Autonomous Robots manuscript No.
(will be inserted by the editor)

Real-Time Motion Planning with a Fixed-Wing UAV using an Agile Maneuver Space

Joshua M. Levin · Meyer Nahon · Aditya A. Paranjape

Received: date / Accepted: date

Abstract Small fixed-wing unmanned aerial vehicles (UAVs) are becoming increasingly capable of flying at low altitudes and in constrained environments. This paper addresses the problem of automating the flight of a fixed-wing UAV through highly constrained environments. The main contribution of this paper is the development of a maneuver space, integrating steady and transient agile maneuvers for a class of fixed-wing aircraft. The maneuver space is integrated into the Rapidly-Exploring Random Trees (RRT) algorithm. The RRT-based motion planner, together with a flight control system, is demonstrated in simulations and flight tests to efficiently generate and execute a motion plan through highly constrained 3D environments in real-time. The flight experiments – which effectively demonstrated the usage of three highly agile maneuvers – were conducted using only on-board sensing and computing.

Keywords Aerial robotics · Real-time motion planning · Agile flight · Control

Joshua M. Levin
Department of Mechanical Engineering
McGill University
Montreal, QC H3A 0C3
Tel.: +1 (647) 270 - 5909
E-mail: joshua.levin@mail.mcgill.ca

Meyer Nahon
Department of Mechanical Engineering
McGill University
Montreal, QC H3A 0C3

Aditya A. Paranjape
Tata Research, Development and Design Centre
Tata Consultancy Services Ltd.
Hadapsar, Pune 411013

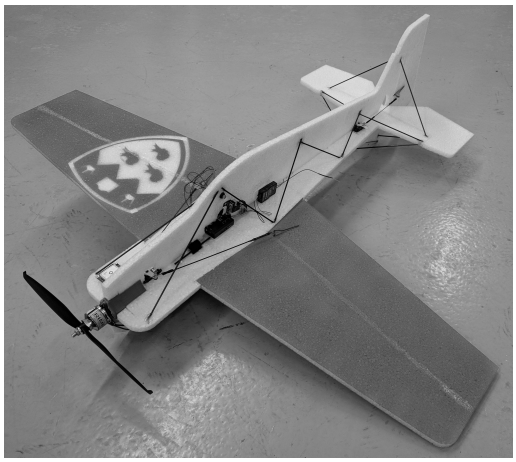


Fig. 1 A small agile fixed-wing UAV

1 Introduction

The research and development of autonomous unmanned aerial vehicles (UAVs) has been steadily growing over recent years due to the range of applications they are, and could potentially be, suitable for. Examples of jobs UAVs are filling include: search and rescue, aerial photography, road traffic monitoring, and pipeline monitoring. Many tasks for autonomous UAVs involve flight at high altitudes, in open airspace, where basic path planning techniques, such as waypoint following, are sufficient. With recent technological developments, however, UAVs are becoming increasingly competent at flying at low altitudes and in constrained environments, i.e. environments that are obstacle-dense and/or include challenging passages.

A class of UAVs that are well-suited to this type of flight are small agile fixed-wing UAVs (see Fig. 1). Among other characteristics, their high thrust-to-weight

ratio and powerful slipstream make them distinct from conventional fixed-wing aircraft. For years, remote control (RC) pilots have been flying these UAVs for leisure and competition. As the name suggests, these aircraft are highly agile, and thus require an extremely proficient pilot to control. More recently, these fixed-wing UAVs (along with other similar aircraft configurations) have been the subject of autonomous flight research. A significant portion of the existing literature has focused on dynamics modeling, trajectory generation, and control design techniques for hover transitions and perching maneuvers (Sobolic, 2009; Wickenheiser and Garcia, 2006, 2008; He et al, 2018). Maneuvers that make use of the lateral dynamics have also been investigated (Park, 2012; Barry, 2012; Selig, 2014). The primary value of the design of these aircraft for autonomous flight is their ability to combine efficient forward flight with high maneuverability. Many unique challenges accompany the pursuit of autonomous flight with these fixed-wings due to their large flight envelope and complex dynamic behavior; here we focus primarily on the problem of motion planning.

The specific planning problem we address in this work is that of efficiently generating a feasible motion plan in a highly constrained, three-dimensional, known environment with static obstacles. Sampling-based planning algorithms are well suited for efficient real-time planning with limited computational resources. The most prevalent sampling-based methods are the Probabilistic Roadmap (PRM) (Kavraki et al, 1996), Rapidly-Exploring Random Trees (RRT) (LaValle, 1998), and their variants (Karaman and Frazzoli, 2011b). The PRM algorithm is a multi-query algorithm that is probabilistically complete, but requires solving two-point boundary-value problems to steer the system between two states. Solving a boundary-value problem is a costly operation that is impractical for our application. The RRT algorithm is a single-query planner that is highly effective at generating dynamically feasible trajectories rapidly. The RRT algorithm handles complex constraints easily, finding a path to the goal region with minimal map exploration. As such, it is well-aligned with the aim of this work. Although the algorithm is not designed to produce optimal trajectories, our implementation ensures the resulting paths are smooth in the horizontal plane, i.e. continuity of heading is ensured. It is worth noting that an asymptotically optimal version of RRT has been developed, called RRT* (Karaman and Frazzoli, 2011b). Like PRM, RRT* requires solving costly boundary-value problems, and thus has not been pursued here.

A modern and prevalent alternative for robot planning and control is model predictive control (MPC), also known as receding horizon control (Kim et al, 2002; Schouwenaars et al, 2004a). The MPC planning method repeatedly solves an optimal control problem on-line, with obstacles as additional constraints. This method can be highly effective with linear and even nonlinear models, but the model we plan to take advantage of in this work is too complex to be used in a real-time optimization with the available on-board resources. Instead, we use the model for off-line pre-computations of maneuvers that can be used by the planner in real-time, in a framework similar to that which has been described as a 'maneuver automaton' (Frazzoli et al, 2005). The maneuver automaton, which captures formal properties of a trajectory library, is used as a hybrid representation of a vehicle model, wherein motion primitives are used to pre-compute a cost-to-go map. In Frazzoli et al (2002), the states of the automaton are trim states of the vehicle, and maneuvers are used to transition between the states. In Gavrilets et al (2001), the concept is explored for learning motion primitives from human-piloted aerobatic flight, and in Schouwenaars et al (2004b), the hybrid model replaces trim states with 'linear time-invariant modes' and fixed-duration transitions. We employ a similar hybrid representation concept here, where the model of the aircraft's dynamics is represented by a maneuver space made up of motion primitives. Each motion primitive we generate pairs a dynamically feasible trajectory with its associated feedforward control inputs. The size of the maneuver space (i.e. the number of motion primitives) was chosen to balance our need to manage limited computational resources with our goal of representing a significant portion of the aircraft's flight envelope. The maneuver space consists of trim states, as well as the three agile maneuvers developed in a paper currently under review for publication.

Prior literature includes a number of similar planning methodologies for UAVs. In Lee and Shim (2014), a pre-defined motion primitive set is used for 2D RRT-based path planning with a fixed-wing UAV. Pre-computed motion primitives are used in an A*-based planner in MacAllister et al (2013). In Allen and Pavone (2015), a real-time framework was developed, which incorporates a look-up table of boundary-value problem solutions into a sampling-based algorithm called Fast Marching Trees (FMT*). Trajectory funnels for robust motion planning were applied to a highly maneuverable fixed-wing aircraft in Majumdar and Tedrake (2017).

The key contribution of this paper is the development of the maneuver space and its implementation within the RRT-based algorithm. Relative to the existing literature, these contributions differentiate this work in the following ways. The dynamic feasibility constraints used to generate the maneuver space rep-

resent a high-fidelity, physics-based model. The model is used to generate aggressive and agile motion primitives that fully exploit the aircraft's flight envelope. The trajectory solutions are composed of the aircraft's full 12-state vector, as well as all of its control inputs. Additionally, we note that by the way in which the motion primitives are integrated into the planning algorithm, the size of the library has no effect on planning time; typically, the computational cost of RRT increases with the number of primitives (Vieira and Grassi, 2014). Furthermore, our implementation places no constraints on the sequencing of primitives. Eliminating the need for an extremely large set of pre-defined transition maneuvers, a transition heuristic is applied to allow any primitive to follow another. With respect to the existing literature, the contributions work to produce an efficient motion planner that is capable of generating more aggressive trajectories, with high-fidelity optimal control policies.

This paper builds on the authors' previous work. The dynamics of these UAVs are highly nonlinear, and cannot accurately be represented with linear models or the traditional fixed-wing stability-derivatives approach. We instead rely on a full nonlinear six degree-of-freedom model developed by Khan and Nahon (Khan, 2016). In a work that is currently under review for publication, we used this model to develop trajectory generation and control strategies for aggressive agile maneuvers. Here, we aim to incorporate the control system and these maneuvers into a planning framework. Note that another agile maneuver, the knife-edge, was investigated by the authors in Levin et al (2017), however it does not fit the framework of the motion planning algorithm developed here. During flight, the feedforward control inputs of the primitives are paired with a stabilizing feedback controller that accounts for errors, inaccuracies, noise, and disturbances. This controller was first presented in Bulka and Nahon (2018), and consists of a position tracker, a quaternion-based attitude tracker, and a thrust controller. In a number of ways, the work here is a significant extension of Levin et al (2018a), in which the basic motion planning framework was originally proposed. With respect to this work, the new developments here are as follows:

1. The planning algorithm is augmented to run in real-time and select trajectories intelligently.
2. A transition maneuver heuristic is included in the planning framework to switch between motion primitives.
3. The planner is compared in simulations to a baseline approach that uses Dubins curves.
4. More extensive flight testing is performed and discussed in depth. The flight tests are the first to demonstrate their level of implementation, in terms of the extent of the flight envelope utilized, and fully relied on on-board sensing and computing.

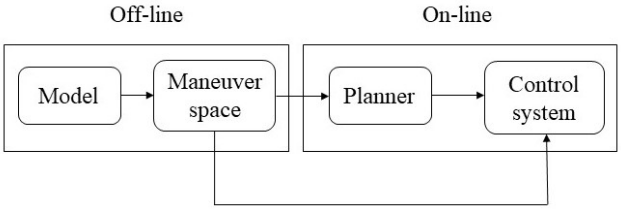


Fig. 2 High-level automation architecture

The paper is organized as follows. In Section 2, we present a high-level view of the various algorithmic components brought together in the remainder of the paper. The aircraft configuration and model are described in Section 3. Section 4 details the development of the maneuver space, and the feedback controller is briefly described in Section 5. The motion planning algorithm is presented in Section 6, with a focus on the integration of the maneuver space. In Section 7, simulations are run to compare the planner to a baseline approach (using Dubins curves), and evaluate trajectory tracking performance. Flight test experiments are demonstrated in Section 8, and concluding remarks follow in Section 9.

2 High-Level Automation Architecture

The high-level automation architecture can be partitioned into off-line and on-line components, as seen in Fig. 2. In the off-line process, the model is used, in Section 4, within an optimization framework – in effect, as a set of dynamic constraints – to generate a set of maneuvers, also termed 'motion primitives'. Without the need to be executed in real-time, the model – without any simplifications – can be used to ensure the generated maneuvers are optimized and dynamically feasible. We call the set of maneuvers a 'maneuver space'.

The planner and control system both run on-board the aircraft. The RRT-based planner begins running once the aircraft is in its initial condition, so that this state can be measured and used in the algorithm. The plan is generated and iteratively sent to the control system, which proceeds to execute it.

The maneuver space is used mainly by the control system. All the relevant information detailing the maneuvers is stored on-board the aircraft's autopilot and accessed by the control system therein. In short, the planner, described in Section 6, tells the control system which motion primitive to execute at a given time,

and the controller seeks out this primitive from the maneuver space to compute the exact trajectory to track, and feedforward control inputs to use. The maneuver space is also used by the planner, although the planner does not need to store its entire contents. By way of example, unlike the control system, the planner does not need to know the time-dependent trajectory, nor the control inputs corresponding to an agile maneuver. It simply needs to know where the maneuver begins and ends, and with what particular heading.

3 Aircraft

The methodology described in this paper is applied to a small agile fixed-wing UAV (Fig. 1). This UAV belongs to a class of aircraft that are characterized by such features as a high thrust-to-weight ratio, a powerful slipstream, and a low aspect ratio. All together, the characteristics of these aircraft make them extremely maneuverable. A notable example of their maneuverability is that they are able to transition into and hold a nose-up hover; thus they can maintain a stationary airborne configuration.

3.1 Dynamics Model

A comprehensive model of the aircraft's dynamics is used in this work to build the maneuver space and to run simulations. The model is largely derived from the work of Khan and Nahon. The full details of the model can be found in Khan and Nahon (2013) for the thruster dynamics, Khan and Nahon (2015a) for the slipstream modeling, and Khan and Nahon (2015b) for the nonlinear aerodynamic modeling. A brief summary of the model can also be found in Levin et al (2018b). Here, a few notes on the most relevant aspects of the model will be provided.

For the purpose of calculating the forces and moments, the model is broken down into three main sections, for the thruster, the slipstream, and the aerodynamics. The thruster model uses blade element momentum theory to compute the aerodynamic and gyroscopic forces and moments produced by the motor and propeller. This model captures static, axial, oblique, and reverse flow conditions. It accounts for the battery, electronic speed controller, motor, and propeller. The thruster model was experimentally validated in Khan and Nahon (2013).

The model in Khan and Nahon (2015a) computes the velocity field and swirl effect that represent the slipstream (also known as 'propwash'). The velocity of the additional airflow of the slipstream can be as great as, if not greater than the speed of the aircraft itself, and has a significant effect on the aircraft's dynamics. The main reason the slipstream effect is so important for agile fixed-wings is that the additional airflow enhances the control authority of the aircraft's flaps. It is this effect alone that grants the aircraft the ability to stabilize itself in a hover, when it has zero groundspeed. In Levin et al (2018b), slipstream effects are quantified for an agile fixed-wing UAV.

The aerodynamic modeling uses a component breakdown approach, which splits the aircraft's main components (wing, tail, rudder, and fuselage) into segments whose aerodynamics are computed independently and then summed together. This method is employed because aerodynamic effects vary over the span of the aircraft. The model includes lift, drag, and moment coefficients that adjust for low- and high-angle-of-attack regimes. Effects of aspect ratio, stall, control surface deflection, bound vortices, and trailing vortices are all accounted for in the model, as described in Khan and Nahon (2015b).

3.2 Aircraft Configuration

Model parameters are selected in accordance with the aircraft platform used for flight testing, see Fig. 1. For the airframe of the UAV, we use the RC plane model *McFoamy* by *West Michigan Park Flyers*, which has a 0.86 m wingspan. Sitting on the nose of the plane is an *Electrifly PowerFlow* 10×4.5 propeller, attached to a *RimFire 400 Outrunner* brushless DC Motor by *Great Planes*. The motor is controlled via an *Electrifly Silver Series* 25A brushless electronic speed controller. Metal gear servomechanisms actuate the control surfaces, ailerons, elevator, and rudder. The on-board computing equipment consists of a *Pixhawk Mini* autopilot and an *ODROID XU4*, which will both be discussed in further detail in Section 8. The *Pixhawk Mini* sits on top of the aircraft's body, and the *ODROID* is fixed underneath. All electronic components on the aircraft are powered by an 11.1 V lithium polymer battery. Fully equipped, the aircraft weighs 0.576 kg.

4 Maneuver Space

The maneuver space is a key aspect of the planner because it allows the planning algorithm to generate dynamically feasible trajectories – which exploit the aircraft's full flight envelope – in real-time, without having to solve complex dynamic constraints. It acts as a hybrid representation of the aircraft's dynamics, in place of the nonlinear ordinary differential equations.

The maneuver space consists of finitely many motion primitives, which are dynamically feasible trajectories and their associated feedforward control inputs. The set of primitives, which are generated off-line, is large enough to represent a significant portion of the aircraft's flight envelope, and compact enough to fit in limited computational resources. The motion primitives are classified as either trim primitives or agile maneuver primitives. Trim primitives are steady maneuvers with constant control inputs that can be coasted along indefinitely. Agile maneuver primitives are finite-time transitions that accomplish a specific, purposeful change in the aircraft's configuration.

4.1 Trim Primitives

Trim primitives are the aircraft's basic flight modes, and the set of them make up the greater portion of the maneuver space. Because the feedforward control inputs used to hold them remain constant, trim primitives can be used by the planner for any length of time. The trim primitives included in the maneuver space are:

- straight and level flight
- climbs/descents
- banked turns
- helical banked turns
- hover

Trim primitives can be solved for in various ways; for example, using MATLAB's *fsolve* function, or a bifurcation analysis (Ananthkrishnan and Sinha, 2001). With the exception of hover, all trim primitives are found here by solving a trajectory optimization problem. The exact form of the cost function used to solve for the trim primitives is not relevant, except to preclude overactuation if there are multiple sets of control inputs that can be used to achieve the same trim condition.

$$\min\ J \triangleq \int_0^{t_f} \left(\delta_a^2 + \delta_e^2 + \delta_r^2 + \left(\frac{\omega_T}{8000}\right)^2 \right) dt$$

subject to the first-order dynamics of the aircraft,

and the path constraints:

$$\begin{aligned} V = V_d, \quad \dot{\phi} = \dot{\theta} = 0, \quad \dot{\psi} = \dot{\psi}_d, \quad \dot{z} = \dot{z}_d \qquad (1)\\ \delta_a \in [-\delta_{a_{max}}, \delta_{a_{max}}], \qquad \dot{\delta_a} = 0^\circ\,\mathrm{s}^{-1}\\ \delta_e \in [-\delta_{e_{max}}, \delta_{e_{max}}], \qquad \dot{\delta_e} = 0^\circ\,\mathrm{s}^{-1}\\ \delta_r \in [-\delta_{r_{max}}, \delta_{r_{max}}], \qquad \dot{\delta_r} = 0^\circ\,\mathrm{s}^{-1}\\ \omega_T \in [\omega_{T_{min}}, \omega_{T_{max}}], \qquad \dot{\omega}_T = 0\ \mathrm{rpm/s} \end{aligned}$$

The weight in the denominator of the last term of the cost function makes the penalty on thrust roughly similar in magnitude to that on the control surface inputs, so that it does not dominate the cost. The weight is proportional to the control input units, radians for control surfaces and rpms for thrust. The control inputs, ailerons, δ_a, elevator, δ_e, rudder, δ_r, and thrust, ω_T, may take on any value within their physical limits, but constraints are put on their derivatives so that they remain constant. The desired speed, V_d, is set to a constant 7 $\mathrm{m\,s^{-1}}$, which is a normal cruising speed for this aircraft. The roll and pitch rates, $\dot{\phi}$ and $\dot{\theta}$, are set to be zero so that the maneuvers are steady. The optimization problems are solved here using GPOPS-II (Patterson and Rao, 2014), which is MATLAB-based general purpose optimal control software.

As mentioned, the planner is developed for constant speed flight, except when hovering and when performing agile maneuvers. Fixed-wing aircraft can maximize their range and endurance predictably at specific flight speeds, and the aircraft would navigate close to some such suitable speed. Operating at a specific flight speed is consistent with the objective of the motion planner, which is to guide the aircraft to a desired goal region. The objective does not, for instance, include arrival time constraints. While further agility could be harnessed by loosening constraints on the flight speed, doing so would increase the size of the maneuver space and presumably result in a more difficult trajectory tracking problem. Exploring various flight speeds is a worthwhile endeavor for future work, but was considered superfluous in this context.

Each trim primitive, except hover, is obtained for different combinations of the desired yaw rate, $\dot{\psi}_d$, and climb/descent rate, $\dot{z}_d$. For example, straight and level flight sets both of these values to zero. To keep the size of the maneuver space compact, primitives are solved for in incremental values. For the yaw rate, we find primitives from -110 $^\circ\,\mathrm{s^{-1}}$ to 110 $^\circ\,\mathrm{s^{-1}}$ in increments of 10 $^\circ\,\mathrm{s^{-1}}$ and for the climb/descent rate we sample from -2 $\mathrm{m\,s^{-1}}$ to 2 $\mathrm{m\,s^{-1}}$ in increments of 1 $\mathrm{m\,s^{-1}}$. There are a total of 116 trim primitives; straight and level flight, 4 climbs/descents, 22 banked turns, 88 helical banked turns, and the hover. The states and control inputs that define each maneuver are stored in a look-up table used by the control system. The look-up table also includes the states and control inputs of the hover primitive, which are determined via solution of the aircraft's equations of motion.

4.2 Agile Maneuver Primitives

Agile maneuver primitives enhance the maneuver space – and thus the motion planner – with functional changes of the aircraft's pose that could not otherwise be achieved,

or at least not as effectively, using trim primitives. In contrast to the trim primitives, an agile maneuver must be executed over a pre-computed finite amount of time, and the states and control inputs are time-dependent. Three agile maneuvers were developed for use in the maneuver space: a cruise-to-hover transition (CTH), a hover-to-cruise transition (HTC), and an aggressive turn-around (ATA). The cruise-to-hover transition, as its name suggests, transitions the aircraft from straight and level flight (cruise) to a hover, and the hover-to-cruise transition performs the reverse maneuver. Together, these two maneuvers allow the aircraft to start and stop in a hover. The aggressive turn-around maneuver rapidly reverses the aircraft's heading. With this maneuver in its repertoire, the aircraft can turn away from dead ends with only one primitive, and using less space than would be required by piecing together even the most aggressive trim primitives.

The agile maneuver primitives are also found by solving trajectory optimization problems:

$$\min\ J \triangleq 4t_f + \int_0^{t_f} \left(\left(\frac{\dot{\delta}_a}{10}\right)^2 + \left(\frac{\dot{\delta}_e}{10}\right)^2 + \left(\frac{\dot{\delta}_r}{10}\right)^2 + \left(\frac{\dot{\omega}_T}{2000}\right)^2 \right) dt$$

subject to the first-order dynamics of the aircraft, the path constraints:

$$\begin{aligned} \delta_a &\in [-\delta_{a_{max}}, \delta_{a_{max}}], & \dot{\delta}_a &\in [-\dot{\delta}_{a_{max}}, \dot{\delta}_{a_{max}}] \\ \delta_e &\in [-\delta_{e_{max}}, \delta_{e_{max}}], & \dot{\delta}_e &\in [-\dot{\delta}_{e_{max}}, \dot{\delta}_{e_{max}}] \\ \delta_r &\in [-\delta_{r_{max}}, \delta_{r_{max}}], & \dot{\delta}_r &\in [-\dot{\delta}_{r_{max}}, \dot{\delta}_{r_{max}}] \\ \omega_T &\in [\omega_{T_{min}}, \omega_{T_{max}}], & \dot{\omega}_T &\in [-\dot{\omega}_{T_{max}}, \dot{\omega}_{T_{max}}] \end{aligned} \tag{2}$$

and the boundary conditions (see Table 1):

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{x}_0, & \mathbf{u}(0) &= \mathbf{u}_0 \\ \mathbf{x}(t_f) &= \mathbf{x}_f, & \mathbf{u}(t_f) &= \mathbf{u}_f \end{aligned}$$

Eq. 2 includes a minimum-time cost function that additionally penalizes control inputs rates to produce smooth control input time histories. It is well known that abrupt changes in control inputs can be detrimental to a mechanical system, and smooth inputs tend to result in more robust trajectories. The weights in the cost function are again proportional to units and aim to balance the penalties on each control input. The maneuvers are no longer steady, but the control inputs and their derivatives are constrained by the physical limits of the aircraft, namely, the limits and rate limits of the motor and control surface servomechanisms. Additionally, the maneuvers are finite-time transitions and thus must satisfy boundary conditions on the state and control vectors, $\mathbf{x} = [u,\ v,\ w,\ p,\ q,\ r,\ q_1,\ q_2,\ q_3,\ q_4,\ x,\ y,\ z]^\mathsf{T}$ and $\mathbf{u} = [\delta_a, \delta_e, \delta_r, \omega_T]^\mathsf{T}$, at $t = 0$ and $t = t_f$. The boundary conditions for each maneuver are listed in Table 1. Euler angles are listed in the table, in place of the actual nonlinear constraints on quaternions, to give an intuitive representation of the problem.

The solutions to each problem come in the form of time-dependent reference trajectories and feedforward control inputs. These are stored in matrices that are interpolated by time in the control system. Figure 3 displays visualizations of the three agile maneuver trajectories. The three maneuvers alone prove to be useful for motion planning, and simple to integrate into the framework without compromising any of its attributes – most notably computational efficiency.

4.3 Transitioning Between Primitives

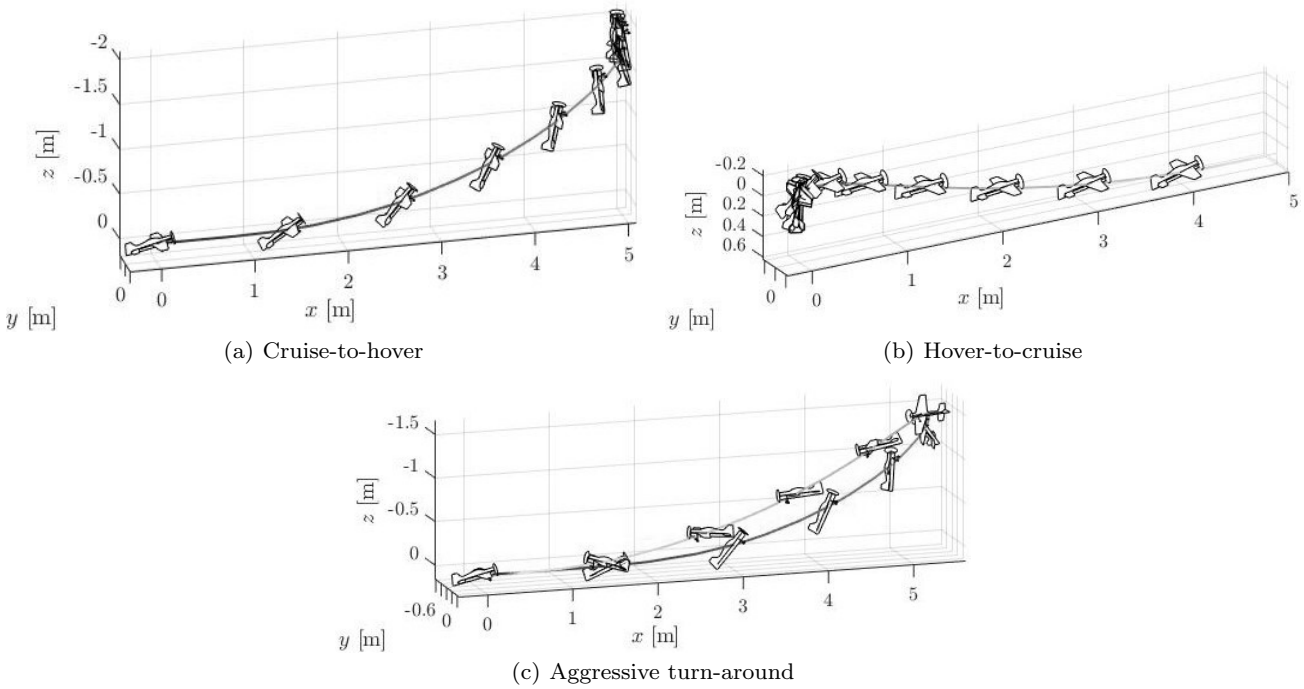
The aircraft has finite agility and thus requires time to transition from one primitive to another. In theory, finite-time transition primitives could be used, but to connect the 116 trim primitives alone would require a massive expansion of the maneuver space. Instead, we implement a transition maneuver heuristic in the planner and control system.

To smooth the transition between any two primitives (trim or agile), a time-delay heuristic is implemented. For the duration of the time-delay, the feedforward inputs and reference trajectory of the subsequent primitive are commanded, *except* for the path and heading, which are extended from the previous primitive. Consider Fig. 4, in which A to B is one trim primitive, and B' to C is another. The trajectory from B to B' is the transition maneuver. The maneuver extends the curvature of the path and heading of the A to B primitive, while all steady states tracked and control inputs commanded are that of the B' to C primitive. This measure is performed by the motion planner, rather than as a post-processing step, meaning that the tracked trajectory is equivalent to the desired/planned one.

The rationale behind the transition maneuver is based on the time-scale separation principle, as it applies to the physics of fixed-wing flight (Snell et al, 1992); in particular, the modal time-scales of the fast rotational and slow translational dynamics. Accordingly, we diminish position tracking errors by allowing the aircraft a short amount of time to continue along the path predicted by its current motion as it begins to transition into the steady states of the next primitive. What this enables is a transition that avoids sudden changes in variables that cannot react fast enough, while preparing the fast variables for the next state. The time-scale separation principle is mirrored in the design of the feedback controller, which has independent inner (attitude) and outer (position) control loops.

Table 1 Boundary conditions for agile maneuver primitives. Straight and level trim conditions denoted by subscript $_{SL}$, hover trim conditions denoted by subscript $_{H}$.

States and Controls	Boundary Conditions					
	ATA		CTH		HTC	
	$t=0$	$t=t_f$	$t=0$	$t=t_f$	$t=0$	$t=t_f$
$\boldsymbol{u}$	u_{SL}	u_{SL}	u_{SL}	0	0	u_{SL}
$\boldsymbol{v}$	0	0	0	0	0	0
$\boldsymbol{w}$	w_{SL}	w_{SL}	w_{SL}	0	0	w_{SL}
$\boldsymbol{p,q,r,\phi}$	0	0	0	0	0	0
$\boldsymbol{\theta}$	θ_{SL}	θ_{SL}	θ_{SL}	90°	90°	θ_{SL}
$\boldsymbol{\psi}$	0	π	0	0	0	0
$\boldsymbol{x}$	0	0	0	–	0	–
$\boldsymbol{y}$	0	0	0	0	0	0
$\boldsymbol{z}$	0	0	0	–	0	–
$\boldsymbol{\delta}_a$	0	0	0	0	0	0
$\boldsymbol{\delta}_e$	$\delta_{e_{SL}}$	$\delta_{e_{SL}}$	$\delta_{e_{SL}}$	0	0	$\delta_{e_{SL}}$
$\boldsymbol{\delta}_r$	0	0	0	0	0	0
$\boldsymbol{\omega}_T$	$\omega_{T_{SL}}$	$\omega_{T_{SL}}$	$\omega_{T_{SL}}$	ω_{T_H}	ω_{T_H}	$\omega_{T_{SL}}$



(a) Cruise-to-hover

(b) Hover-to-cruise

(c) Aggressive turn-around

Fig. 3 Agile maneuver trajectories. The aircraft are drawn slightly smaller than to scale, and the path lines progress from darker to lighter grey with time

An analysis of the aircraft's dynamics is used to determine the duration of the delay. Essentially, we want to determine the time it typically takes for the roll angle to reach a commanded value. It is the roll dynamics which should dominate the calculation of the duration because the roll angles are undergoing large discontinuous changes. Pitch angles are also discontinuously commanded, but the changes are relatively small. Under the control system (the combination of feedforward inputs and feedback control laws), the roll dynamics behave similarly to a low-pass filter of the form $\frac{1}{\tau s+1}$. This low-pass filter may be viewed as the first-order Padé approximation of a time-delayed input, t_d (Kuo and Golnaraghi, 2003, p.183). For step input commands (as is the case for the change in commanded roll angle between primitives), it can be shown that τ is a suitable value of the time-delayed input, t_d (Paranjape et al, 2015).

To determine the value of τ, simulations of the aircraft model and control system were run in Simulink. Step input commands for roll were given, and the outputs of these commands superimposed with low-pass

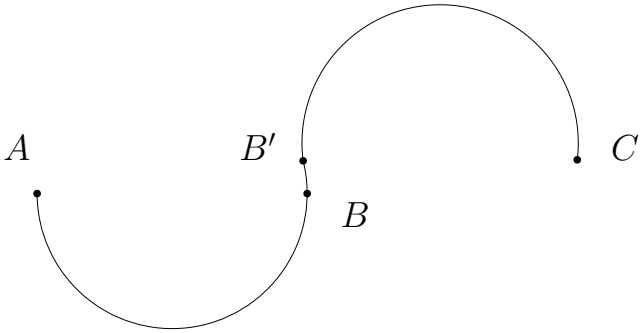


Fig. 4 Structure of a transition maneuver. A to B is one trim primitive, B to B' is the transition maneuver, and B' to C is the subsequent primitive

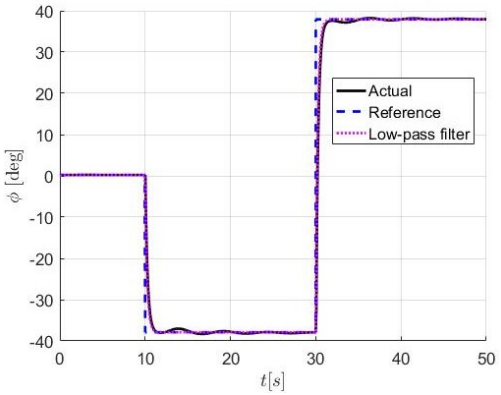


Fig. 5 Comparison of actual roll dynamics to low-pass filter

filters was observed. The value of τ in the low-pass filter was tuned until the output of the filter closely matched the aircraft's actual dynamics under the control system. The value of τ found this way, 0.23 s (at $V = 7\text{m}\,\text{s}^{-1}$), was given to the transition trajectory time-delay constant, t_d. Fig. 5 shows the simulated roll dynamics as the controller tracks a pre-defined motion plan that involves three trim primitives. During the first 10 seconds, the reference trajectory is the straight and level primitive. During the next 20 seconds, a banked turn at a rate of $\dot{\psi} = -60^\circ\,\text{s}^{-1}$ is commanded; and finally a banked turn with $\dot{\psi} = 60^\circ\,\text{s}^{-1}$ is commanded. The blue line is the commanded roll angle throughout the plan, and the black line shows the actual roll, under the control system. The final line, in magenta, plots the reference roll angle having gone through the low-pass filter. As can be seen, with $\tau = 0.23$ s, the actual roll dynamics are approximated closely by the low-pass filter.

The transition maneuver heuristic was also validated in trajectory tracking simulations. A series of primitives were sequenced in the following order: straight and level flight, a banked turn to the left, and a banked turn to the right. Using the feedback controller and simulation environment (described in Sections 5 and 7.2, respectively), the primitive sequence was tracked; first with

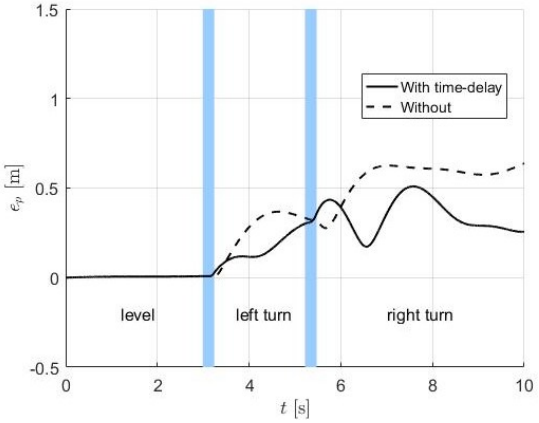


Fig. 6 Position tracking errors with and without using the time-delay transition maneuvers, for turns using $\dot{\psi} = \pm 30^\circ\,\text{s}^{-1}$. The background is colored light blue during the time periods when the transition maneuver is being executed

and then without including the transition maneuver. Two sets of trials were conducted, one using turns of $\dot{\psi} = 30^\circ\,\text{s}^{-1}$, and another using turns of $\dot{\psi} = 60^\circ\,\text{s}^{-1}$. In both cases, the transition maneuver reduced the overall position tracking errors by approximately 35%. With the $30^\circ\,\text{s}^{-1}$ turns, the root-mean-square error (RMSE) on position was reduced from 0.33 m to 0.21 m, and with $60^\circ\,\text{s}^{-1}$ turns, the error was reduced from 0.44 m to 0.28 m. The position tracking errors for the first case, $\dot{\psi} = \pm 30^\circ\,\text{s}^{-1}$, are plotted in Fig. 6. The plot illustrates how the position tracking performance is improved by using the time-delayed transitions. Notice as well that the position error, e_p, remains stable throughout the transition maneuvers (the light blue sections).

5 Feedback Controller

The control system is tasked with tracking the reference trajectories generated by the motion planner. The controller combines feedforward and feedback control inputs. The feedforward control inputs are associated with the primitives, and thus come from the motion planner. The control system, as implemented on the *Pixhawk Mini*, reads in a list of maneuvers sent by the planner, and uses the information stored in the maneuver space to interpret the data in the list as full-state time-dependent trajectories and feedforward control inputs. The feedback controller is intended to account for modeling inaccuracies, external disturbances, and sensor measurement noise. It is made up of three components: a position tracker, a quaternion-based attitude tracker, and a thrust controller. The feedback controller is discussed at length in Bulka and Nahon (2018), and a brief summary follows here.

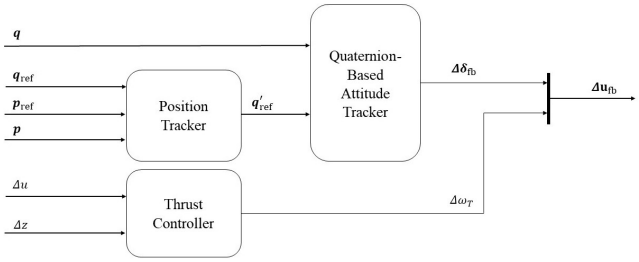


Fig. 7 Block diagram of feedback controller

The position tracker takes the desired attitude as input and modifies it in accordance with errors such that the direction of the aircraft's thruster points towards the desired position. The attitude tracker diminishes quaternion-based attitude errors, using control laws largely grounded in the aerodynamic properties of the UAV to actuate the control surfaces. The control laws in the attitude tracker account for the local airflow – due to airspeed *and* slipstream – over the control surfaces. The thrust controller uses PID control laws to regulate forward speed and altitude. A basic block diagram of the feedback controller architecture is presented in Fig. 7; where $\mathbf{q}$ and $\mathbf{p}$ are the attitude quaternion and position, respectively, and u and z are the forward speed and altitude. The term $\mathbf{q}'_{\text{ref}}$ represents the modified desired attitude. The full feedback control inputs, $\Delta\mathbf{u}_{fb}$, include the control surface inputs, $\Delta\boldsymbol{\delta}_{fb}$, and thrust input, $\Delta\omega_T$.

6 Motion Planner

The motion planner is based on the RRT algorithm, which is a single-query planning method that efficiently explores an environment such that a feasible path to the goal region can be constructed rapidly. A tree is built by steering towards randomly generated points until the goal region is reached. Steering is done using the dynamically feasible trajectories of the maneuver space, which is implemented as a library of trajectories. The algorithm is run in real-time and initiates with the aircraft's actual configuration. The ways in which the planner deviates from the standard RRT algorithm will be discussed in this section. The deviations mainly center around incorporating the library, and using the agile maneuvers intelligently. A pseudo-code version of the high-level algorithm is presented in Algorithm 1. Note that the algorithm is specifically set up to guide the aircraft from an initial hover to a hover in the goal region.

Algorithm 1: RRT

```
input: Map, initial configuration (x_i)
Initialize tree with x_i
Generate hover-to-cruise primitive from x_i via
 SteerAgile (HTC)
while the goal region has not been reached do
    while time interval has not elapsed do
        Generate a random point in the map, p_rand
        ExtendTree towards p_rand (Algorithm 2)
    end
    UpdateTree (Algorithm 3)
end
```

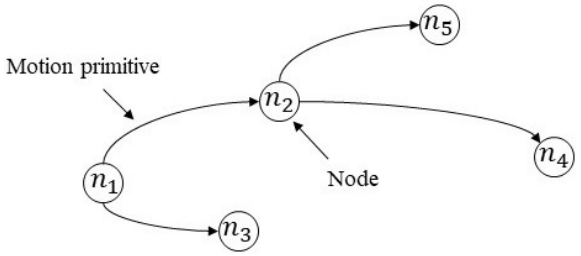


Fig. 8 Nodes and motion primitives

6.1 Tree Data Structure

The tree that is built by the planner consists of nodes, each of which defines the state of the aircraft and the type of motion primitive that precedes it. In Fig. 8, the node n_2, for example, contains not only the pose and time when the aircraft should reach it, but the edge (or motion primitive) that connects it to n_1. The full list of information stored in each node is as follows:

- Position (in Cartesian coordinates)
- Heading
- Time
- Type of preceding trajectory

The state of the aircraft at the node is defined by the position, heading, and time. The type of preceding trajectory denotes the type of primitive (trim or agile maneuver) that was used to arrive at that state from the previous node. If the preceding trajectory is a trim primitive, the type of preceding trajectory will include the yaw rate and climb/descent rate. In the case of an agile maneuver primitive, the type simply defines which of the agile maneuver primitives it is.

6.2 Extend Tree

The *Extend Tree* function aims to add a new node to the tree. In general, a random node is generated and steered towards (using a motion primitive) from the nearest node in the tree. This method biases the search into

the largest Voronoi regions, i.e. the unexplored areas (LaValle, 1998); in our case, in the three-dimensional Cartesian space, $\mathcal{C} = \mathbb{R}^3$. This concept is known as the *Voronoi bias* and is the key aspect of the RRT algorithm. We add a slight goal node bias, sampling the end point instead of a random one every 40 iterations. This balances exploration with movement towards the goal region. If the primitive extended from the nearest node ends up colliding with an obstacle, a new attempt is made with the next nearest node, and so forth for five iterations. These few iterations help build through narrow corridors and around walls (Frazzoli et al, 2002). The chosen value of five was arrived at via manual tuning. A value too high results in the aforementioned benefit of the endeavor being lost, while too high a value needlessly slows down the algorithm while searching for connections in hopeless dead ends.

The *Extend Tree* function terminates in any of the following cases: a collision-free primitive is found, the list of tree nodes has been exhausted, or the maximum number of iterations through the list has been reached. Upon completion, the function outputs the node that is being extended away from, the primitive used for steering, and, in the case of a trim primitive, the time to remain along it.

The planning algorithm makes use of the agile maneuver primitives in specific ways. The plan is designed to begin from a hover, and thus the first primitive generated is always a hover-to-cruise maneuver. The cruise-to-hover maneuver is attempted every time it would land the aircraft in the goal region. The algorithm, therefore, always terminates with this maneuver, and thus with the aircraft in a hover. The aggressive turn-around maneuver is generated if a trim primitive extended from the nearest node results in a collision. This signals that the tree is headed towards an obstacle, and the aggressive turn-around maneuver can be used to immediately steer away from it, in a minimal amount of space. The maneuver is connected to the nearest node to which the random sample failed to connect. Note that the maneuver is only attempted after the first of the five iterations mentioned above. Although sequential turn-around maneuvers would be unlikely to occur anyway – because the end of the maneuver points the aircraft back into previously charted, obstacle-free territory – a simple amendment to the algorithm eliminates the possibility of this occurring. The basics of the *Extend Tree* logic are described in Algorithm 2.

Algorithm 2: ExtendTree

```
input: p_rand
List nodes in order of nearness to p_rand
if maximum number of iterations have not been
 reached then
    foreach node, η, in the list do
        if a cruise-to-hover maneuver would land the
         aircraft in the goal region then
            Generate maneuver primitive ρ_prim from
             η via SteerAgile (CTH)
        else
            Generate ρ_prim from η via Steer (p_rand)
            if η is the first node in the list and ρ_prim
             results in a collision then
                Generate ρ_prim from η via
                 SteerAgile (ATA)
            end
        end
        if ρ_prim is collision-free then
            break
        end
    end
end
```

6.3 Steer

There are two steer functions, one for trim primitives, *Steer*, and the other for agile maneuver primitives, *Steer-Agile*. These functions determine the connections of new nodes to the tree. The primary goal of the steer functions designed here is to efficiently expand the tree. They waste no time sampling primitives, nor attempting to solve for an optimal connection (e.g. the shortest path between two nodes). The primary steer function, that for trim primitives, analytically determines which single trim primitive to use for the connection, and for how long to coast along it. The function solves for yaw rate, $\dot{\psi}$, climb/descent rate, $\dot{z}$, and coasting time, Δt (as will be described in Eq. 3). This approach to steering highlights a salient feature of our methodology, which is that the size of the motion primitive library can be increased indefinitely without having any effect on the time spent creating connections. We found that in practice, for our purposes, nothing apparent is lost by failing to make more accurate connections, nor neglecting to consider a larger subset of available connections (i.e. piecing together multiple primitives to connect nodes). Results demonstrating this observation are provided in Section 7.1.

The steer function for trim primitives searches in the neighborhood of circular arc parameters. It takes as input the configuration of the node it is steering away from, $\mathbf{p}_1(x_1, y_1, z_1)$ and ψ_1, and the point it is steering towards, $\mathbf{p}_2(x_2, y_2, z_2) = \mathbf{p}_{rand}$. Determining which trim primitive to use and for how long to coast along it is solved for analytically. This information is derived from the geometry of the circular arc connecting the two node points, as seen in Fig. 9. Recalling that the transition maneuvers are generated in the planning phase, the point $\mathbf{p}_1$ is not in fact the node being steered

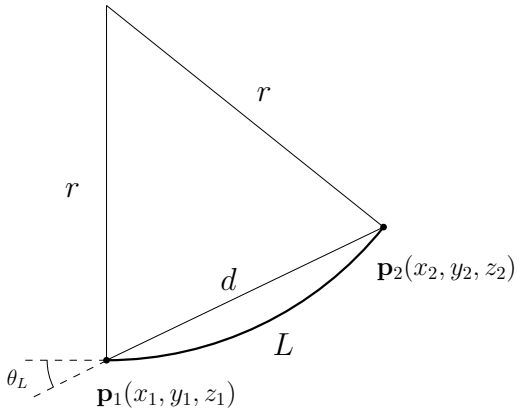


Fig. 9 Top-down view of three-dimensional circular arc defining trim primitive geometry

away from, but the node that automatically proceeds it by way of the time-delay. Referring back to Fig. 4, $\mathbf{p}_1$ would correspond with B' and $\mathbf{p}_2$ with C. The equations relating to Fig. 9 solve for the trim primitive (yaw rate, $\dot{\psi}$, and climb/descent rate, $\dot{z}$) and coasting time, Δt, that bring the aircraft as close as possible to $\mathbf{p}_2$:

$$\begin{aligned} d &= ||\mathbf{p}_2 - \mathbf{p}_1|| \\ \theta_L &= \arctan(\frac{y_2 - y_1}{x_2 - x_1}) - \psi_1 \\ r_{x,y} &= \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{2 \sin \theta_L} \\ L &= \frac{d\theta_L}{\sin \theta_L} \\ \dot{\psi} &= \frac{V}{r_{x,y}} \\ \Delta t &= \frac{L}{V} \\ \dot{z} &= \frac{z_2 - z_1}{\Delta t}, \end{aligned} \tag{3}$$

where θ_L measures the difference between the vector d and the heading, ψ_1, of the node at $\mathbf{p}_1$. The term $r_{x,y}$ is the projection of r on the horizontal plane. The desired constant speed, $V = 7\mathrm{m\,s^{-1}}$, appears in these equations to calculate the coasting time. Given that there are a finite number of trim primitives in the trajectory library, the yaw rate and climb/descent rate calculated in Eq. 3 are each approximated to the closest available rates. In addition to the end node of the trim primitive, intermediate nodes are also returned by this function. This is useful to the planner in that it generates more tree node options to be steered away from in the next *Extend Tree* phase.

The steer function that handles agile maneuver primitives requires as input only the the type of agile maneuver (of the three) and the node to steer away from. The function uses this information to output the end node of the maneuver, the data of which is pre-computed. No intermediate nodes are returned from this steer function because the planner and controller are not designed to exit agile maneuvers partway through.

6.4 Collision Check

The collision checking function detects if a primitive is outside the bounds of the environment or overlapping an obstacle. The function performs checks in intervals along the primitive and discards the primitive as a whole if any segment has a collision. The displacements of each of the three agile maneuvers are nearly restricted to the vertical plane, as can be seen in Fig. 3, and are pre-computed. Therefore, the collision check on agile maneuvers is trivial; it looks for any collision along a path with the same forward and vertical displacements.

As implemented here, the function detects collisions with obstacles that are rectangular prisms, based on geometric constraints. It could presumably be replaced with a function that can handle more complex obstacle geometries, so long as they can be approximated by polyhedrons. For an alternative approach based on circular or cylindrical obstacles and circular trajectories, see Paranjape et al (2015).

To account for the aircraft's geometry (i.e. that it is not a point mass) and non-ideal tracking performance from the controller, a buffer distance is added to all obstacles and environment boundaries. In effect, obstacles are inflated in the collision checker so that the aircraft stays a safe distance away from them.

6.5 Update Tree

The update function allows the planner to run in real-time, and is the point of communication between the planner and the control system. Its jobs are to choose which nodes of the tree the aircraft should follow until the next time the function is called, and to update the tree of nodes to account for the aircraft's real-time motion. The function is called iteratively as the aircraft moves up the tree.

The update function first determines the optimal node (of those available at the time) to guide the aircraft towards. It checks all nodes to find out which is 'nearest' (as will be defined) to the goal. It then determines how far along the tree to move in the direction of that node for the current iteration, i.e. how many nodes to commit to for one time interval given the aircraft's dynamics. The aircraft commits to following

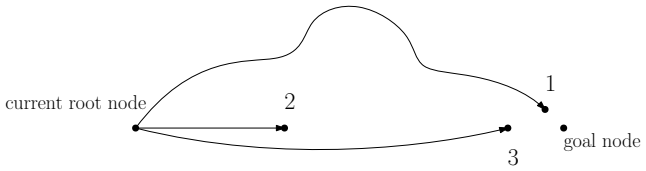


Fig. 10 Nearness quantity. Node number 3 is 'nearest' to the goal node when compared to nodes 1 and 2

these nodes, and they are sent to the control system to be tracked. In the unlikely scenario that the optimal node is reached in this step but is not in the goal region, the planner continues to run while the aircraft moves towards this node and then along its children. It is possible that within this time new nodes that lead to the goal (or might eventually) are added to the tree. If the aircraft happens to catch up to the optimal node, is not within the goal region, and has no children to follow, the algorithm ends by sending a cruise-to-hover primitive to the controller. From here, the user has the option to re-initiate the motion planner or recover the aircraft.

The 'nearness' quantity is calculated based on the length of the path to the node, the straight-line distance between the node and the goal, and the distance between the current root and the node, see Equation 4. In the hypothetical situation depicted in Fig. 10, node number 3 would be 'nearest' to the goal of the three options. It is not as close to the goal as node 1, nor is the path to it from the root node as short as node 2, but the balance of these quantities (evaluated by Equation 4) makes it 'nearest' the goal.

$$nearness = \frac{\begin{array}{c}length\ of\ path \\ to\ node\end{array} + \begin{array}{c}distance\ from \\ node\ to\ goal\end{array}}{distance\ from\ current\ root\ to\ node} \tag{4}$$

To account for the aircraft's real-time motion, the update function also prunes the tree of the nodes that become infeasible as a result of the commitment; nodes that will be 'behind' the aircraft in time (and their children) as it moves 'up' the tree. The *Update Tree* function is presented in Algorithm 3.

Algorithm 3: UpdateTree

```
input: η_root, p_goal
Find tree node, η_opt, nearest p_goal
Get list of nodes connecting η_root to η_opt
for η_temp ← η_root to η_opt do
    if η_temp is in goal region or η_temp exceeds time
     interval then
        break
    end
    if η_temp is η_opt then
        if η_opt has children then
            Continue along children of η_opt
        else
            Send cruise-to-hover node to controller
        end
    else
        Send η_temp to controller
    end
end
Prune tree of infeasible nodes
```

7 Simulations

In this section, two types of simulations are run. First, the motion planner itself is simulated over various obstacle-dense maps. To help evaluate its performance, the planning algorithm is compared to a baseline approach. The other form of simulation uses the aircraft dynamics model and feedback controller to track trajectory solutions solved for by the motion planner.

7.1 Motion Planning Simulations

Simulations were run to validate the motion planner and contrast it against a baseline approach. The approach employed for this purpose is RRT with Dubins curves, which is a commonly used technique for path-planning with ground vehicles (Takei et al, 2010; Karaman et al, 2011) and fixed-wing UAVs (Lugo-Cárdenas et al, 2014; Owen et al, 2014; Allen and Pavone, 2015; Karaman and Frazzoli, 2011a). Dubins curves are minimum-distance paths between two points with prescribed headings, for a vehicle that is subject to the constraints of the Dubins kinematic model (Dubins, 1957):

$$\begin{aligned} \dot{x} &= V\cos\psi \\ \dot{y} &= V\cos\psi \\ \dot{\psi} &= u, \end{aligned} \tag{5}$$

where (x, y) is the position of the vehicle, V is a constant speed, and ψ is the heading. By incorporating an additional configuration variable for altitude, the Dubins model has been extended to 3D problems (Chitsaz and LaValle, 2007), but for simplicity we will use the 2D model here for comparison. From these equations, we see the most apparent difference between the two approaches, which is that Dubins curves are the product of a very simple kinematic model evolving on the configuration space $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. Using this model, $\mathbf{p}_1$ and $\mathbf{p}_2$ of Fig. 9 (in 2D and with prescribed headings) are connected using the shortest feasible path. The solution to

this problem, given the constraints of Eq. 5, is proven to always consist of minimum-radius circular arcs and/or straight line segments (Dubins, 1957). Therefore, using the Dubins curve approach, every time the trajectory changes heading, it does so using the most aggressive turn.

To employ the Dubins curve approach, we replaced our steer functions with one that solves for the optimal Dubins path. Both motion planning frameworks were run on a 100 m by 100 m map that includes 50 randomly generated obstacles. Representative samples of trajectory solutions are shown in Fig. 11. To compare the two approaches, Maps A and C use the same layout, as do Maps B and D. In the figures, the axes stretch the length of the environment, the black objects represent obstacles, and the orange spheres are the desired goal regions. The trajectories flown are colored blue, except for the agile maneuvers, where magenta is the hover-to-cruise maneuver, green is the cruise-to-hover maneuver, and the aggressive turn-arounds are colored red. The remaining parts of the tree, which were not flown, are colored black. Note that we had to incorporate a part of our maneuver space, the hover transitions, into the Dubins curve approach just to be able to solve the desired planning problem, which includes stationary initial and final states.

The few cases plotted in Fig. 11 highlight features of the proposed approach. Because the maneuver space includes many more primitives than there are Dubins curves, the turns of the trajectories tend to be smoother and less aggressive. Although the Dubins curves do solve for the shortest paths connecting individual nodes, this optimality tends to be lost in terms of the full trajectory solutions, as can be seen in figures 11(c) and 11(d).

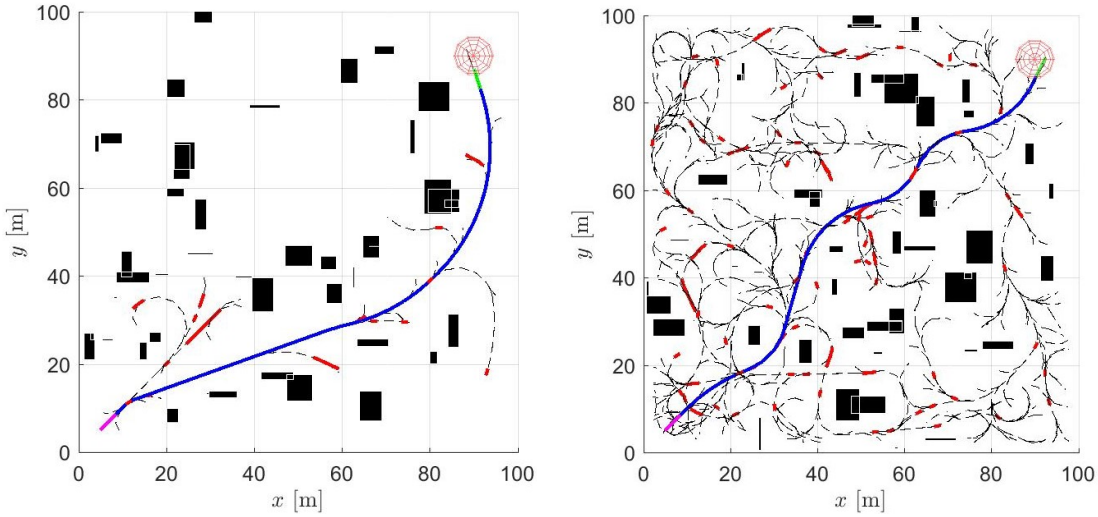
In a different map, we investigate how the two approaches would fare in a situation that necessitated a near 180-degree turn-around; the results are plotted in Fig. 12. While Dubins curves can indeed be used to generate a feasible path through this map, there are disadvantages to the approach relative to ours. The planner using Dubins curves generally takes more time to solve such a problem (the greater number of black paths in Fig. 12(b) signifies the longer time it took to find a feasible solution). This is because many positive collision checks have to occur before the Dubins curves can navigate a path out of the dead end. Using the proposed maneuver space, however, one of the first positive collision checks results in the generation of the functionally-designed aggressive turn-around maneuver, which immediately provides the beginning of a way out. Fig. 12 also illustrates how our planner tends to generate smoother trajectories through narrow corridors, where the Dubins approach bounces around between the minimum-radius curves. Though the Dubins model is restricted to two dimensions, we demonstrate the applicability of our planner for a 3D environment in Figure 13.

We evaluated the performance of each algorithm, in terms of computational efficiency and cost (length of the path solution). To do so, we programmed them both on the test platform's computer (an ODROID XU4 – see Section 8.1). Each algorithm was run 1000 times over the map of Fig. 11, and again over the map of Fig. 12. Only the former map uses randomly generated obstacles, but in either case the planner itself is random in its sampling of the environment, making each solution unique. For the map of Fig. 11, the average time to find a feasible trajectory using the maneuver space was 150 milliseconds, compared to 96 milliseconds using Dubins curves. The average path length was 180 m using the maneuver space, and 182 m using Dubins curves. These results reinforce the point that the optimality of individual Dubins paths does not carry over to the full solution. For the map of Fig. 12, in which the aircraft had to retreat from a dead end, the average computation time and path length using the maneuver space was 20 milliseconds and 71 m, respectively. With Dubins curves, the algorithm took slightly longer, 26 milliseconds, and averaged a significantly longer path of 102 m.

A final matter of differentiation between the two approaches is how well they lend themselves to the trajectory tracking problem. In this respect, there are a few things to note about the Dubins curves approach. There are no transition maneuvers between curves, and the kinematic model assumes accelerations can be controlled directly. Being restricted by the aircraft's dynamics, no such arbitrary accelerations can in fact be generated. Also recall that the only turns available are the minimum-radius turns. A conservatively chosen turn-rate constraint will limit the abilities of the planner to navigate around obstacles, while a high turn-rate will require the aircraft to track a more aggressive trajectory. Minimum-radius turns without transitions make the tracking problem demanding, and on top of this, the Dubins model offers no feedforward control input solutions for the aircraft's actuators.

7.2 Trajectory Tracking

Prior to conducting flight tests, the aircraft dynamics model was used to simulate motion plan tracking. Relative to the actual testing area, the simulation environment allows for flight through larger maps. The simulation architecture is illustrated in Fig. 14 as a block



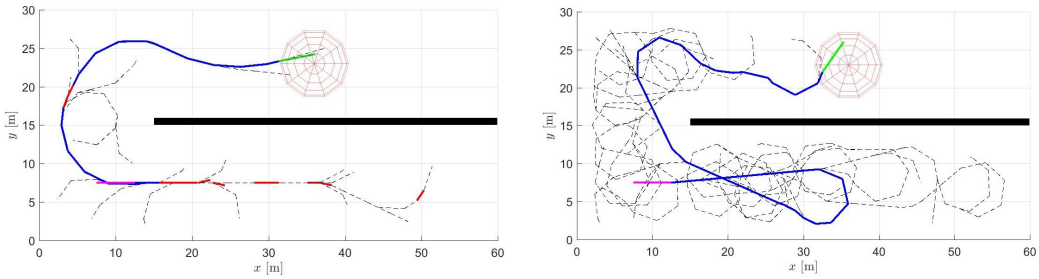
(a) Map A: using maneuver space

(b) Map B: using maneuver space

(c) Map C: using Dubins curves

(d) Map D: using Dubins curves

Fig. 11 Motion plans through a 100 m by 100 m map with 50 randomly generated obstacles. Maps A and B use the proposed maneuver space approach; Maps C and D use Dubins curves



(a) Map E: using maneuver space

(b) Map F: using Dubins curves

Fig. 12 Motion plans involving a retreat from a narrow corridor with a dead end. Map E uses the proposed maneuver space approach; Map F uses Dubins curves

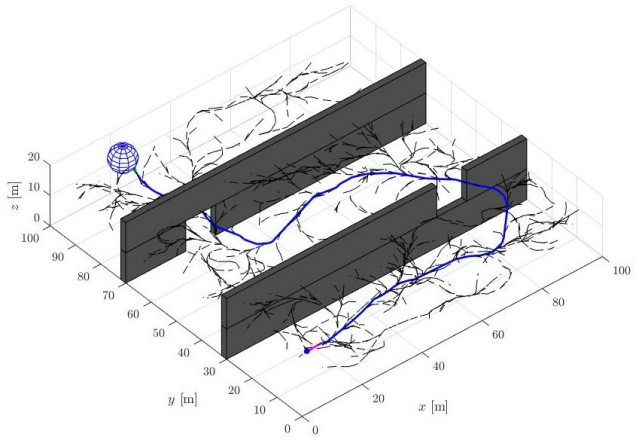


Fig. 13 3D motion planning with maneuver space through environment with narrow gaps

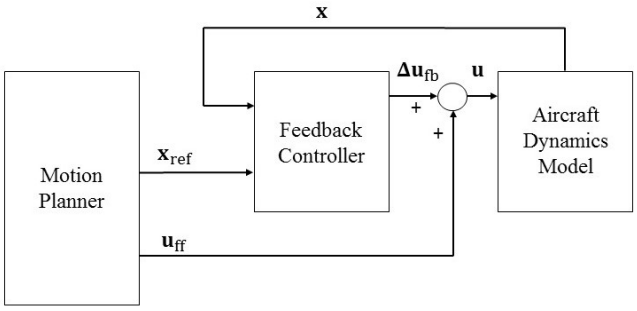


Fig. 14 Block diagram of simulation environment

diagram. The motion planner outputs reference trajectories, x_{ref}, and feedforward control inputs, u_{ff}. The output of the feedback controller, Δu_{fb}, is summed with the feedforward part to produce the full control input, u.

The Dubins approach solves only for the reference path and heading, since it is based on the simple kinematic model of Eq. 5. Of particular note is that the Dubins model provides no means of calculating the feedforward control inputs. Accordingly, these terms were set to zero when tracking the Dubins curves (we made the exception to include feedforward inputs during hover transitions). For good measure, we ran the set of simulations associated with the Dubins approach an extra time, including the feedforward inputs generated using the high-fidelity model. In effect, we thereby treat the Dubins paths as a small subset of our trim primitives: straight and level flight, a sharp banked turn to the right, and a sharp banked turn to the left.

We simulated all of Maps A-F in Figures 11 and 12. As an example, the path tracking performance for the trajectory shown in 11(b) is plotted in Fig. 15. The RMSE and maximum error on position for each simulation are listed in Table 2. Maps A, B, and E use the maneuver space, while Maps C, D, and F use Dubins curves – note the large discrepancies in tracking performance between the two approaches. In the case of Map D, the aircraft essentially failed to track the trajectory; the position errors grew so large we opted not to list them. Also shown in this table are the results of tracking the Dubins curves whilst incorporating the feedforward inputs generated using the high-fidelity model. Even after integrating this aspect of our motion primitives into the Dubins approach, the tracking performance remained inferior. With respect to the RMSE values, there was still a difference in position error by a factor of approximately five between the two methods. These results can be attributed to the lack of transition modeling between the straight segments and highly aggressive turns.

With respect to the maneuver space approach, the position errors, along with the 0.86 m wingspan of the aircraft, can be used to inform the buffer distance parameter found in the collision checker. Given these values, it would be reasonable to set the buffer distance to at least 1.5 m to ensure safe, collision-free flight.

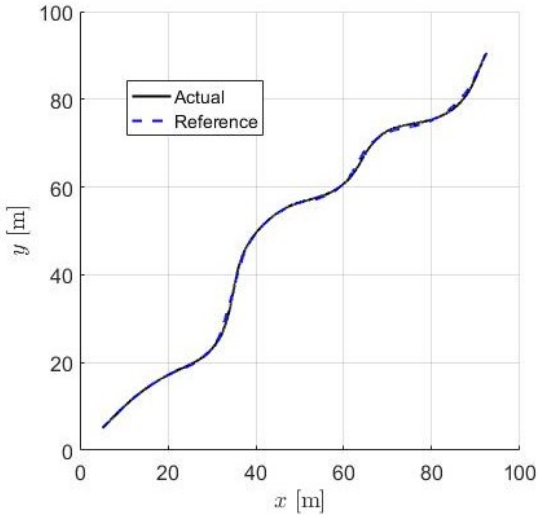


Fig. 15 Simulated path tracking

Table 2 Root-mean-square errors and maximum errors for position tracking in simulations. Feedforward control inputs denoted by 'FF'.

Map	RMSE [m]	max(e_p)[m]
A (Maneuver Space)	0.22	0.86
B (Maneuver Space)	0.27	0.62
E (Maneuver Space)	0.38	0.71
C (Dubins)	5.04	12.37
D (Dubins)	–	–
F (Dubins)	3.81	13.68
C (Dubins + FF)	1.21	2.76
D (Dubins + FF)	1.58	2.91
F (Dubins + FF)	2.13	3.79

The tracking performance and buffer size must be interpreted with respect to the environment the aircraft is tasked with passing through. As long as there is sufficient room left after the obstacles are buffered for the tree to efficiently expand through the map, as has been the case here, the 1.5 m distance is acceptable.

8 Flight Test Experiments

Flight tests with the small agile fixed-wing UAV were conducted in the *Concordia Stinger Dome*. The dome is made of fabric that is GPS-transparent. The flights were mainly limited to one quarter of the dome, a 30 m by 60 m field.

8.1 Experimental Setup

The control system is programmed on the *Pixhawk Mini* flight controller as a module in the PX4 open-source flight stack firmware. Inside the *Pixhawk Mini* is a sensor suite comprised of a barometric pressure sensor and two motion tracking devices with gyroscopes, accelerometers, and a compass. An external GPS is mounted on the aircraft's nose. On-board state estimation is performed using *Pixhawk*'s default Extended Kalman Filter.

The motion planning algorithm is programmed in C++ on an ODROID XU4. The ODROID is a single-board computer with a 2GHz quad-core processor and 2GB of RAM. The board runs Ubuntu on a Linux kernel. The ODROID is connected to the Pixhawk via an FTDI USB to UART cable. Communication goes both ways and uses the MAVLink protocol. When the motion planner is triggered to begin, the Pixhawk sends the aircraft's initial pose to the ODROID. As the motion planning algorithm runs, the ODROID sends individual nodes to the Pixhawk (in the update function), which gets read within the control system module. The tree data stored in each node is interpreted by the Pixhawk's control system as a reference trajectory for a specific time interval.

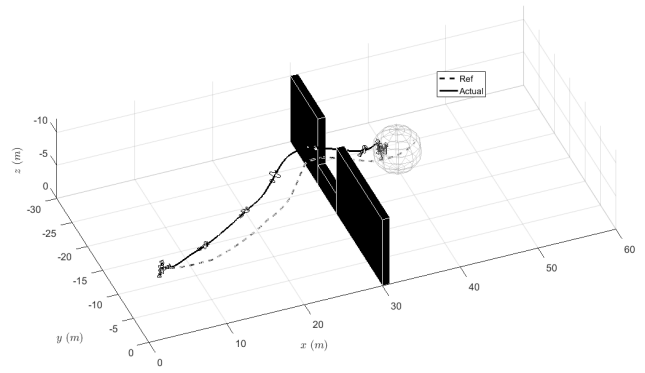
The experimental procedure is as follows. Using a *Futaba T7C* RC transmitter, a trained pilot manually takes off and flies the aircraft into a hover. The pilot then flicks a switch on the transmitter to put the aircraft into an autonomously controlled hover (the hover trim primitive). Next, he flicks another switch to trigger the motion planner to begin. It is at this time that the *Pixhawk Mini* sends the aircraft's current measured configuration (position and heading) to the motion planner as the initial condition to the algorithm. As a practical measure, the planner was programmed to delay the first iteration of the *Update Tree* function until the tree reached the goal region. While not strictly necessary, this ensured a feasible path to the goal region existed before taking off from the hover. The delay never lasted more than a few seconds; and often less than one. Note that the planner continues to run while in flight, and as per the *Update Tree* function, may end up on a more direct path towards the goal.

8.2 Results and Discussion

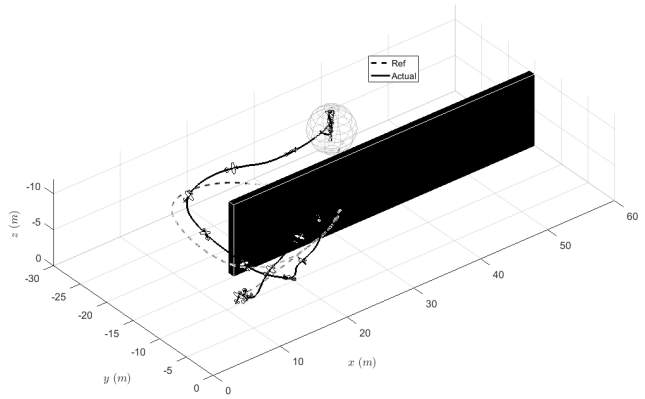
Due to logistical constraints, the actual environment is obstacle-free. To no different effect than having actual obstacles, three different maps of virtual obstacles were programmed onto the ODROID. The available sections of the dome are relatively small and thus the flights are short; nonetheless, they showcase many features of the motion planner.

Figure 16 shows trajectories and path tracking results for three new maps. The maps are different than those of Section 7 because the space in the dome is limited to a smaller area. Again, in Fig. 16, the plot edges match the maps' boundaries. The plots show the reference paths and the actual paths flown, and the grey spheres are the goal regions. In the first map, the aircraft must climb to and navigate through a narrow gap. In the second map, the aircraft begins with a heading that is pointed towards a dead end, and thus must turn around to proceed in the proper direction. It executes an aggressive turn-around maneuver here to do so. The third map uses a larger portion of the dome, a 60 m by 60 m field, and the virtual obstacles overlay actual obstacles in the dome – wires and meshing that separate quarters of the field. A supplementary video (Online Resource 1) includes the flight test results for Maps 1 and 2, and one other flight. Objects are edited into the video to give the effect of actual obstacles being present.

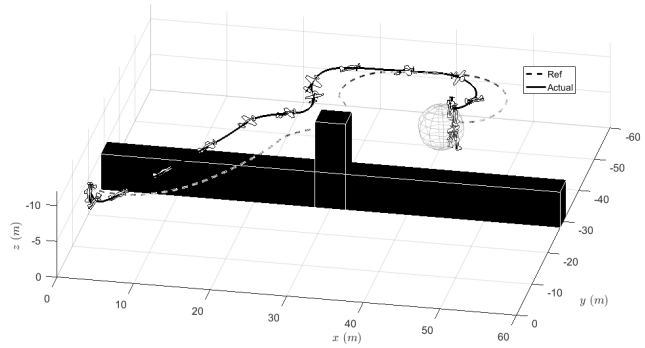
Figure 17 shows time histories of the state variables and control inputs for the flight test associated with Map 3 in Fig. 16(c). We take a closer look at the results of this map because it has the longest trajectory of the three, and the discussion of its results largely extends to the findings of the other maps. The position errors along each axis and in total, e_p, are shown in Fig. 17(a). Note that the x, y, and z axes are aligned with the map, as in Fig. 16(c). We see that around 2 s the position errors start to grow as the hover-to-cruise maneuver is occurring. Thereafter, the errors more or less plateau and only diminish around when the cruise-to-hover maneuver takes place. The reason the errors plateau instead of diminish is attributed to the fact that the aircraft is continually being destabilized by switching motion primitives. As demonstrated in Section 4.3,



(a) Map 1



(b) Map 2

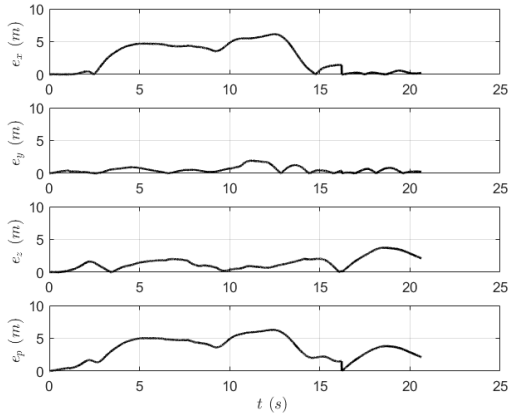


(c) Map 3

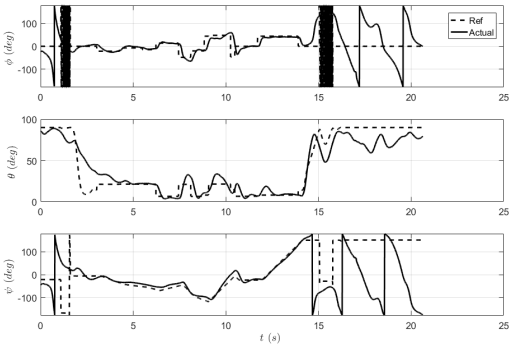
Fig. 16 Flight test results. Commanded paths are drawn as dashed lines, and actual paths flown as solid lines. Grey spheres represent the goal regions and obstacles are drawn in black

the transition maneuver heuristic helps deal with this tracking problem, however, it does not eliminate it. The change from one primitive to the next can most clearly be seen in Fig. 17(b), where every step input change in the reference roll angle implies that a new primitive is being commanded.

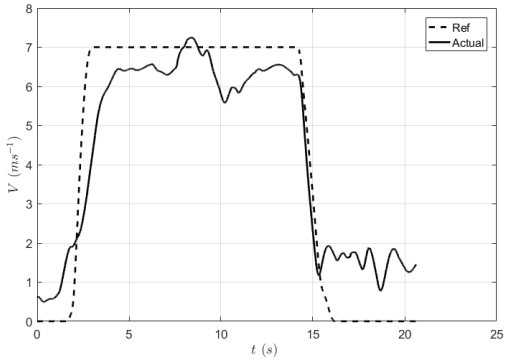
Figure 17(b) plots the attitude history of Map 3's trajectory as Euler angles. Note that the aircraft is in a hover at the beginning and end of the plan. This causes a singularity in the Euler angle attitude representation,



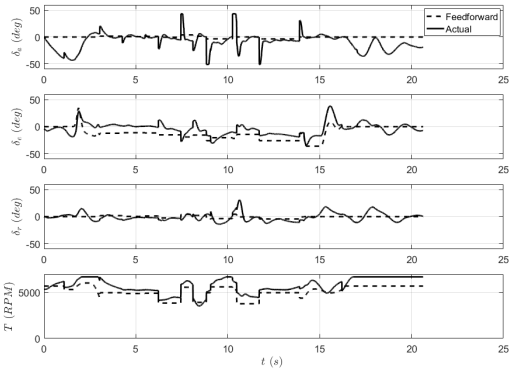
(a) Position errors



(b) Attitude tracking



(c) Speed tracking



(d) Control inputs

Fig. 17 Flight test states and control inputs for Map 3 of Fig. 16(c)

Table 3 Root-mean-square errors and maximum errors for position tracking in flight tests.

	Map 1	**Map 2**	**Map 3**
RMSE [m]	2.82	3.03	3.88
max(e_p)[m]	4.49	4.56	6.30

which is why the roll and yaw values are spiking back and forth. The reference yaw angle time history is continuous, but the roll and pitch values change in steps because they are associated with the primitives. This highlights the importance of the transition maneuvers, which allow the aircraft time to reach (and ideally settle) at the state of the new primitive. The attitude tracking is at its worst at the beginning of the plan. At approximately two seconds in, the hover-to-cruise maneuver is initiated. We note that the pitch profile is not tracked as accurately during this maneuver as it is for the remainder of the plan.

In Fig. 17(c), the speed throughout the plan is plotted. Throughout the middle portion of the plan, the aircraft is able to stay near the desired constant speed of $V = 7\mathrm{m\,s^{-1}}$. The most challenging sections for the speed tracking portion of the control system are the hovers. The aircraft is inherently unstable in this configuration, and often has to use non-zero velocities to maintain a commanded position.

The feedforward and full control inputs are plotted in Fig. 17(d). The difference between the feedforward and actual inputs is the feedback control. By comparing the solid and dashed lines, we can see from these plots that both feedforward and feedback inputs are valuable. The elevator, rudder, and thrust control are largely guided by the feedforward inputs, i.e. the actual control is close to the feedforward control. The ailerons, however, are using a large amount of feedback control. The aileron response is proportional to the large step input changes in roll that are being commanded.

The values of RMSE and max error in position for all three maps are given in Table 3. The position errors are larger than those found in the simulations of Section 7.2, in part because of the presence of measurement noise, imperfect state estimation, and modeling inaccuracies in the experimental setup. We also note the availability of the dome limited the time that could be spent tuning controller gains and the time-delay constant, all of which can have a significant effect on the feedback controller's performance. The errors could presumably be resolved in future work. In addition to more extensive tuning, one option would be to implement gain scheduling, so that the optimal gains for the hover-to-cruise maneuver could be used, and thus the initial errors could be reduced. An investigation of alternative methods for nonlinear control, such as time-varying linear quadratic regulators (TVLQRs) Barry (2012) would also be valuable. Furthermore, position errors could be addressed on the side of the motion planner: a re-planning algorithm could be implemented for when position tracking errors become sufficiently large. The framework for such an algorithm is described in Appendix A.

As a general remark about the flight control, we believe that the weight and weight distribution of the fully equipped aircraft is contributing to tracking deficiencies. The layout of the fixed-wing UAV makes it impossible to place all heavy equipment at its center of gravity, and thus the weight distribution is less than ideal. We noticed a decline in tracking performance when the ODROID was added to the aircraft. If it were possible for only one computer to handle the autopilot and planning systems, it could be placed near the aircraft's center of gravity.

9 Conclusions

In this work, a real-time motion planner for a small agile fixed-wing UAV was implemented for flight through highly constrained three-dimensional environments. We found the method of solving optimal control problems to be an effective way of generating dynamically feasible motion primitives that take advantage of the aircraft's physical capabilities. The trajectories were assembled into a maneuver space and thereby incorporated into the planning algorithm. The time-delay approach to switching between primitives helps smooth the transitions without compromising the efficiency of the algorithm or using any extra on-board resources. The RRT-based planning algorithm, tested on multiple maps, was able to consistently and rapidly find a plan to the goal region, and make effective use of three agile maneuvers.

The algorithm and control system were validated in simulations and flight tests. In simulations, the planner was evaluated against a baseline approach that uses Dubins curves. Using the proposed planning approach, the root-mean-square errors on position were approximately 0.3 m; using Dubins curves, the errors were more than an order of magnitude greater. In flight tests, the RMSE values were higher, approximately 3 - 4 m. The errors appeared to arise as a result of the hover-to-cruise maneuver, and then persist throughout the flight. Potential solutions to this problem were suggested for future work, including more extensive gain tuning, gain scheduling, and the addition of a re-planning phase in the motion planning algorithm.

We note that the algorithm was not designed with any specific intention of dealing with wind gusts, which are a serious consideration for outdoor flight and should also be the subject of future work. The aforementioned re-planning step in the algorithm would presumably be able to help in this respect.

Acknowledgements This research was supported by the Natural Sciences and Engineering Research Council of Canada (grant no. PGSD3-490220-2016) and by le Fonds de Recherche du Quebec - Nature et Technologies (grant no. 2016-PR-191001).

A Re-planning

This appendix outlines how a re-planning step can be incorporated into the motion planner to eliminate cumulative position errors if and when they grow sufficiently large. The actions of re-planning would neatly fit at the end of Algorithm 3, under a conditional statement that gets added to check for position error against a user-defined constant: **if** $e_p > \epsilon$.

Re-planning involves two actions, the first of which is to modify the nodes being sent to the controller during the update, such that they align with the actual position and heading of the aircraft. The motion primitives themselves remain the same, i.e. $\dot{\psi}$, $\dot{z}$, and Δt are known, but the positions and headings of each node must be recalculated. This is done through rearrangement of Eq. 3:

$$\begin{aligned} x_2 &= x_1 + \left(\frac{V}{\dot{\psi}} \sin(\psi_1 + \dot{\psi}\Delta t) - \frac{V}{\dot{\psi}} \sin\psi_1 \cos(\arcsin\frac{\dot{z}}{V}) \right) \\ y_2 &= y_1 + \left(-\frac{V}{\dot{\psi}} \cos(\psi_1 + \dot{\psi}\Delta t) + \frac{V}{\dot{\psi}} \cos\psi_1 \cos(\arcsin\frac{\dot{z}}{V}) \right) \\ z_2 &= z_1 + \dot{z}\Delta t \\ \psi_2 &= \psi_1 + \dot{\psi}\Delta t \end{aligned} \tag{6}$$

The other action taken during re-planning is to prune the tree of all nodes other than the ones being sent to the aircraft during the update. While it may seem detrimental to throw away these previously generated nodes, the algorithm is very efficient at building (or re-building) a tree in real-time. The alternative, to keep the tree nodes, would require translating each node and re-checking each primitive for collisions. This is a much more costly process (namely, the collision checking), that would not be of any great benefit given how quickly the tree can be re-built.

Tests were run on the ODROID XU4 to evaluate the practicality of employing the re-planning step. We investigated whether the planner was efficient enough to recover from pruning almost all of the tree nodes in real-time. Using the map of Fig. 13, the motion planning algorithm was run twenty times. The planner was left to run until the Update Tree function had commanded a path that ended in the goal region; as though the aircraft were in flight and the algorithm were running in real-time. Instead of setting up the ODROID in a simulation loop with the aircraft dynamics model, we simply programmed fake position errors into the algorithm such that the re-planning step would be repeatedly triggered; for each run, the re-planning step was triggered twice. Each time the re-planning step occurred, the algorithm was able to rapidly rebuild a new tree. By the next time the Update Tree function was called after re-planning (it is called every half second), the tree would have already grown to hold approximately 1000 new nodes, on average. For reference, the tree rarely ever grew to have many more than 2000 nodes at a time, for the map in question. In the twenty runs, nineteen successfully resulted in a path to the goal region. In the one other case, the planner got stuck and had to send a command to perform the cruise-to-hover maneuver before reaching the goal region. It cannot necessarily be determined that the re-planning step was the cause of this failure. We observed that the number of tree nodes in subsequent calls of the Update Tree function, before and after re-planning, was barely affected, and thus we can at least rule out the notion of the failure being caused by a lack of trajectory options post-tree-pruning.

References

Allen R, Pavone M (2015) *Toward a Real-Time Framework for Solving the Kinodynamic Motion Planning Problem.* In: 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 928–934, DOI 10.1109/ICRA.2015.7139288

Ananthkrishnan N, Sinha NK (2001) Level flight trim and stability analysis using extended bifurcation and continuation procedure. *Journal of Guidance, Control, and Dynamics* 24(6):1225–1228, DOI 10.2514/2.4839

Barry AJ (2012) Flying between obstacles with an autonomous knife-edge maneuver. Master's thesis, Massachusetts Institute of Technology

Bulka E, Nahon M (2018) Automatic control for aerobatic maneuvering of agile fixed-wing UAVs. *Journal of Intelligent & Robotic Systems* DOI 10.1007/s10846-018-0790-z

Chitsaz H, LaValle SM (2007) *Time-Optimal Paths for a Dubins Airplane.* In: 2007 46th IEEE Conference on Decision and Control, IEEE, pp 2379–2384, DOI 10.1109/CDC.2007.4434966

Dubins LE (1957) 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, DOI 10.2307/2372560

Frazzoli E, Dahleh MA, Feron E (2002) Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics* 25(1):116–129, DOI 10.2514/2.4856

Frazzoli E, Dahleh MA, Feron E (2005) Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics* 21(6):1077–1091, DOI 10.1109/TRO.2005.852260

Gavrilets V, Frazzoli E, Mettler B, Piedmonte M, Feron E (2001) Aggressive maneuvering of small autonomous helicopters: A human-centered approach. *The International Journal of Robotics Research* 20(10):795–807, DOI 10.1177/02783640122068100

He Z, Li D, Lu Y (2018) Disturbance compensation-Based piecewise linear control design for perching maneuvers. *IEEE Transactions on Aerospace and Electronic Systems* pp 1–16, DOI 10.1109/TAES.2018.2849898

Karaman S, Frazzoli E (2011a) *Optimal Kinodynamic Motion Planning Using Incremental Sampling-Based Methods.* In: 49th IEEE Conference on Decision and Control (CDC), IEEE, pp 7681–7687, DOI 10.1109/CDC.2010.5717430

Karaman S, Frazzoli E (2011b) Sampling-based algorithms for optimal motion planning. *The Interna-*

tional Journal of Robotics Research 30(7):846894, DOI 10.1177/0278364911406761

Karaman S, Walter MR, Perez A, Frazzoli E, Teller S (2011) *Anytime Motion Planning Using the RRT**. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), IEEE

Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Transactions on Robotics and Automation* 12(4):566–580, DOI 10.1109/70.508439

Khan W (2016) Dynamics modeling of agile fixed-wing unmanned aerial vehicles. PhD thesis, McGill University

Khan W, Nahon M (2013) Toward an accurate physics-based UAV thruster model. *IEEE/ASME Transactions on Mechatronics* 18(4):1269–1279, DOI 10.1109/tmech.2013.2264105

Khan W, Nahon M (2015a) Development and validation of a propeller slipstream model for unmanned aerial vehicles. *Journal of Aircraft* 52(6):1985–1994, DOI 10.2514/1.C033118

Khan W, Nahon M (2015b) *Real-time Modeling of Agile Fixed-Wing UAV Aerodynamics*. In: 2015 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, pp 1188–1195, DOI 10.1109/icuas.2015.7152411

Kim HJ, Shim DH, Sastry S (2002) *Nonlinear Model Predictive Tracking Control for Rotorcraft-based Unmanned Aerial Vehicles*. In: Proceedings of the American Control Conference, IEEE, pp 3576–3581, DOI 10.1109/ACC.2002.1024483

Kuo BC, Golnaraghi F (2003) *Automatic Control Systems*. John Wiley & Sons, New York

LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept, Iowa State University

Lee D, Shim DH (2014) *RRT-Based Path Planning for Fixed-Wing UAVs with Arrival Time and Approach Direction Constraints*. In: 2014 International Conference on Unmanned Aircraft Systems, IEEE, pp 317–328

Levin JM, Paranjape A, Nahon M (2017) *Agile Fixed-Wing UAV Motion Planning with Knife-Edge Maneuvers*. In: 2017 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, pp 114–123, DOI 10.1109/icuas.2017.7991475

Levin JM, Paranjape AA, Nahon M (2018a) *Motion Planning for a Small Aerobatic Fixed-Wing Unmanned Aerial Vehicle*. In: The International Conference on Intelligent Robots and Systems (IROS), IEEE, pp 8464–8470, DOI 10.1109/IROS.2018.8593670

Levin JM, Paranjape AA, Nahon M (2018b) Sideslip and slipstream in extreme maneuvering with fixed-wing unmanned aerial vehicles. *Journal of Guidance, Control, and Dynamics* 41(7):1610–1616, DOI 10.2514/1.G003086

Lugo-Cárdenas I, Flores G, Salazar S, Lozano R (2014) *Dubins Path Generation for a Fixed Wing UAV*. In: 2014 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, pp 339–346, DOI 10.1109/ICUAS.2014.6842272

MacAllister B, Butzke J, Kushleyev A, Pandey H, Likhachev M (2013) *Path Planning for Non-Circular Micro Aerial Vehicles in Constrained Environments*. In: 2013 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 3933–3940, DOI 10.1109/ICRA.2013.6631131

Majumdar A, Tedrake R (2017) Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research* 36(8):947–982, DOI 10.1177/0278364917712421

Owen M, Beard RW, McLain TW (2014) *Handbook of Unmanned Aerial Vehicles*. Springer Netherlands, DOI 10.1007/978-90-481-9707-1 120

Paranjape AA, Meier KC, Shi X, Chung SJ, Hutchinson S (2015) Motion primitives and 3-D path planning for fast flight through a forest. *The International Journal of Robotics Research* 34(3):357–377, DOI 10.1177/0278364914558017

Park S (2012) Autonomous aerobatics on commanded path. *Aerospace Science and Technology* 22(1):64–74, DOI 10.1016/j.ast.2011.06.007

Patterson MA, Rao AV (2014) GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using HP-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software* 41(1):1–37, DOI 10.1145/2558904

Schouwenaars T, How J, Feron E (2004a) *Receding Horizon Path Planning with Implicit Safety Guarantees*. In: Proceedings of the 2004 American Control Conference, IEEE, pp 5576–5581, DOI 10.23919/ACC.2004.1384742

Schouwenaars T, Mettler B, Feron E, How J (2004b) Hybrid model for trajectory planning of agile autonomous vehicles. *Journal of Aerospace Computing, Information, and Communication* 1:629–651, DOI 10.2514/1.12931

Selig MS (2014) Real-time flight simulation of highly maneuverable unmanned aerial vehicles. *Journal of Aircraft* 51(6):1705–1725, DOI 10.2514/1.c032370

Snell SA, Enns DF, Jr WLG (1992) Nonlinear inversion flight control for a supermaneuverable aircraft. *Journal of Guidance, Control, and Dynamics* 15(4):976–984, DOI 10.2514/3.20932

Sobolic FM (2009) Agile flight control techniques for a fixed-wing aircraft. Master's thesis, Massachusetts Institute of Technology

Takei R, Tsai R, Shen H, Landa Y (2010) *A Practical Path-Planning Algorithm for a Simple Car: a Hamilton-Jacobi Approach*. In: Proceedings of the 2010 American Control Conference, IEEE, pp 6175–6180, DOI 10.1109/ACC.2010.5531607

Vieira HL, Grassi V (2014) *Improving RRT's Efficiency through Motion Primitives Generation Optimization*. In: 2014 Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol, IEEE, pp 37–42, DOI 10.1109/SBR.LARS.Robocontrol.2014.20

Wickenheiser AM, Garcia E (2006) Longitudinal dynamics of a perching aircraft. *Journal of Aircraft* 43(5):1386–1392, DOI 10.2514/1.20197

Wickenheiser AM, Garcia E (2008) Optimization of perching maneuvers through vehicle morphing. *Journal of Guidance, Control, and Dynamics* 31(4):815–823, DOI 10.2514/1.33819