}_{\text{wp}}\|} \geq 1 \tag{2.42}$$

where $\boldsymbol{p}_{\text{rel}} = \boldsymbol{p} - \boldsymbol{p}_0$ and $\boldsymbol{p}_{\text{wp}} = \boldsymbol{p}_1 - \boldsymbol{p}_0$ [30].

Land mode

In *Land* mode, the plane will attempt to land at a given coordinate. The landing approach is divided into two different stages, the *approach* stage and *flare* stage.

During the approach stage, the UAV tries to accomplish the commanded *glide slope*, which is dependent on the previous waypoint position relative to the landing point. When the altitude decreases below h_{flare}, it enters the flare stage which means the throttle is completely turned off. During this stage the UAV will try to hold a target descent rate $\dot{h}_{\text{flare}}$ which is defined by the user [31].

3

Motion planning theory

3.1 Definitions and terminology

Motion planning is defined as the task of finding a path from a starting state to a goal state which fulfills a given set of constraints, while minimizing or maximizing some performance measure. These constraints might include differential constraints of the system and obstacle avoidance among others. Common performance measures include minimal time or minimal energy required. To introduce this chapter general definitions and terminology that are used to describe motion planning in this thesis are introduced. A thorough description of motion planning theory is found in [23].

3.1.1 Graph terminology

First the mathematical concept of graphs is introduced, following the definitions in [10].

Definition 3.1 (Graph). A *graph* is defined as a set $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ where $\mathcal{V}$ are the *vertices* of the graph and $\mathcal{E}$ are the *edges*. Two vertices $v_i, v_j \in \mathcal{V}$ where $i \neq j$ might be connected by an edge $e_{i,j} \in \mathcal{E}$ or not connected.

Definition 3.2 (Weighted graph). In a *weighted graph*, each edge is assigned a cost $C(e) \in \mathrm{R}$.

Definition 3.3 (Directed graph). In a *directed* graph, it is possible that $c_{i,j} \neq c_{j,i}$ and there might not be an edge $e_{j,i}$ even if $e_{i,j} \in \mathcal{E}$.

Definition 3.4 (Walk). A *walk* in a weighted and possibly directed graph is defined as a set of vertices $\mathcal{V}_p \subseteq \mathcal{V}$ and edges $\mathcal{E}_p \subseteq \mathcal{E}$ where the vertices in $\mathcal{V}_p$ are connected by the edges in $\mathcal{E}_p$.

Definition 3.5 (Path). A *path* in a weighted and possibly directed graph is defined as a walk where each edge and vertex occurs only once. The total *cost* of a path is defined as

$$C(\mathcal{V}_p, \mathcal{E}_p) = \sum_{e \in \mathcal{E}_p} C(e) \tag{3.1}$$

3.1.2 Motion planning terminology

Some common terms used in motion planning are also defined.

Definition 3.6 (State and action spaces). The *state space* $\mathcal{X}$ and *action space* $\mathcal{U}$ are defined as the set of obtainable states x and available actions u for the studied system. $\mathcal{X}$ can be further divided into

$$\mathcal{X} = \mathcal{X}_{\text{free}} + \mathcal{X}_{\text{obst}} \tag{3.2}$$

where $\mathcal{X}_{\text{obst}}$ are states which contain some kind of obstacle.

Definition 3.7 (Motion plan). A motion plan is defined as a sequence of states $\{x(t_0), \ldots, x(t_n)\}$ such that

$$x(t) \in \mathcal{X}_{\text{free}}, \quad t \in [t_0, t_n] \tag{3.3}$$

and actions $\{u(t_0), \ldots, u(t_n)\}$ such that

$$u(t) \in \mathcal{U}, \quad t \in [t_0, t_n] \tag{3.4}$$

which takes the system from a specified initial state $x(t_0) = x_0$ to a goal state $x(t_n) = x_g$ while fulfilling

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(x(t), u(t)) dt \tag{3.5}$$

where $f(x(t), u(t))$ is called the *transition function* [23]. The time dependencies of x and u will henceforth be omitted for brevity.

3.1.3 Differential constraints

The transition function $f(x, u)$ introduces differential constraints which restrict the set of possible actions and states that the system can obtain. An important class of systems under differential constraints are *non-holonomic* systems.

Definition 3.8 (Non-holonomic system). In a *non-holonomic* system, the current state $x(t)$ is dependent on the order in which the actions $u(t_i), \quad t_i < t$ were performed.

A formal definition, and extensive discussion of non-holonomic systems, is given in [23, Chapter 15]. Systems only capable of motion in a direction dependent on the current state, such as cars and fixed-wing UAVs belong to this class of systems.

3.2 Sampling-based motion planning

Both $\mathcal{X}$ and $\mathcal{U}$ are continuous, and generally need to be discretized before motion planning techniques can be applied. These discretized subsets are henceforth denoted $\mathcal{X}_s$ and $\mathcal{U}_s$. This means that the resulting path will only be *resolution complete*, *i.e.* the solution will depend on the sampling resolution. In sampling-based motion planning a *reachability graph* is commonly used [23].

Definition 3.9 (Reachability graph). Given a starting state $x_0(t_0) \in \mathcal{X}_s$, the *reachable set* $R(x_0, \mathcal{U}_s)$ is defined as the set of states which are reached by applying any action $u \in \mathcal{U}_s$. By incrementally calculating the reachable set for each $x \in R(x_0, \mathcal{U}_s)$ a *reachability tree* $\mathcal{T}_r(x_0, \mathcal{U}_s)$ is created. The reachability tree is a directed graph where each vertex consists of a state x which is reachable from x_0 by applying some action sequence $\{u_1, \ldots, u_n\} \in \mathcal{U}_s$. By pruning any duplicate states from $\mathcal{T}_r$ it is finally transformed to the *reachability graph* $\mathcal{G}_r(x_0, \mathcal{U}_s)$

3.2.1 Forward simulation

The next state in $\mathcal{G}_r$ given a specified input action u is obtained by integrating the transition function $f(x, u)$ on $[0, \Delta t]$. In practice this integral is calculated using some numerical approximation method. A common choice is the fourth-order *Runge-Kutta integration method*, which is defined in [23] as

$$x(\Delta t) \approx x(0) + \frac{\Delta t}{6}(w_1 + 2w_2 + 2w_3 + w_4) \tag{3.6}$$

where

$$\begin{aligned} w_1 &= f(x(0), u) \\ w_2 &= f(x(0) + \frac{1}{2}\Delta t w_1, u) \\ w_3 &= f(x(0) + \frac{1}{2}\Delta t w_2, u) \\ w_3 &= f(x(0) + \Delta t w_3, u) \end{aligned} \tag{3.7}$$

3.2.2 Motion primitives

Often it is neither feasible nor desirable to sample from all possible actions in $\mathcal{U}_s$. A common method is to instead create a set of *motion primitives* $\mathcal{P}$ which consists of sequences of actions that take the system from desired initial and final states [37].

Maneuver-based motion primitive generation is introduced in [9] as the method of generating $\mathcal{P}$ based on a fixed set of maneuvers. One such maneuver is *heading change*, which is defined as taking the system from an initial heading ψ_0 to a final heading ψ_g. This primitive set can be generated by solving the optimal control

problem

$$\min_{x(t),u(t),T} \quad J = \Phi(x(T), T) + \int_0^T l(x(t), u(t))dt \tag{3.8a}$$

$$\text{subject to} \quad \psi(0) = 0, \quad \psi(T) = \Delta\psi \tag{3.8b}$$

$$\dot{x} = f(x(t), u(t)) \qquad t \in [0, T] \tag{3.8c}$$

$$x(t) \in \mathcal{X} \qquad t \in [0, T] \tag{3.8d}$$

$$u(t) \in \mathcal{U} \qquad t \in [0, T] \tag{3.8e}$$

where $\Delta\psi = \psi_g - \psi_0$ and the performance metrics $\Phi(x(T), T)$ and $l(x(t), u(t))$ are chosen such that a desired property, such as required energy or time is minimized. The motion primitive set $\mathcal{P}$ then consists of the solutions of (3.8) for different values of $\Delta\psi$.

3.3 Graph search methods

The problem of finding the minimum cost path between two vertices in a graph $\mathcal{G}$ is well studied, and there are numerous algorithms for solving it. These algorithms require that $C(e) \geq 0 \forall e \in \mathcal{E}$. By using such algorithms together with the concept of reachability graphs defined in (3.9) resolution-optimal motion plans can be calculated [9].

3.3.1 A-star search

The A^* algorithm was introduced in [19] and is widely used to find the shortest path in graphs. Two important components of this algorithm is the *cost-to-come* $g(x)$ and *cost-to-go* or *heuristic* $h(x, x_g)$. $g(x)$ is defined as the cost of the shortest path from the starting state x_0 to x, and $h(x, x_g)$ as the cost of the shortest path from x to the goal x_g. Thus for any state x the total cost for a path through this state to the goal is given as $g(x) + h(x, x_g)$. A perfect heuristic would be the actual cost from each initial state to the goal state. Since this is generally not known an approximate heuristic $\tilde{h}(x, x_g)$ has to be used. Two necessary condition for optimality of the resulting path is that $\tilde{h}(x, x_g)$ is *admissible* and *consistent*, as defined below.

Definition 3.10 (Admissible heuristic). A heuristic function $\tilde{h}(x, \tilde{x})$ is *admissible* if

$$\tilde{h}(x, \tilde{x}) \leq h(x, \tilde{x}) \quad \forall x \tag{3.9}$$

where $h(x, \tilde{x})$ is the true cost-to-go from x to $\tilde{x}$.

Definition 3.11 (Consistent heuristic). A heuristic function $\tilde{h}(x, \tilde{x})$ is *consistent* if

$$\tilde{h}(x, \tilde{x}) \leq h(x, \hat{x}) + \tilde{h}(\hat{x}, \tilde{x}) \tag{3.10}$$

where $h(x, \tilde{x})$ is the true cost-to-go from x to $\tilde{x}$, for all $(x, \hat{x}, \tilde{x}) \in \mathcal{X}$.

An outline of motion planning with A^* is presented in Algorithm 1. The function EXPAND$(x, \mathcal{P})$ returns all states reached from x by applying motion primitives in $\mathcal{P}$ and the associated cost of each primitive. The function POP$(\mathcal{O})$ returns the state in the open set with the lowest estimated total cost, and CURRENT_COST$(x, \mathcal{O})$ returns the estimated total cost currently stored for x.

Algorithm 1 A^* based motion planning

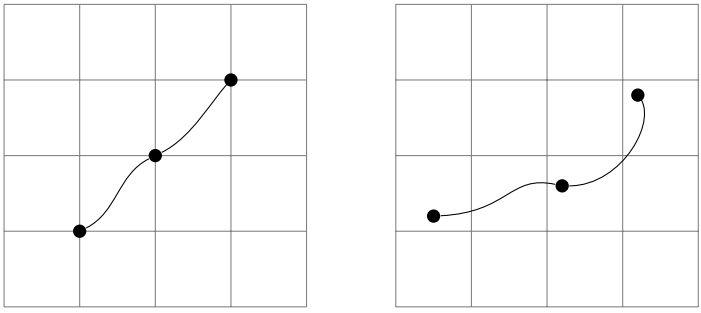
Require: Motion primitive set $\mathcal{P}$, valid states $\mathcal{X}_{\text{free}}$, initial state x_0, final state x_g, open set $\mathcal{O}$, closed set $\mathcal{C}$
  $\mathcal{C} \leftarrow \{x_0\}$
  $\mathcal{O} \leftarrow$ EXPAND$(x_s, \mathcal{P})$
  while $\mathcal{O} \neq \emptyset$ **do**
    $(x, g(x)) \leftarrow$ POP$(\mathcal{O})$
    if $x == x_g$ **then** ▷ Goal found
      return $g(x)$
    end if
    for all $(\tilde{x}, \tilde{c}) \in$ EXPAND$(x, \mathcal{P})$ **do**
      if $\tilde{x} \in \mathcal{X}_{\text{free}}$ **and** $\tilde{x} \notin \mathcal{C}$ **then**
        $c_{\text{tot}} = g(x) + \tilde{c} + \tilde{h}(\tilde{x}, x_g)$ ▷ Estimate total cost
        $c \leftarrow$ CURRENT_COST$(\tilde{x}, \mathcal{O})$
        if $\tilde{x} \notin \mathcal{O}$ **or** $c > c_{\text{tot}}$ **then**
          $\mathcal{O} \leftarrow \mathcal{O} \bigcup \{(\tilde{x}, c_{\text{tot}})\}$ ▷ Update total cost estimate
        end if
      end if
    end for
    $\mathcal{C} \leftarrow \mathcal{C} \bigcup \{x\}$
  end while

3.3.2 Hybrid A-star

A disadvantage of the original A^* formulation is that it only allows states to take on the discretized values $x_s \in \mathcal{X}_s$. This is extended in [14] to the Hybrid A^* formulation, which allows continuous states. To discretize the state-space it is divided into cells, and states in the same cell are considered equal. The difference from the classic A^* formulation is illustrated in Figure 3.1.

The Hybrid A^* algorithm also includes the concept of *analytic expansions*, *i.e.* the model of the system is simulated from the current state x to the goal x_g at each expansion step. If this simulated path is feasible, *i.e.* doesn't collide with obstacles, a solution is considered found. This was shown to decrease execution time of the algorithm, and it also allows the goal state to be reached exactly instead of reaching the closest discrete state.

A disadvantage of the sampling method used in Hybrid A^* is that removing all states but one in each cell also removes all theoretical optimality guarantees of the solution. The algorithm is therefore rather motivated by its practical usability, and extensive use of the algorithm shows that the solution often lies in a close



(a) *Regular A**

(b) *Hybrid A**

Figure 3.1: *Difference between regular and hybrid A**

neighborhood of the optimal solution. Both in the original and later works this issue is handled by improving the initial solution using numerical optimization methods [14] [46].

3.3.3 Non-holonomic heuristics

A common choice of heuristic function when the goal is to find minimum-length paths is the euclidean distance $\tilde{h}(x, x_g) = \|x_g - x\|$. This is guaranteed to be an admissible heuristic if a single fixed coordinate frame is used. However, in many cases for non-holonomic systems this measure greatly underestimates the actual cost-to-go, which leads to unnecessary node expansions and increased computation time of the algorithm. It is therefore desirable to use another heuristic which takes the non-holonomic properties of the system into account [37].

Dubin's metric

Dubin's path was introduced in [15] and provides an analytical solution for the shortest path between two states (x_0, y_0, ψ_0) and (x_g, y_g, ψ_g) with a constraint on maximal rate of turn $|\dot{\psi}| \leq \dot{\psi}_{\max}$. The length of a Dubin's path has been widely used as a heuristic for non-holonomic systems only capable of forward motion, such as car-like robots and fixed-wing UAVs [20].

Heuristic Look-Up Table

Another efficient method for non-holonomic systems is to pre-compute the optimal cost from a number of start states to a number of goal states and store these in a Heuristic Look-Up Table (HLUT) [22]. However, since the HLUT must be finite in size, a fallback heuristic such as euclidean distance is used if the value of

$h(x, \tilde{x})$ is not available for some x and $\tilde{x}$. This results in a trade-off between HLUT size and algorithm efficiency, and states where the difference between $h(x, \tilde{x})$ and the fallback heuristic is large should be prioritized.

Given a set of motion primitives $\mathcal{P}$ the HLUT can be efficiently generated using Dijkstra's shortest path algorithm. An outline of this method is given in Algorithm 2, where $\mathcal{X}$ is the set of states for which HLUT values are desired, and the function definitions are equal to the ones in Algorithm 1.

Algorithm 2 HLUT generation using Dijkstra's algorithm

Require: Motion primitive set $\mathcal{P}$, desired states $\mathcal{X}$, initial state x_0, open set $\mathcal{O}$, closed set $\mathcal{C}$

```
C ← {x_0}
O ← EXPAND(x_0, P)
while O ≠ ∅ do
    (x, g(x)) ← POP(O)
    for all (x̃, c̃) ∈ EXPAND(x, P) do
        if x̃ ∈ X and x̃ ∉ C then
            c_tot = g(x) + c̃                          ▷ Calculate cost-to-go
            c ← CURRENT_COST(x̃, O)
            if x̃ ∉ O or c > c_tot then
                O ← O ⋃ {(x̃, c_tot)}                  ▷ Update cost-to-go
            end if
        end if
    end for
    C ← C ⋃ {x}
    HLUT(x_0, x) = g(x)                               ▷ Store value in HLUT
end while
```

3.3.4 Inflated heuristics and sub-optimality guarantees

If $\tilde{h}(x, \tilde{x})$ is an admissible heuristic and the heuristic $\epsilon\tilde{h}(x, \tilde{x})$ with some inflation factor $\epsilon > 1$ is used during planning, the resulting path is not guaranteed to be optimal. However, an important result is that the sub-optimal path is guaranteed to be at most ϵ times longer than the optimal one. This property is exploited in so-called anytime algorithms, since inflating the heuristic value often leads to much faster solutions which is desirable in realtime implementations [28].

4

Motion planning using waypoint optimization

4.1 Waypoint sampling

The desired output of the motion planner in this thesis is a waypoint sequence $\mathcal{M}$, as defined in section 2.5.3, which takes the UAV from a desired initial state $(x_{N,0}, y_{E,0}, \psi_0)$ to a goal state $(x_{N,g}, y_{E,g}, \psi_g)$. Moreover, physical constraints of the UAV and wind should be taken into account. This formulation is well aligned with *input sampling* methods such as Hybrid A^* [14]. In such methods, the reachability graph is created by forward simulation of the transition function $f(x, u)$ using input values u sampled from a set $\mathcal{U}_s$.

4.1.1 State and input set definition

Based on the kinematic model in Equation (2.18) the state vector is defined as

$$x = (x_N, y_E, \psi) \tag{4.1}$$

The input is defined as

$$u = \boldsymbol{p}_{i+1} - \boldsymbol{p}_i \equiv (\Delta x_N, \Delta y_E) \tag{4.2}$$

i.e. the coordinates of the next waypoint relative to the current, specified in the inertial frame.

4.1.2 State transition function

The definition of u combined with the trajectory following controller derived in Section 2.5.2 leads to a model of the closed-loop system. Using such a model during forward simulation implies that it will be possible for the controller to follow the resulting reference trajectory.

The desired COG for the current line-segment is

$$\psi_u = \text{atan2}(\Delta y_E, \Delta x_N) \tag{4.3}$$

If roll dynamics are neglected, the commanded rate of turn is obtained by inserting the roll command from Equation (2.35) into Equation (2.18), which gives

$$\dot{\psi}_{\text{cmd}} = \frac{a_{\text{cmd}}}{V_a} \tag{4.4}$$

where a_{cmd} is given by Equation (2.34) with η as defined in (2.31)–(2.33). However, for a real UAV the magnitude of the rate of turn is limited by some $\dot{\psi}_{\max}$. The actual value of $\dot{\psi}$ is thus

$$\dot{\psi} = \begin{cases} \dot{\psi}_{\text{cmd}} & |\dot{\psi}_{\text{cmd}}| \leq \dot{\psi}_{\max} \\ \text{sgn}(\dot{\psi}_{\text{cmd}})\dot{\psi}_{\max} & \text{otherwise} \end{cases} \tag{4.5}$$

Finally, the kinematic model of the closed-loop system becomes

$$\dot{x} = f(x, u) = \begin{bmatrix} V_N \\ V_E \\ \dot{\psi} \end{bmatrix} \tag{4.6}$$

4.2 Input set generation

An input set $\mathcal{U}_s$ is a subset of a motion primitive set $\mathcal{P}$ introduced in Section 3.2.2 since it contains only of pre-computed inputs but no state trajectories. Therefore, the heading-change method introduced in [9] to generate $\mathcal{P}$ can also be applied when generating $\mathcal{U}_s$. The resulting inputs will consist of waypoints that result in a desired change of direction while taking UAV kinematics, wind and tracking performance of the controller into account.

4.2.1 Optimal control formulation

The input set is generated by solving the optimal control problem

$$\min_{x(t),u,T} \quad J = \Phi(x(T), u) + \int_0^T V_a \, dt \tag{4.7a}$$

$$\text{subject to} \quad \psi(0) = \psi_{\text{wca}} \tag{4.7b}$$

$$|\psi_{\text{cog}}(x(T)) - \Delta\psi_{\text{cog}}| \leq \Delta\psi_{\min} \tag{4.7c}$$

$$\dot{x} = f(x(t), u) \tag{4.7d}$$

$$x(t) \in \mathcal{X} \tag{4.7e}$$

$$u \in \mathcal{U} \tag{4.7f}$$

for different values of $\boldsymbol{w}$ and direction change $\Delta\psi_{\text{cog}}$. To increase the feasible region, the constraint on $\psi_{\text{cog}}(x(T))$ is relaxed to allow values in a region around

the desired $\Delta\psi_{\text{cog}}$ instead of a strict equality constraint. The closed-loop kinematic model (4.6) depends on wind, which has to be taken into account when generating $\mathcal{U}_s$. This dependency as well as other relevant properties of (4.7) are discussed in the sections below.

Discretization of the wind direction

During motion planning in wind, the heading relative to the wind $\psi_r = \psi - \psi_w$ might take on any value. In practice, this implies that inputs must be generated for a set of discrete wind directions $\{\psi_{r,0}, \ldots, \psi_{r,n}\}$ which cover 360 degrees. Given a relative heading

$$\psi_r : \psi_{r,i} < \psi_r < \psi_{r,i+1} \tag{4.8}$$

the input $u \in \mathcal{U}_s$ selected by the planning algorithm was generated for $\psi_{r,i}$ or $\psi_{r,i+1}$. The discretization interval $|\psi_{w,i+1} - \psi_{w,i}|$ has to be sufficiently small in order to secure good tracking performance of the closed-loop system.

Planning with COG instead of heading

As shown in Section 2.4.2, the heading required to follow a line-segment is dependent on the current wind $\boldsymbol{w}$. Therefore constraints related to direction change in Equation (4.7) are formulated in terms of ψ_{cog} as defined in Equation (2.26). A direct consequence is that the initial value of ψ when generating inputs should be set to ψ_{wca} defined in Equation (2.25), as this corresponds to an initial ψ_{cog} of $0°$.

Cross-track error penalty

The cross-track error at the end of a line-segment can be calculated as

$$d(x(T), u) = x_N(T) \sin\psi_u - y_E(T) \cos\psi_u \tag{4.9}$$

If $d(x(T), u) \neq 0$ the initial cross-track error for the next line-segment will be non-zero. Since the closed-loop system is used when expanding the graph, a small initial error can be mitigated, but large errors should be discouraged as they result in unpredictable behaviour of the controller. The cross-track error penalty is defined as

$$\Phi(x(T), u) = \lambda_d \max(|d(x(T), u)| - d_{\min}, 0) \tag{4.10}$$

and is included in the optimization objective (4.7). This term is, by construction, zero when the final cross-track error is below some acceptable threshold $d_{\min}$. In this case, only the trajectory length is penalized. The penalty for larger cross-track errors is tuned by the scaling factor $\lambda_d > 0$.

4.2.2 Solving the optimal control problem

Methods commonly used to solve optimal control problems include *multiple shooting* and *direct collocation* [13]. However, the following properties of (4.7) makes it hard to solve with such methods:

1. The closed-loop system is highly non-linear, especially when including the saturation from Equation (4.5).

2. In optimal control problems the input $u(t)$ can normally be chosen freely from $\mathcal{U}$ for each time-step, while in this formulation the input is forced to be a constant $u(t) = u$, $0 < t < T$.

The second property implies that when transformed to a Nonlinear Program using *e.g.* multiple shooting, the optimization variables corresponding to $x(t)$ in each time-step all depend on the same constant u. In this sense the resulting formulation is more closely related to a *direct shooting* problem, which are known to be less linear and thus harder to solve [13].

Derivative-free Optimization

Since all properties of the solution of Equation (4.7) are dependent on the choice of the input u, different solutions can be studied by simulating the closed-loop system for different choices of u. Choices of u which lead to solutions that violate the constraints can easily be pruned. A number of solutions with different characteristics for a desired course change $\Delta\psi_{\text{cog}} = 90°$ are illustrated in Figure 4.1-4.4. The brighter color defines a higher objective value, and the green and red arrows indicate the initial and final positions of the UAV. The wind was defined as $\psi_w = 0°$ and $W = 5$ m/s. As can be seen the feasible region is non-convex but there is a clear global optimum.

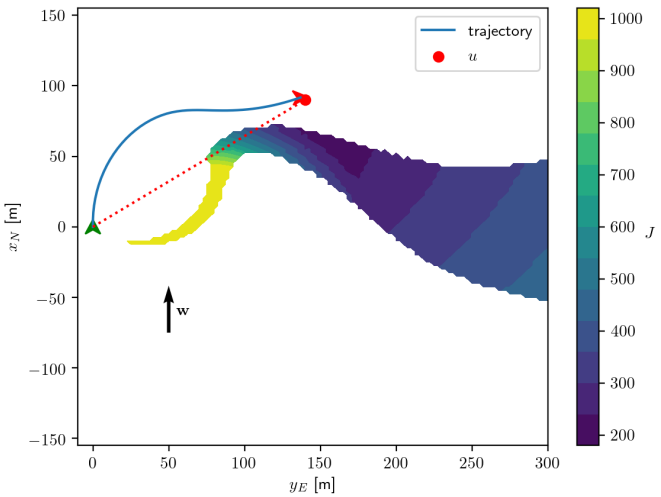


Figure 4.1: *$u = (90, 140)$: Infeasible solution due to incorrect final ψ_{cog}.*

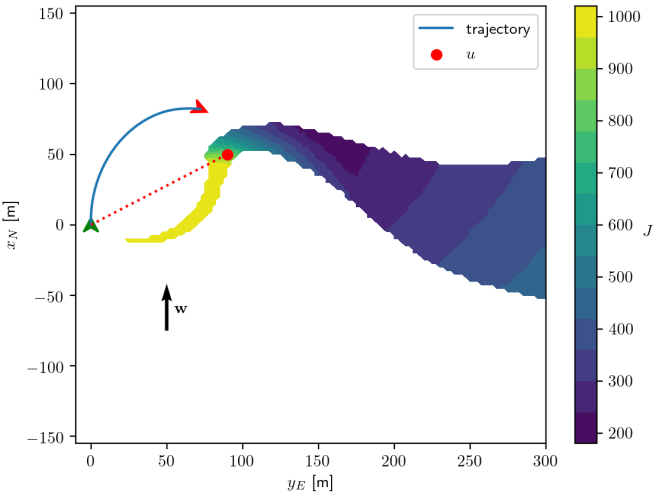


Figure 4.2: *$u = (50, 90)$: Sub-optimal solution due to large final cross-track error, $J = 891$.*

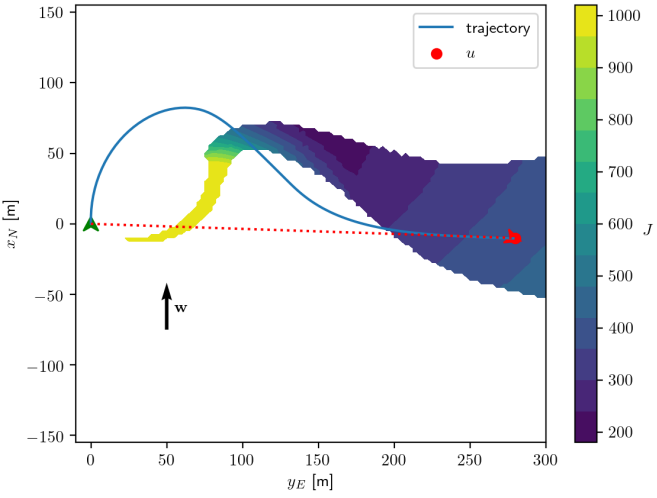


Figure 4.3: *$u = (-10, 280)$: Sub-optimal solution due to unnecessarily long trajectory, $J = 378$.*

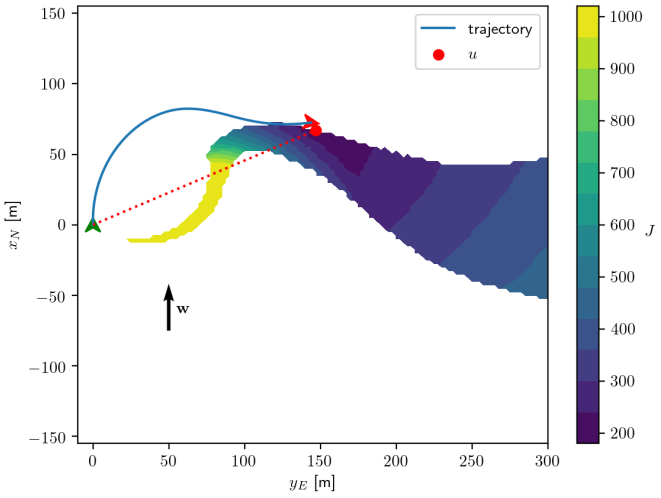


Figure 4.4: *$u = (67, 147)$: Optimal solution, $J = 186$.*

Since there are only two free parameters, the north and east coordinates of u, an approximate optimum could be found by performing a grid search over different values of these parameters. However, this solution would depend on the discretization interval of the grid and searching over a grid with sufficiently fine resolution is computationally expensive. A more efficient method is to use *derivative-free* optimization methods, as presented in [38]. In those methods the optimization problem is formulated as

$$\min_{\xi \in \mathbb{R}^n} \quad F : \xi \to \mathbb{R} \tag{4.11a}$$

$$\text{subject to} \quad \xi \in \Omega \subseteq \mathbb{R}^n \tag{4.11b}$$

$$\tag{4.11c}$$

where no other information, such as the derivatives of F, is available. One class of derivative-free methods called Mesh Adaptive Direct Search (MADS) was introduced in [8]. This method is based on creating an increasingly fine grid around the currently optimal solution on which the objective function is evaluated. In [8] this method is shown to successfully converge to the global optimum of various non-convex optimization problems using the derivative-free optimization formulation.

4.2.3 Robustness during wind variations

The requirement to generate a set of inputs for each possible wind speed limits the practical applicability of the method. A more useful approach is to generate input sets which handle wind speeds $W \in [W_{\min}, W_{\max}]$. This problem can be

formulated as finding an input u which is feasible for both $W_{\min} = (1 - \delta_W)\tilde{W}$ and $W_{\max} = (1 + \delta_W)\tilde{W}$ for some $\delta_W < 1$ and $\tilde{W} = (W_{\max} - W_{\min})/2$. To find a solution which is feasible in the extreme cases $W = W_{\min}$ and $W = W_{\max}$, the derivative-free optimization problem was formulated as

$$\min_{x,u} \quad F(x,u) = \max(J_{\text{low}}(x,u), J_{\text{high}}(x,u)) \tag{4.12a}$$

$$\text{subject to} \quad (x,u) \in \Omega \tag{4.12b}$$

$$\tag{4.12c}$$

where J_{low} is the objective value of (4.7) for $W = W_{\min}$ and J_{high} is the objective value for $W = W_{\max}$. The feasible set Ω is defined as the values of x and u where the constraints in (4.7) hold for all $W \in [W_{\min}, W_{\max}]$.

4.3 Improvement step

As mentioned in Section 3.3.2 the initial solution from Hybrid A^* is often improved using numerical optimization. However, due to the limitations presented in Section 4.2.2 such methods are not available. Therefore, a simpler and practically motivated approach was used.

The initial solution computed by the Hybrid A^* search is henceforth denoted

$$\mathcal{M}_{\text{init}} = \{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n\} \tag{4.13}$$

which is an ordered sequence of n waypoints $\boldsymbol{p}_i$. A sub-sequence of a mission is denoted

$$\mathcal{M}_{k:l} = \{\boldsymbol{p}_k, \ldots, \boldsymbol{p}_l\}, \quad 0 \leq k < l \leq n \tag{4.14}$$

A *reduced set* of waypoints is defined as

$$\mathcal{M}_{k,l} = \{\boldsymbol{p}_k, \boldsymbol{p}_l\} \tag{4.15}$$

i.e. the first and last waypoint of a sub-sequence $\mathcal{M}_{k:l}$. By simulating the closed-loop system using $\mathcal{M}_{\text{init}}$ the inital COG and cross-track error $(\psi_{\text{cog}}, d)_i$ at each waypoint can be found. Since (4.6) minimizes the cross-track error in each timestep, the following relation always holds:

$$L(\mathcal{M}_{l:k}) \geq L(\mathcal{M}_{l,k}) \tag{4.16}$$

where $L(\cdot)$ denotes the length of the trajectory produced by simulating (4.6) with a given waypoint sequence. If the same COG and cross-track error is achieved and no obstacles are collided with while using $\mathcal{M}_{k,l}$ the intermediate waypoints of $\mathcal{M}_{k:l}$ can be eliminated. This method is outlined in Algorithm (3) where the function SIMULATE$(\mathcal{M}, \mathcal{X}_{\text{obst}})$ returns the COG and cross-track error achieved by simulating $\mathcal{M}$ and if there were any collissions with $\mathcal{X}_{\text{obst}}$. The result of applying the improvement step to a Hybrid A^* solution is illustrated in Figure 4.5.

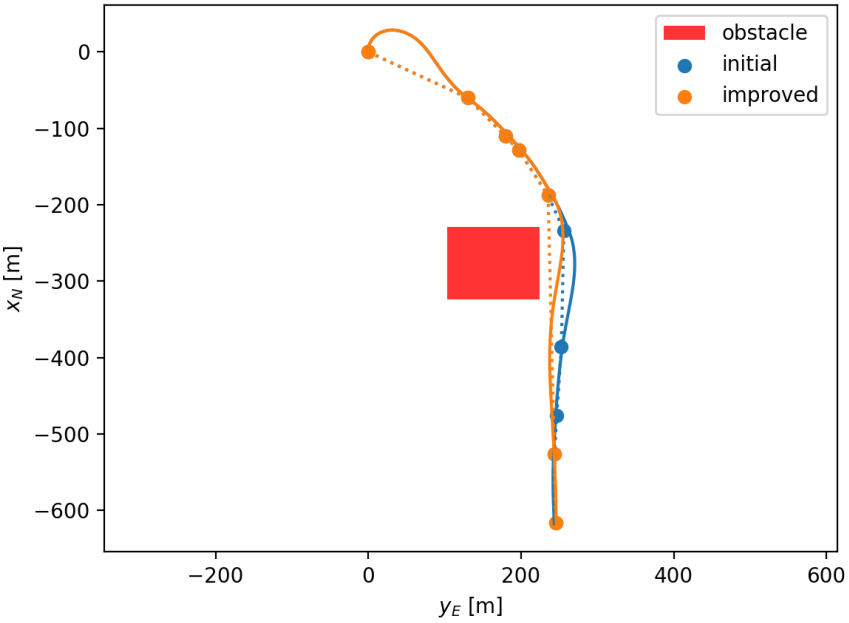


Figure 4.5: *Trajectory length reduction by eliminating waypoints.* $x_0 = (0, 0, 0°)$, $x_g = (-615, 245, 180°)$. *By eliminating the intermediate waypoints in the submission* $\mathcal{M}_{4:8}$ *the same goal state is reached but the trajectory length is reduced.*

Algorithm 3 Solution improvement by waypoint elimination

Require: Initial mission $\mathcal{M}_{\text{init}}$ and corresponding COG and cross-track errors $\{(\psi_{\text{cog}}, d)_i\}$
  $\mathcal{M}_{\text{imp}} \leftarrow \{\boldsymbol{p}_0\}$
  $i \leftarrow 0$
  while $i \leq n$ **do**
    $j \leftarrow i + 1$
    $(\boldsymbol{p}_{\text{best}}, i_{\text{best}}) \leftarrow (\boldsymbol{p}_j, j)$
    while $j \leq n$ **do**
      $\psi_{\text{cog}}, d, \text{has\_collided} \leftarrow \text{SIMULATE}(\mathcal{M}_{i,j}, \mathcal{X}_{\text{obst}})$
      if not has_collided **and** $|\psi_{\text{cog}} - \psi_{\text{cog},j}| \leq \Delta\psi_{\min}$ **and** $|d - d_j| \leq d_{\min}$ **then**
        if j==n **then**
          $\mathcal{M}_{\text{imp}} \leftarrow \mathcal{M}_{\text{imp}} \bigcup \{\boldsymbol{p}_j\}$
          return $\mathcal{M}_{\text{imp}}$
        end if
        $(\boldsymbol{p}_{\text{best}}, i_{\text{best}}) \leftarrow (\boldsymbol{p}_j, j)$
      end if
    end while
    $\mathcal{M}_{\text{imp}} \leftarrow \mathcal{M}_{\text{imp}} \bigcup \{\boldsymbol{p}_j\}$
    $i \leftarrow i_{\text{best}}$
  end while

4.4 Heuristic function

As discussed in Section 3.3.1, the choice of heuristic function is crucial in achieving good performance of the planner. The goal of the heuristic function is to estimate the length of the shortest path relative to the air from an initial state x_0 to a final state x_g.

4.4.1 Cost estimation for straight line-segments

Assuming that the heading ψ has converged to ψ_{wca}, the speed of the UAV along a straight line-segment in the inertial frame is given by

$$V_{\parallel} = \cos\psi_s(V_a \cos\psi_{\text{wca}} + W\cos\psi_w) + \sin\psi_s(V_a \sin\psi_{\text{wca}} + W\sin\psi_w) \tag{4.17}$$

where ψ_s is the direction defined by the line. This means that the time it takes for the UAV to travel along the line is equal to

$$t = \frac{\|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|}{V_{\parallel}} \tag{4.18}$$

where $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ are the start and end waypoints of the line. Thus, the distance travelled relative to the air is equal to

$$s_a = V_a t = \frac{V_a}{V_{\parallel}}\|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\| \tag{4.19}$$

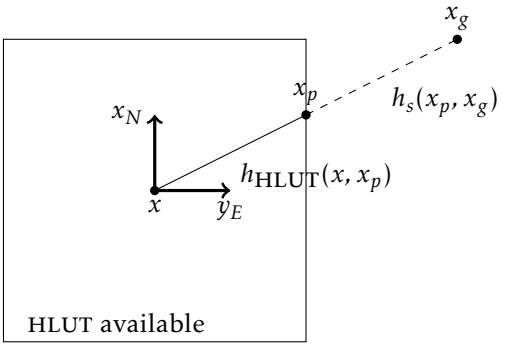


Figure 4.6: *Projection of queries on HLUT*

and s_a provides a good heuristic estimate for traveling along a straight line-segment in wind assuming that $\psi_0 = \psi_g = \psi_{\text{wca}}$. This also implies that the Euclidean distance $\|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|$ is not an admissible heuristic if $V_a/V_{\|} < 1$.

4.4.2 Cost estimation for arbitrary initial and final heading

Estimating the cost for traveling between states with arbitrary ψ_0 and ψ_g is a more challenging problem than straight line-segments. Methods to calculate such time-optimal paths in the presence of wind are given in both [29] and [42], but since there is no analytical solution in all cases these methods rely on numerical root-finding techniques. Computing these values every time an heuristic estimate is needed was deemed infeasible due to the high computational cost.

When the heuristic cannot be calculated in real-time, an option is to use a HLUT as discussed in Section 3.3.3. By using the generated inputs $\mathcal{U}_s$ when calculating costs stored in the HLUT, these directly correspond to the true cost-to-go. However, a drawback of using a HLUT is that the wind speed W affects the cost, and thus different values of W require different HLUTs.

To estimate the cost of queries not stored in the HLUT, these queries can be projected as shown in Figure 4.6. The total heuristic value can then be estimated as

$$\tilde{h}(x, x_g) = h_{\text{HLUT}}(x, x_p) + h_s(x_p, x_g) \tag{4.20}$$

where $h_s(x, \tilde{x})$ is the estimated cost for a straight line-segment.

4.4.3 Wind variation effects on the heuristic

If the actual wind speed $\tilde{W}$ is different from the wind speed W used during planning, this might affect the admissibility of the heuristic. To study this effect, consider traveling along a straight path segment of length $\Delta s = \|x - \tilde{x}\|$ under the assumptions in Section 4.4.1. An admissible heuristic is then

$$\tilde{h}(x, \tilde{x}) = \frac{V_a}{V} \Delta s \tag{4.21}$$

where V is the velocity in the inertial frame. Wind has the largest effect on V when traveling in direct tailwind or headwind, and in those cases $V = V_a \pm \tilde{W}$. The heuristic function $h(x, \tilde{x})$ used during planning is the same but with $V = V_a \pm W$. The ratio between the admissible and actual heuristics becomes

$$\epsilon = \frac{h}{\tilde{h}} = \frac{V_a \pm \tilde{W}}{V_a \pm W} \tag{4.22}$$

where the signs in the numerator and denominator are always equal. As mentioned in Section 3.3.4 the heuristic is not admissible if $\epsilon > 1$ which is the case if $\tilde{W} > W$ when traveling in tailwind, or $\tilde{W} < W$ when traveling in headwind. In these cases, using this heuristic estimate is analouge to using an inflated heuristic with inflation factor ϵ. Moreover, the effects of using an incorrect wind estimate will be more significant if the magnitude of $\tilde{W}$ is close to that of V_a.

5

Robust landing sequences

5.1 Problem formulation

The problem of landing a fixed-wing UAV on a runway was studied in many previous works, *e.g.* [44] and [36]. However, small and light-weight UAVs such as the ones studied in this thesis can land in any area as long as the ground is flat enough. The main issue is instead that there might be obstacles such as trees around the landing area which limit the possible approach directions. Wind also plays an essential role, since landing in headwind enables much shorter approach paths relative to the ground.

The problem of landing is thus defined as finding the inputs which lands the UAV as close to the center as possible in a pre-defined landing area $\mathcal{A}$. The landing area is defined as a rectangular region with walls of height h_{safe}, and to ensure safe landing the UAV must enter $\mathcal{A}$ above this altitude. There might also be obstacle regions $\mathcal{X}_{\text{obst}}$ around the landing area where the UAV is not permitted to fly. The problem definition is illustrated in Figure 5.1.

5.2 Landing sequence

A landing sequence for fixed-wing UAVs is defined by an approach point $\boldsymbol{p}_a$ and landing point $\boldsymbol{p}_l$. These points define an approach direction ψ_l. The landing velocity V_l depends on ψ_l, the airspeed V_a and current wind as

$$V_l = \cos\psi_l(V_a\cos\psi_{\text{wca}} + W\cos\psi_w) + \sin\psi_l(V_a\sin\psi_{\text{wca}} + W\sin\psi_w) \tag{5.1}$$

The landing sequence is divided into an approach phase and a flare phase, which are illustrated in Figure 5.2.

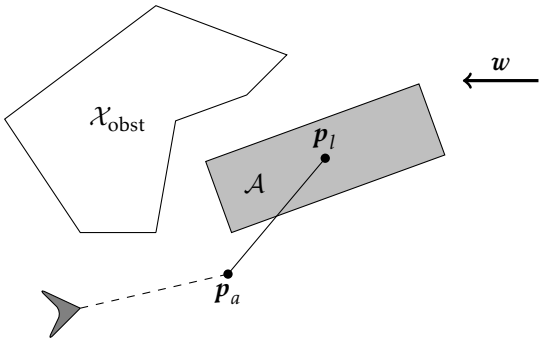


Figure 5.1: *Landing sequence problem definition. The goal is to find the approach point $\boldsymbol{p}_a$ and landing point $\boldsymbol{p}_l$ which lands the UAV in the specified landing area $\mathcal{A}$. The UAV is not permitted to fly above obstacle regions in $\mathcal{X}_{obst}$.*

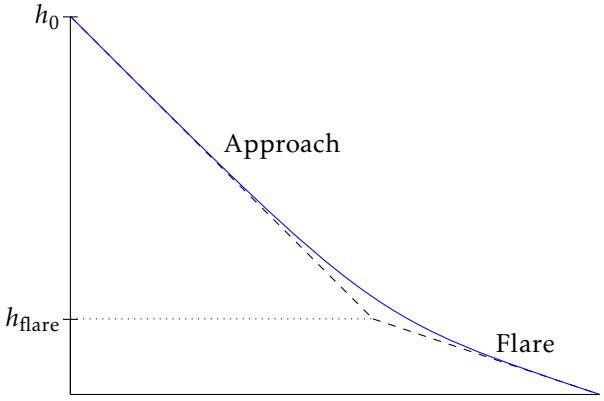


Figure 5.2: *Altitude profile of a fixed-wing landing sequence*

During the approach phase, the autopilot commands an approach sink-rate

$$\dot{h}_{\text{cmd}} = \frac{h_0 - h_{\text{flare}}}{\|\boldsymbol{p}_a - \boldsymbol{p}_l\| - R_{\text{flare}}} V_l \tag{5.2}$$

where h_0 is the initial altitude, h_{flare} is the flare altitude and R_{flare} is the flare distance. To ensure a sufficiently low touchdown speed, the flare phase is activated once the UAV reaches the altitude h_{flare} above the ground. In this mode it instead tries to achieve a pre-defined constant flare sink-rate

$$\dot{h}_{\text{cmd}} = \dot{h}_{\text{flare}} \tag{5.3}$$

which means that the flare distance is given by

$$R_{\text{flare}} = h_{\text{flare}} \frac{V_l}{\dot{h}_{\text{flare}}} \tag{5.4}$$

Due to physical limitations in the system, the landing sequence has to be defined such that

$$\dot{h}_{\text{cmd}} \leq \dot{h}_{\text{max}} \tag{5.5}$$

for some constant $\dot{h}_{\text{max}}$ during the approach.

5.3 Calculating a landing sequence

The goal of the landing sequence generation is to ensure safe landing in the designated area $\mathcal{A}$. There are two important measures to discuss regarding the safety of a landing sequence, the altitude of the UAV when entering $\mathcal{A}$ and the distance from the landing point to the center of $\mathcal{A}$. If the entry altitude is too low, there is a risk of colliding with surrounding obstacles and if the distance to the center is too large, the UAV will not land in $\mathcal{A}$. Thus, by minimizing these two errors the safety of the landing sequence is maximized. This also maximizes the robustness to other errors such as variations in wind speed.

Since a partial goal of the landing sequence is to land as closely as possible to the center point $\boldsymbol{p}_c$ of $\mathcal{A}$, any landing sequence is defined by placing $\boldsymbol{p}_a$ and $\boldsymbol{p}_l$ along a line which passes through $\boldsymbol{p}_c$ and points in the direction given by ψ_l. This fact can be used to divide the problem in two parts, where first the best ψ_l is determined and then $\boldsymbol{p}_a$ and $\boldsymbol{p}_l$ based on the chosen direction.

5.3.1 Determining the approach direction

Any line through $\boldsymbol{p}_c$ with a given direction will cross the walls of $\mathcal{A}$ in exactly two points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, as is illustrated in Figure 5.3. Thus, the following constraints must be considered:

- The distance $\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|$ has to be large enough such that the altitude h in $\boldsymbol{p}_1$ is larger than h_{safe} while allowing the constraint (5.5) to be satisfied
- The approach direction ψ_l has to be chosen such that the initial trajectory up until $\boldsymbol{p}_a$ is not inside $\mathcal{X}_{\text{obst}}$

To find the minimum feasible distance, the altitude of the UAV when entering $\mathcal{A}$ is denoted h_e. Assuming that $h_e = h_{\text{safe}}$, the minimum feasible distance to the flare point is given by

$$R_{\text{min}} = (h_{\text{safe}} - h_{\text{flare}}) \frac{V_l}{\dot{h}_{\text{max}}} \tag{5.6}$$

To ensure landing in $\mathcal{A}$ it is thus required that

$$\|\boldsymbol{p}_1 - \boldsymbol{p}_2\| \geq R_{\text{min}} + R_{\text{flare}} \tag{5.7}$$

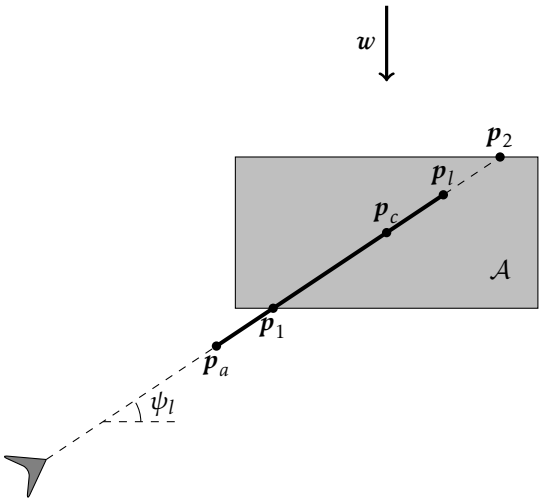


Figure 5.3: *Variables determining a landing sequence. Given an approach direction ψ_l, a line going through the center of the landing area $\boldsymbol{p}_c$ will cross the edges of $\mathcal{A}$ in exactly two points, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$.*

where R_{flare} is given by Equation (5.4). To ensure the second constraint, a simple approach is to create lines starting in $\boldsymbol{p}_c$ with length $K(R_{\min} + R_{\text{flare}})$ and direction $\psi_l + 180°$ for some $K \geq 0.5$ and different discrete values of ψ_l. The set of feasible approach directions $\{\psi_l\}_{\text{feas}}$ can then be found by checking each corresponding line for intersections with $\mathcal{X}_{\text{obst}}$. Finally, the approach direction is chosen as

$$\psi_l^* = \underset{\psi \in \{\psi_l\}_{\text{feas}}}{\arg\min}\, R(\psi) \tag{5.8}$$

where

$$R(\psi) = R_{\min}(\psi) + R_{\text{flare}}(\psi) \tag{5.9}$$

5.3.2 Determining the approach points

After fixing the approach direction to $\psi_l = \psi_l^*$ the next step is to calculate the values of $\boldsymbol{p}_a$ and $\boldsymbol{p}_l$. Since the approach direction is fixed, the remaining variables can be redefined as

$$R_a = (\boldsymbol{p}_a - \boldsymbol{p}_2) \cdot \hat{l} \tag{5.10a}$$

$$R_l = (\boldsymbol{p}_l - \boldsymbol{p}_2) \cdot \hat{l} \tag{5.10b}$$

where $\hat{l}$ is a unit vector pointing in the direction $\psi_l + 180°$. This definition ensures landing in $\mathcal{A}$ as long as $0 \leq R_l \leq 2R_c$, where

$$R_c = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|/2 \tag{5.11}$$

The problem is thus finding R_a and R_l so that $|h_e - h_{\text{safe}}|$ is maximized and $|R_c - R_l|$ is minimized, while fulfilling the given constraints. From Equation (5.2), the commanded sink-rate is then

$$\dot{h}_{\text{cmd}} = \frac{h_0 - h_{\text{flare}}}{R_a - R_l - R_{\text{flare}}} V_l \tag{5.12}$$

and the altitude during the approach is given by

$$h(R) = h_0 - R\frac{\dot{h}_{\text{cmd}}}{V_l} = h_0 - R\frac{h_0 - h_{\text{flare}}}{R_a - R_l - R_{\text{flare}}} \tag{5.13}$$

where h_0 is the initial altitude. To ensure enough altitude when entering $\mathcal{A}$, it is required that

$$h_e = h(R_a - 2R_c) \geq h_{\text{safe}} \tag{5.14}$$

The landing parameters can thus be calculated by solving the optimization problem

$$\min_{R_a, R_l} \quad J = |R_c - R_l|^2 - |h_e - h_{\text{safe}}|^2 \tag{5.15a}$$

$$\text{subject to} \quad 0 \leq R_l \leq 2R_c \tag{5.15b}$$

$$\frac{h_0 - h_{\text{flare}}}{R_a - R_l - R_{\text{flare}}} V_l \leq \dot{h}_{\max} \tag{5.15c}$$

$$h_0 - \frac{R_a - 2R_c}{R_a - R_l - R_{\text{flare}}}(h_0 - h_{\text{flare}}) \geq h_{\text{safe}} \tag{5.15d}$$

This is a non-linear optimization problem with linear constraints. In this work, it was solved using the IPOPT solver [45], which is implemented in the CASADI toolkit, a general toolkit for solving nonlinear optimization problems numerically [7].

6

Implementation and experiments

6.1 Implementation details

The following sections describe relevant details regarding the implementation of the proposed method.

6.1.1 Obstacle avoidance

To ensure low execution times it is crucial to use an efficient method of checking for collisions between states and $\mathcal{X}_{\text{obst}}$. In this implementation, the S2Geometry library developed by Google was used [6]. This is a C++ library which contains efficient methods to index geometrical objects of any shape, and checking for collisions between different geometries such as points, lines and polygons.

6.1.2 Input set generation

The input set $\mathcal{U}_s$ was generated using the approach described in Section 4.2. It was generated for wind directions $\psi_{w,s} = \{0°, 20°, 40°, \dots, 340°\}$ and desired final course changes $\Delta\psi_{\text{cog}} = \{20°, 40°, \dots, 180°\}$, resulting in a total of 162 inputs for each specific W. Symmetries of the system reduce the set of necessary inputs as solutions for $\Delta\psi_{\text{cog}} = \{-20°, -40°, \dots, -180°\}$ are simply found by mirroring the y_E coordinate of u. The optimization problem was solved using NOMAD [26], a C++ implementation of the MADS algorithm introduced in Section 4.2.2. Cross-track error constraints were defined by $\lambda_d = 25$ and $d_{\min} = 2.5$ m, COG error $\Delta\psi = 15°$ and wind variation $\delta_W = 0.25$. The initial guess for u was found by performing a grid search over the values

$$\mathcal{U}_{s,\text{init}} = \{(\Delta x_N, \Delta y_E) : |\Delta x_N| \leq 300, 0 \leq \Delta y_N \leq 300\} \tag{6.1}$$

with a step size of 10 meters and selecting the input with the lowest value of the objective in Equation (4.12). Simulations of the closed-loop system for the generated inputs u, calculated for some different wind directions and $W \in [3.75, 6.25]$ m/s, are shown in Figure 6.1.

6.1.3 State-space discretization

To apply graph-search methods, the state-space has to be discretized. In this work the values of x_N and y_E were discretized into cells of size $d = 10$ meters, and the heading ψ was discretized in steps of 20°. The Hybrid A^* method presented in Section 3.3.2 was used when sampling the state space, allowing continuous values of the state vector x but assigning those to the closest discretized state.

6.1.4 State expansions

The step EXPAND in Algorithm 1 presented in Section 3.3.1 has to take both the wind direction ψ_w and the heading ψ of x into account. Since the inputs in $\mathcal{U}_s$ are generated using initial course $\psi_{\text{cog}} = 0$, it is first necessary to calculate the closest relative wind direction

$$\psi_{w,\text{rel}} = \underset{\psi_{w,s} \in \{\psi_{w,s}\}}{\arg\min} \, |(\psi - \psi_w) - \psi_{w,s}| \tag{6.2}$$

which is used to select the inputs for expansion. When mirroring inputs the wind direction also has to be mirrored, *i.e.* $\tilde{\psi}_w = 360° - \psi_w$ is used to calculate $\psi_{w,\text{rel}}$. The selected inputs also have to be rotated, *i.e.* the initial reference $u = (\Delta x_N, \Delta y_E)$ is transformed to

$$\tilde{u} = (\cos\psi\Delta x_N + \sin\psi\Delta y_E, -\sin\psi\Delta x_N + \cos\psi\Delta y_E) \tag{6.3}$$

Finally, the expanded states and corresponding costs are found by simulating the closed-loop system (4.6) using each selected $\tilde{u}$ as input. The actual wind direction ψ_w is used instead of $\psi_{w,s}$ in these simulations.

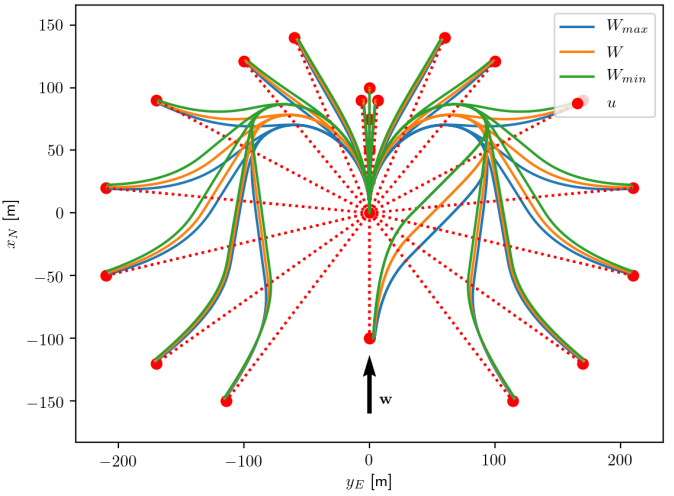
Handling perpendicular winds

A drawback of using straight line-segments as the control reference is that some inputs become problematic when the difference between ψ and ψ_w is close to 90°. In this situation, expanding using an input which corresponds to a course change of $\Delta\psi_{\text{cog}} \approx 180°$ might result in the trajectory controller choosing to fly in tailwind instead of headwind, leading to a large cross track error. This situation is illustrated in Figure 6.2.

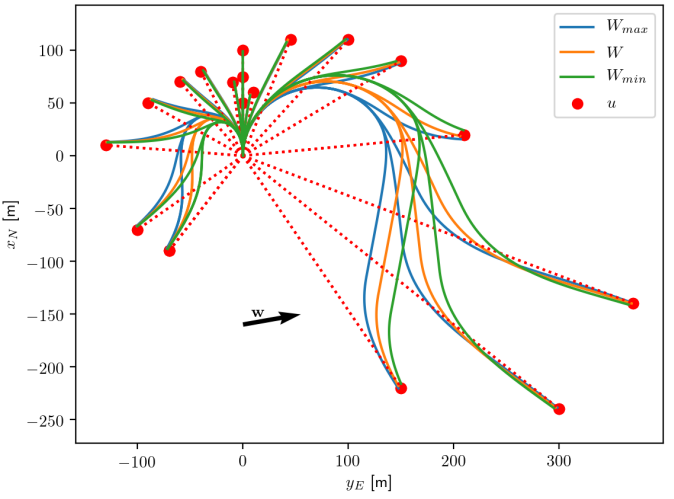
This issue was mitigated by defining a set ψ_{safe} as

$$\psi_{\text{safe}} = \{\psi : |\sin\psi| < \frac{1}{\sqrt{2}}\} \tag{6.4}$$

If $|\psi - \psi_w| \notin \psi_{\text{safe}}$ during expansion only inputs corresponding to $|\Delta\psi_{\text{cog}}| \leq 160°$ are used.



(a) *Inputs generated for* $\psi_w = 0°$



(b) *Inputs generated for* $\psi_w = 80°$

Figure 6.1: *Inputs for different wind directions,* $W \in [3.75, 6.25]$ *m/s*

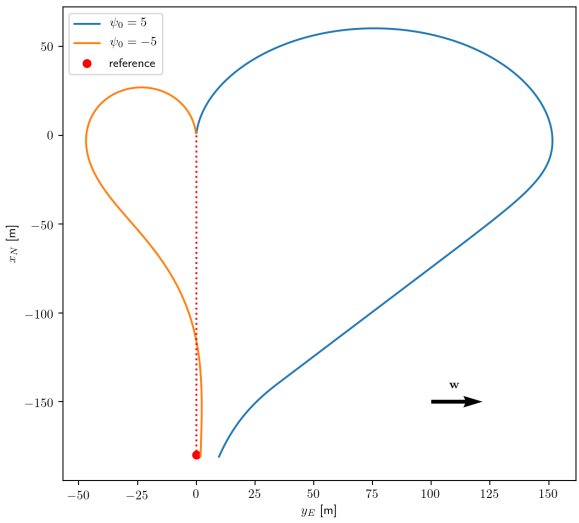


Figure 6.2: *Large cross track error for $\Delta\psi_{cog} \approx 180°$ when the wind is perpendicular to UAV motion*

6.1.5 Heuristic Lookup Table

The HLUT was generated using the method in Algorithm 2 presented in section 3.3.3, using the wind-direction $\psi_w = 0$. This implies that entries have to be generated for initial values of ψ from 0° to 180° to cover all possible situations. To query a stored heuristic value $\tilde{h}(x, \tilde{x})$ it is thus necessary to rotate both x and $\tilde{x}$ by the angle ψ_w in order for the query to align with the HLUT.

The set of states for which to generate entries was selected as

$$\mathcal{X} = \{(x_N, y_E) : |x_N| \leq D \cup |y_E| \leq D\} \tag{6.5}$$

for $D = 400$ m. To ensure that HLUT are entries are available for at least states within a smaller set with $D = 200$ m, an additional A^* search was performed for each such missing state after the initial generation. For $W = 5$ m/s the resulting HLUT consists of 951099 entries.

6.1.6 Waypoint controller

To send the calculated motion plan and landing sequence to the waypoint controller, these have to be converted to the MAVLink protocol which is supported by the ArduPlane autopilot [2]. This interface was implemented using the MAVROS plugin in ROS [16]. ROS is a modular framework for robotics applications, with API:s available in both Python and C++ [5].

Parameter	Value	Description
x_0	$(0, 0, 0°)$	Initial state
V_a	14 m/s	Airspeed
W	5 m/s	Wind speed
h_0	40 m	Initial altitude
h_{safe}	10 m	Landing area safety altitude
h_{flare}	3 m	Flare altitude
$\dot{h}_{\text{flare}}$	0.5 m/s	Flare sink-rate
$\dot{h}_{\max}$	3 m/s	Maximum sink-rate
$\dot{\psi}_{\max}$	17°/s	Maximum rate of turn
$\psi_{l,s}$	10°	Approach direction discretization

Table 6.1: *Simulation parameters*

6.1.7 Wind estimation

In simulated experiments, the wind was assumed to be perfectly estimated, *i.e.* the values of W and ψ_w configured in the simulator were also passed to the algorithm. During real flight experiments, the EKF based wind measurement system described in Section 2.3.2 was used to provide estimates. Since the wind is assumed constant in this work, a Moving Average (MA) filter with a window size of 2 seconds was used to remove small variations in the measurements.

6.2 Simulation experiments

In this section, setup and results of simulation experiments are presented.

6.2.1 Experimental setup

The proposed method was evaluated by performing a number of simulations in the Ardupilot Software In The Loop (SITL) environment [32]. This environment is based on the JSBSim flight dynamics simulator [1], and is capable of simulating wind effects. The default simulation model is based on the Rascal 110 fixed-wing UAV [43]. The parameters used during these simulations are summarized in Table 6.1. Simulations were performed on a Macbook Pro computer with a 2,5 GHz Dual-Core Intel Core i7 processor.

6.2.2 Results

A numher of landing sequences and the respective altitude profile between $\boldsymbol{p}_a$ and $\boldsymbol{p}_l$ are shown in Figure 6.3-6.6. Some relevant properties of the different solutions are summarized in Table 6.2. ψ_l^* and $R_a^* - R_l^*$ is the optimal approach direction and total landing distance for each given ψ_w. h_e^* and $|R_l^* - R_c|$ is the calculated entry altitude and distance from the landing point to the center of $\mathcal{A}$. h_e

is the actual entry altitude and $|R_l - R_l^*|$ the distance from the calculated landing point to the actual touchdown point of the UAV, both obtained from the simulation. Finally, T is the execution time of the entire landing sequence calculation.

$\psi_\mathbf{w}$	$\psi_\mathbf{l}^*$	$\mathbf{R_a^* - R_l^*}$	$\mathbf{h_e^*}$	$\mathbf{h_e}$	$\mathbf{\lvert R_l^* - R_c \rvert}$	$\mathbf{\lvert R_l - R_l^* \rvert}$	$\mathbf{T}$
0°	120°	272 m	17.09 m	9.4 m	0.95 m	4.84 m	0.04 s
90°	300°	232 m	19.45 m	12.68 m	1.49 m	5.87 m	0.32 s
180°	320°	244 m	19.64 m	11.5 m	1.44 m	1.48 m	0.97 s
270°	110°	222 m	20.4 m	13.76 m	1.71 m	5.92 m	0.08 s

Table 6.2: *Landing sequence solution properties*

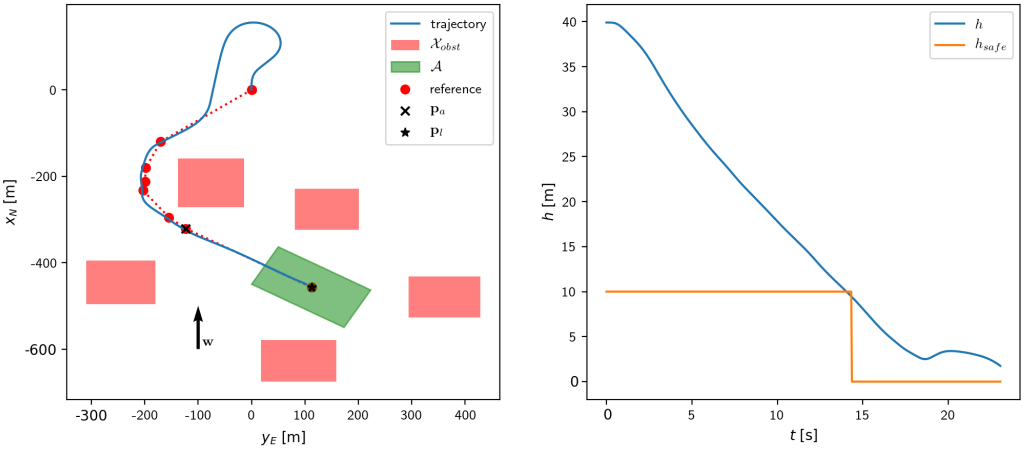


Figure 6.3: *Landing sequence and altitude profile for $\psi_w = 0°$*

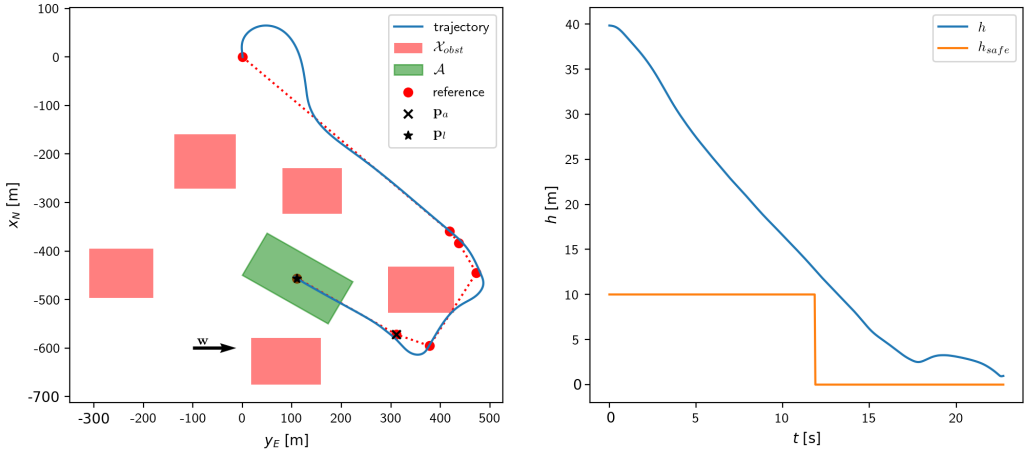


Figure 6.4: *Landing sequence and altitude profile for* $\psi_w = 90°$

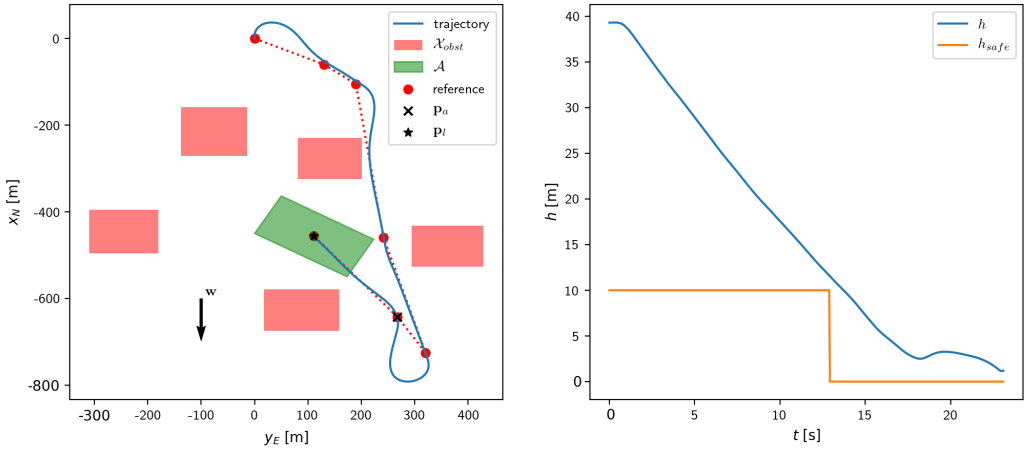


Figure 6.5: *Landing sequence and altitude profile for* $\psi_w = 180°$

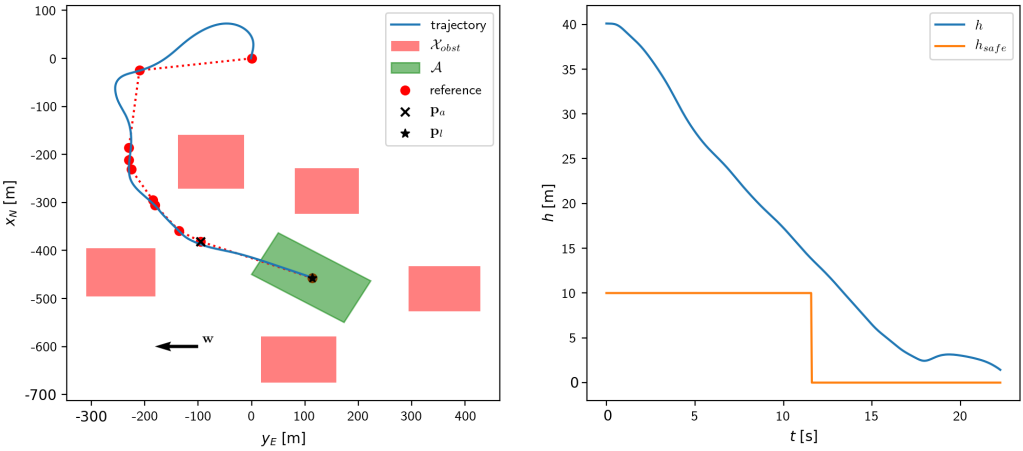


Figure 6.6: *Landing sequence and altitude profile for* $\psi_w = 270°$

6.3 Real flight experiments

In this section, setup and results of real flight experiments are presented.

6.3.1 Experimental setup

The UAV used during real flight experiments is shown in Figure 2.2. This platform is based on a Parrot Disco airframe [3], which was modified to use the PixRacer autopilot [4] with Arduplane flight control software [33]. The UAV is also equipped with a Raspberry Pi 3B+ companion computer on which the landing system was deployed. The companion computer communicates with an external command and control interface using a 4G-LTE modem.

Experiments were conducted in an area near Longitude 11.929/Latitude 54.486 south of Gothenburg, Sweden which is shown in Figure 6.7. The following procedure was followed during the experiments:

1. Launch and takeoff with the UAV
2. Put the UAV in circular movement around a pre-defined coordinate $\boldsymbol{p}_{\text{loiter}}$, until the wind measurement has converged to an almost constant value
3. Compute $\boldsymbol{p}_a$ and $\boldsymbol{p}_l$ using the estimated wind direction and speed
4. Compute a waypoint mission from x_0 to $x_g = (x_{N,A}, y_{E,A}, \psi_l^*)$ with x_0 determined as described below.
5. Send the computed mission to the autpilot and initiate mission execution

Figure 6.7: *Landing area used for real flight experiments. Map data from Open Streetmap.*

6.3.2 Determining the starting state

Since the motion planning algorithm assumes that the initial state of the UAV is exactly equal to the initial state used during planning, it is important that any positioning or heading errors are made as small as possible. This implies that $\boldsymbol{p}_{\text{loiter}}$ cannot be used directly as x_0 since the UAV circles around it, and thus the position and heading depends on when the landing sequence computation is initiated as well as the computation time which is uncertain. Hence the starting state was determined by defining

$$\boldsymbol{p}_{\pm 90^\circ} = \boldsymbol{p}_{\text{loiter}} + D\hat{r}_{\pm 90^\circ} \tag{6.6}$$

where $\hat{r}_{\pm 90^\circ}$ is a unit vector pointing in the direction $\psi_w \pm 90^\circ$ and D was set to 100 m. The starting state was then set to

$$x_0 = (x_{N,\pm 90^\circ}, y_{E,\pm 90^\circ}, \psi_w \pm 90^\circ) \tag{6.7}$$

depending on which of those points is closest to $\boldsymbol{p}_a$. This gives the UAV enough distance to reach the starting state exactly independent of where in the circular movement around $\boldsymbol{p}_{\text{loiter}}$ it is located when the mission is started.

6.3.3 Results

The trajectory and altitude profile of the UAV during a real landing is shown in Figure 6.8. The reported wind speed and direction from the Swedish meteorology

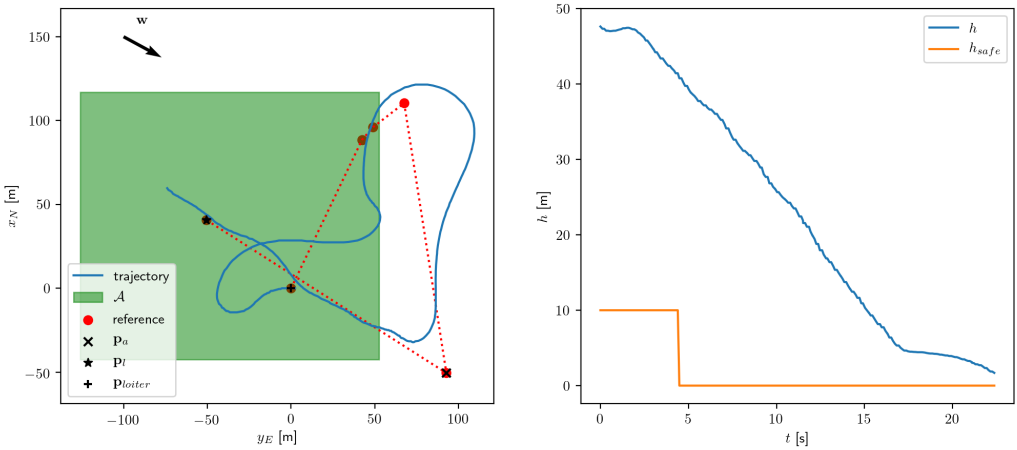


Figure 6.8: *Trajectory and altitude profile of the real uav during execution of a landing sequence. The first waypoint is the point $\boldsymbol{p}_{loiter}$ which the UAV circles around while the landing sequence is calculated. To compensate for the unpredictable initial heading when the landing sequence is initiated, the second waypoint is placed at a fixed distance from $\boldsymbol{p}_{loiter}$ in a direction perpendicular to the estimated ψ_w.*

instutite, SMHI at the time of the experiment was $W = 5$ m/s with gusts of 10 m/s, and $\psi_w \approx 120°$. The filtered estimates of both W and ψ_w during the flight are shown in Figure 6.9.

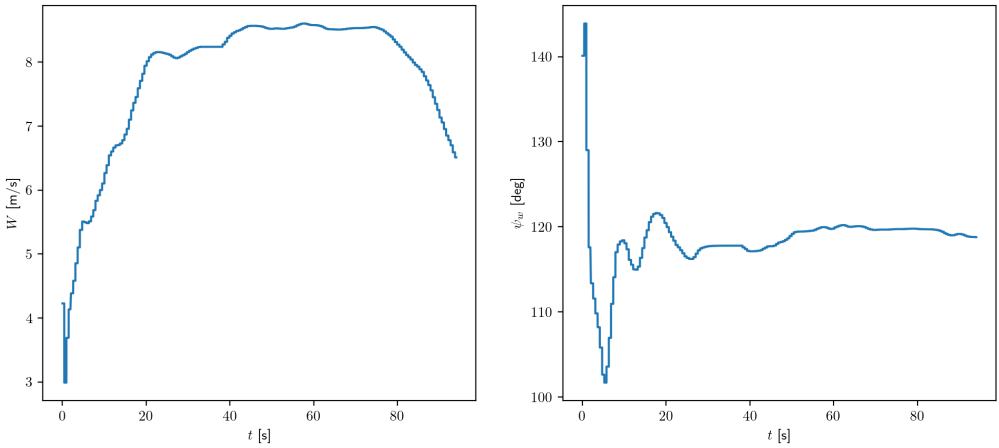


Figure 6.9: *Filtered estimates of W and ψ_w during real flight experiment. Both plots show values from the entire duration of the flight,* i.e. *from the moment the UAV takes of until it lands on the ground again.*

7

Discussion and conclusions

7.1 Results

In this work, a novel method to automatically generate landing sequences for fixed-wing UAVs is proposed. The method automatically handles many of the challenging aspects when specifying such a sequence manually, *e.g.* determining the current direction and speed of the wind.

The method consists of two main components, the *landing sequence calculation* and *motion planner*. Both these components mainly rely on optimization-based methods. To calculate the parameters determining the landing sequence, *i.e.* when the UAV descends to the ground, a non-linear optimization problem is formulated and solved numerically. The motion planner uses a set of pre-computed waypoints together with sampling-based planning techniques to determine a plan which is feasible given the UAV model and current wind conditions. To create the set of waypoints, an optimization problem is solved using derivative-free optimization.

The results from simulation experiments indicate that the method successfully generates feasible landing sequences in different wind conditions. The distance between the planned and actual landing point is negligible relative to the total distance of the landing sequence. The relative magnitude of the error in entry altitude is larger, but the UAV still manages to enter $\mathcal{A}$ above the specified h_{safe} in most cases. The error between calculated entry altitude and actual entry altitude seems quite constant, at least in the simulated evaluations summarized in Table 6.2. This implies that the error could be mitigated by estimating this offset and adding it to the desired h_{safe}. The error could also be mitigated by scaling the second term in the objective of Equation 5.15 with some constant $\lambda_h > 1$. The landing sequence generation is also quite fast, and a solution is found in well below 1 second in most cases. The method was also used in a real world scenario,

which resulted in a successful landing. In this case however, the distance between planned touchdown point and the actual was significantly larger than in the simulations. One reason for this, as can be seen in Figure 6.8, is a too large value of the parameter R_{wp} which determines how close the UAV should be to a target waypoint in order for it to be considered reached. To ensure a safe landing, the selected landing area for this experiment was quite large and thus the constraint on safe entry altitude was negligible. Nevertheless, the results are promising, and should be evaluated further by performing additional real flight experiments in more challenging scenarios.

The proposed method is quite general and could be implemented on any fixed-wing UAV which uses the trajectory controller described in Section 2.5.2. It would also be easy to extend it to support another controller, as the only requirement to create the input set $\mathcal{U}_s$ is that the closed-loop model of the system is written on the same form as Equation 4.6.

7.2 Limitations of the method

Constraining the control reference to consist of waypoints, *i.e.* straight line-segments, limits the system from using more complex trajectories like the ones used in [44]. It also introduces some issues mentioned in Chapter **??**. However, most popular autopilots such as Ardupilot use this formulation [33].

Sampling methods contain inherent limitations, such as the quality of the solution depending on the sampling resolution. In many cases, such as when generating a HLUT there is also a tradeoff between resolution and storage capacity. In the case of a 2-dimensional HLUT as in this thesis, the HLUT size scales quadratically with the sampling resolution used. However, calculating analytical solutions in realtime is often infeasible due to high computational costs.

A large limitation of the landing area definition in Chapter 5 is the assumption that the terrain elevation is constant inside the landing area $\mathcal{A}$. In most real-world cases, such as landing in a slope, the terrain elevation varies. Including this factor in the landing sequence generation would enable landings in a much wider class of scenarios.

The assumption that the wind is constant simplifies many parts of calculating the landing sequence, but such an approximation also introduces limitations in the performance of the method. As can be seen in Figure 6.9, the windspeed W is quite constant while the UAV is flying at a constant altitude. However, there is a clear altitude dependency in the wind-speed estimate, which is visible both during the takeoff and landing portions of the flight. It is therefore likely that including this altitude dependency, especially in the calculation of the final descent parameters $\boldsymbol{p}_a$ and $\boldsymbol{p}_l$ could increase the precision of the landing.

7.3 Future work

The landing sequence method depends on many different parameters, both airframe-specific such as $\dot{\psi}_{\max}$ and general such as discretization step-sizes and wind speeds

used for input generation. The goal of this thesis was mainly to evaluate the feasibility of the proposed method. Hence, most of those parameters were set to "good-enough" values which proved to be feasible but are not necessarily optimal. A possible future work consists of tuning these parameters for the currently used UAV platform, which would require a number of real-world flight experiments in different wind conditions. It would also be interesting to study methods to efficiently and automatically estimate optimal values of these parameters, especially those specific to the airframe.

As mentioned in the previous section, an important future work is to include terrain elevation in the landing sequence generation. Another important area is to study how an additional system mounted on the UAV could automatically detect suitable landing areas, *e.g.* using vision sensors and an elevation and obstacle database. This would be a step towards truly autonomous fixed-wing UAVs, as the system would be able to perform a safe landing without any pilot input. It could also be used to perform emergency landings if the command and control link to the pilot is lost.

Bibliography

[1] Jsbsim - an open source, platform-independent, flight dynamics & control software library in c++, 2019. URL http://jsbsim.org.

[2] Mavlink - micro air vehicle communication protocol, 2019. URL http://mavlink.io.

[3] Parrot disco fpv, 2019. URL https://www.parrot.com/us/drones/parrot-disco-fpv.

[4] Pixracer autpilot, 2019. URL https://docs.px4.io/v1.9.0/en/flight_controller/pixracer.html.

[5] Ros - robotic operating system, 2019. URL http://ros.org.

[6] S2geometry - computational geometry and spatial indexing on the sphere, 2019. URL https://s2geometry.io/.

[7] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[8] Charles Audet and J. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17:188–217, 01 2006. doi: 10.1137/060671267.

[9] Kristoffer Bergman. On motion planning using numerical optimal control, 2019. ISSN 0280-7971.

[10] John Adrian Bondy et al. *Graph theory with applications*, volume 290. 1976.

[11] Cornel-Alexandru Brezoescu. *Small lightweight aircraft navigation in the presence of wind*. PhD thesis, 2013.

[12] A. Cho, J. Kim, S. Lee, and C. Kee. Wind estimation and airspeed calibration using a uav with a single-antenna gps receiver and pitot tube. *IEEE Transactions on Aerospace and Electronic Systems*, 47(1):109–117, January 2011. ISSN 2371-9877. doi: 10.1109/TAES.2011.5705663.

[13] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. In *Fast motions in biomechanics and robotics*, pages 65–93. Springer, 2006.

[14] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *I. J. Robotic Res.*, 29:485–501, 04 2010. doi: 10.1177/0278364909359210.

[15] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957. ISSN 00029327, 10806377. URL http://www.jstor.org/stable/2372560.

[16] Vladimir Ermakov. Mavros - mavlink extendable communication node for ros with proxy for ground control station, 2019. URL http://wiki.ros.org/mavros.

[17] A. Gautam, P. B. Sujit, and S. Saripalli. A survey of autonomous landing techniques for uavs. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1210–1218, May 2014. doi: 10.1109/ICUAS.2014.6842377.

[18] F. Gustafsson. *Statistical Sensor Fusion*. Studentlitteratur, 2010. ISBN 9789144054896. URL https://books.google.se/books?id=yd_2SAAACAAJ.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136.

[20] M. Hwangbo, J. Kuffner, and T. Kanade. Efficient two-phase 3d motion planning for small fixed-wing uavs. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1035–1041, April 2007. doi: 10.1109/ROBOT.2007.363121.

[21] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug 1996. doi: 10.1109/70.508439.

[22] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables, Oct 2006.

[23] Steven M. La Valle. *Planning Algorithms*. 2006.

[24] Jack Langelaan, Nicholas Alley, and James Neidhoefer. Wind field estimation for small unmanned aerial vehicles. *Journal of Guidance, Control, and Dynamics*, 34:1016–1030, 07 2011. doi: 10.2514/1.52532.

[25] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[26] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the mads algorithm. *ACM Transactions on Mathematical Software*, 37(4):1–15, 2011.

[27] D. Lee and D. H. Shim. Rrt-based path planning for fixed-wing uavs with arrival time and approach direction constraints. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 317–328, May 2014. doi: 10.1109/ICUAS.2014.6842270.

[28] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pages 767–774, 2004.

[29] Timothy Mcgee, Stephen Spry, and J Hedrick. Optimal path planning in a constant wind with a bounded turning rate. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference*, 5, 08 2005. doi: 10.2514/6.2005-6186.

[30] Ardupilot: The open source autopilot. Arduplane auto flight mode, 2019. URL `http://ardupilot.org/plane/docs/auto-mode.html`.

[31] Ardupilot: The open source autopilot. Arduplane automatic landing, 2019. URL `http://ardupilot.org/plane/docs/automatic-landing.html`.

[32] Ardupilot: The open source autopilot. Ardupilot sitl advanced testing, 2019. URL `http://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html`.

[33] Ardupilot: The open source autopilot. Arduplane, 2019. URL `http://ardupilot.org/plane/index.html`.

[34] Ardupilot: The open source autopilot. Arduplane: Navigation tuning, 2019. URL `http://ardupilot.org/plane/docs/navigation-tuning.html`.

[35] Sanghyuk Park, John Deyst, and Jonathan How. *A New Nonlinear Guidance Logic for Trajectory Tracking*. doi: 10.2514/6.2004-4900. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2004-4900`.

[36] Linnea Persson. Cooperative control for landing a fixed-wing unmanned aerial vehicle on a ground vehicle, 2016.

[37] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[38] Luis Miguel Rios and Nikolaos V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, Jul 2013. ISSN 1573-2916. doi: 10.1007/s10898-012-9951-y. URL https://doi.org/10.1007/s10898-012-9951-y.

[39] Paul Riseborough. Estimation and control library, 2019. URL https://github.com/PX4/ecl/blob/master/EKF/documentation.

[40] Rolf Rysdyk. Course and heading changes in significant wind. *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, 33:1311–1312, 07 2010. doi: 10.2514/1.48934.

[41] Martin Selecky, Petr Váňa, Milan Rollo, and Tomas Meiser. Wind corrections in flight path planning. *International Journal of Advanced Robotic Systems*, 10:1, 05 2013. doi: 10.5772/56455.

[42] Laszlo Techy and Craig Woolsey. Minimum-time path planning for unmanned aerial vehicles in steady uniform winds. *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, 32:1736–1746, 11 2009. doi: 10.2514/1.44580.

[43] Tomáš Vogeltanz and Roman Jašek. Jsbsim library for flight dynamics modelling of a mini-uav. 2015.

[44] Michael Warren, Luis Mejias, Jonathan Kok, Xilin Yang, Luis Gonzalez, and Ben Upcroft. An automated emergency landing system for fixed-wing aircraft: Planning and control. *Journal of Field Robotics*, 32:1114–1140, 12 2015. doi: 10.1002/rob.21641.

[45] Andreas Wächter and Lorenz Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 03 2006. doi: 10.1007/s10107-004-0559-y.

[46] X. Zhang, A. Liniger, A. Sakai, and F. Borrelli. Autonomous parking using optimization-based collision avoidance. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4327–4332, Dec 2018. doi: 10.1109/CDC.2018.8619433.