Regular Article

Reactive Obstacle-Avoidance for Agile, Fixed-Wing, Unmanned Aerial Vehicles

Eitan Bulka and Meyer Nahon
Department of Mechanical Engineering, McGill University, Montreal, Quebec, Canada

Abstract: Agile, fixed-wing, aircraft have been proposed for diverse applications, due to their enhanced flight efficiency, compared to rotorcraft, and their superior maneuverability, relative to conventional, fixed-wing, aircraft. We present a novel, reactive, obstacle-avoidance algorithm that enables autonomous flight through unknown, cluttered environments using only on-board sensing and computation. The method selects a reference trajectory in real-time from a pre-computed library, based on goal location, instantaneous point cloud data, and the aircraft states. At each time-step, a cost is assigned to candidate trajectories that are collision-free and lead to the edge of the obstacle sensor's field-of-view, with cost based on both distance to obstacles, and the goal. The lowest cost reference trajectory is then tracked. If all potential trajectories result in a collision, the aircraft has enough space to come to a stop, which theoretically guarantees collision-free flight. Our work demonstrates autonomous flight in unknown and unstructured environments using only on-board sensing (stereo camera, IMU, and GPS) and computation with an agile, fixed-wing, aircraft in both simulation and outdoor flight tests. During flight testing, the aircraft cumulatively flew $4.4km$ autonomously in outdoor environments with trees as obstacles with an average speed of $8.1ms^{-1}$ and a top speed of $14.4ms^{-1}$. To the best of our knowledge, ours is the first obstacle-avoidance algorithm suitable for agile, fixed-wing, aircraft that can theoretically guarantee collision-free flight and has been validated experimentally using only on-board sensing and computation in an unknown environment.

Keywords: aerial robotics, obstacle avoidance

1. Introduction

Unmanned aerial vehicles (UAVs) have been increasingly proposed for aerial surveillance, mapping, monitoring, and delivery tasks. Historically, these vehicles fall into two categories: rotorcraft and fixed-wing aircraft. Rotorcraft remain aloft by thrusting vertically, while fixed-wing aircraft use the lift generated by their wings. This fundamental difference causes rotorcraft to be less energy-efficient than fixed-wing aircraft, but also allows them to take off vertically, hover, and fly at low speeds through dense, cluttered environments. On the other hand, conventional, fixed-wing, aircraft require

Received: 18 February 2021; revised: 14 November 2021; accepted: 15 November 2021; published: 3 July 2022.

Correspondence: Eitan Bulka, Department of Mechanical Engineering, McGill University, Montreal, Quebec, Canada, Email: eitan.bulka@mail.mcgill.ca

DOI: https://doi.org/10.55417/fr.2022048

Figure 1. McFoamy, an agile, fixed-wing, UAV

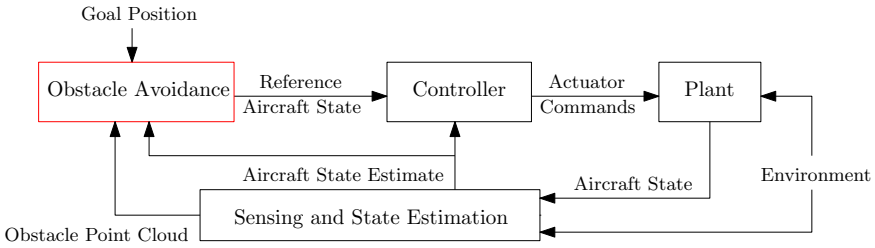


Figure 2. Autonomous Aircraft Block Diagram

runways to takeoff and land and cannot operate in cluttered environments, but have the ability to fly long distances efficiently.

An emerging class of UAVs, *agile, fixed-wing, aircraft* (depicted in Figure 1), aim to bridge the gap between rotorcraft and conventional, fixed-wing, aircraft. These fixed-wing aircraft are characterized by a high thrust-to-weight ratio (> 1), and large control surfaces capable of large deflections. These characteristics allow the aircraft to emulate rotorcraft maneuverability, yet maintain the ability to fly efficiently. These type of platforms are ideal for tasks that require both the ability to maneuver through cluttered environments, *and* the ability to fly long distances efficiently.

The ability of these aircraft to fly autonomously drastically increases their utility while, compared to other formats, those abilities are yet in early development stages. Although recent advances in control-system design have enabled autonomous, agile flight in open skies, independently sensing and avoiding collisions in congested environments remains an unmet challenge. The obstacle-avoidance problem is particularly difficult in the context of small, agile, fixed-wing, UAVs, as they typically fly at high speeds and have complex six degrees of freedom dynamics. Furthermore, these aircraft have a small payload capacity which significantly limits the available computing power and potential options for sensing obstacles. Achieving autonomous flight in cluttered environments with these aircraft spans several research topics, including: motion planning, obstacle detection, dynamics modelling, state estimation, and controller development. Figure 2 illustrates the relations among these complexities in our approach. In this article we focus specifically on avoiding obstacles, that is, generating a reference trajectory that steers the UAV away from potential collisions and towards the goal. Our avoidance algorithm coordinates with an obstacle-detection sensor, a state estimator, a controller, and a dynamics model, which are all available either through off-the-shelf hardware, open-source software, or prior research.

We introduce a novel obstacle-avoidance algorithm for agile, fixed-wing, aircraft that runs in real-time and uses only on-board sensing and computation. Our approach utilizes a depth sensor and specified goal location to select motion primitives from a library computed offline. In contrast to many other approaches, these motion primitives are steady-state – that is, trajectories that can be held indefinitely with constant control inputs, which allows us to match a selected target position to a motion primitive with little computation. At every time increment, we assign a cost to each collision-free candidate, and the low-level controller then tracks the least-cost option. In

a scenario where all candidate motion primitives result in a collision, there is enough space for the aircraft to execute the hover maneuver and come to a complete stop. This is achieved by choosing target positions at the edge of depth sensor's field-of-view strategically, which theoretically guarantees collision-free flight. We validate our approach in various simulation environments and in outdoor flight testing. During the flight testing campaign, the aircraft autonomously flew 4.4 km (cumulatively) in outdoor environments with trees as obstacles with an average speed of $8.1ms^{-1}$ and a top speed of $14.4ms^{-1}$. To the best of our knowledge, ours is the only obstacle-avoidance algorithm suitable for agile, fixed-wing, aircraft that can theoretically guarantee collision-free flight and has been validated experimentally using only on-board sensing and computation in an unknown environment.

In Sec. 2, we discuss the related work, which is followed by an overview of the dynamics modelling, controller development, and state estimation in Sec. 3. In Sec. 4, we present the obstacle-avoidance methodology. In Sec. 5, we discuss our simulation results, which is followed by an overview of our experimental platform in Sec. 6, and outdoor flight testing results in Sec. 7. We conclude the article in Sec. 8.

2. Related Work

Due to the complexity of achieving autonomous flight in cluttered environments, researchers often make various simplifying assumptions. While surveying the literature it is important to acknowledge if the solution presented has been implemented:

- in real-time or off-line
- in an unknown map or known map of the environment
- using on-board or off-board computation
- using on-board or off-board sensing
- with assumptions about the size or motion of the obstacles
- in flight experiments or simulation

In addition to the various assumptions that could be made, the applicability of a proposed algorithm will significantly depend on the type of vehicle and speed at which the vehicle flies.

While some researchers continue to publish methodologies with some of these underlying assumptions, others aim to relax these assumptions in order to plan the motion of UAVs in real-time in unknown environments using on-board sensing and computation with state-of-the-art hardware. Researchers designing algorithms for realistic operating scenarios are able solve the motion planning problem by breaking it down into two parts using a global and local planner (Scherer et al., 2008; Ryll et al., 2019). A global motion planner, operating at a slower update rate and reliant on a transformation of sensor data into a map of the world, is responsible for finding a collision-free trajectory to the goal if one exists. A local or reactive motion planner, operating at a faster update rate and using a more basic form of obstacle detection sensor data, is responsible for guaranteeing collision-free flight with newly-perceived obstacles.

The long-term goal of our research is to enable an agile, fixed-wing, aircraft to autonomously fly through an unknown and unstructured cluttered environment using only on-board sensing and computation, but we limit the scope of the present work to reactive obstacle-avoidance. Future work can combine the contributions of this work with on-board mapping and a global motion planner to achieve the long-term goal. Given the very limited research on algorithms that have been implemented on agile, fixed-wing, UAVs in real-time in unknown environments with on-board sensing and computation, we broaden our literature review to works that demonstrate motion planning with other types of UAVs in real-time in unknown environments with on-board sensing and computation, as well as works that present motion planning with fixed-wing UAVs, but are not implemented real-time in unknown environments with on-board sensing and computation.

2.1. Motion Planning Strategies Not Validated in Unknown Environments with On-board Sensing and Computation

One of the important early works on motion planning with UAVs is presented in (Frazzoli et al., 2002), in which a miniature helicopter navigates through an environment with sliding doors in a simulation environment. In this work, the helicopter motion is broken down into a finite number of trim states (steady-state), and maneuvers (finite time) which connect one trim state to another. These trim states and maneuvers formulate a library of motion primitives which are precomputed off-line. During flight, a tree in the rapidly exploring random trees (RRT) algorithm (LaValle, 1998; LaValle and Kuffner, 2001) is expanded using these motion primitives, as opposed to the entire state-space of the aircraft, which enables the algorithm to run in real-time.

In (Koyuncu et al., 2008), motion planning for an agile, fixed-wing, aircraft is achieved in three steps. First, a goal-biased RRT algorithm is used to find a connectivity path from initial to goal positions. Next, unnecessary detours are removed through a line-of-sight filter, and the refined path is used to generate a sequence of way points. Finally, the waypoints are converted into kinodynamically feasible paths using a model-based probabilistic roadmap (PRM)(Kavraki et al., 1996). In (Koyuncu et al., 2010), a similar approach is presented which generates a sequence of waypoints in the same manner, but this time employs a B-Spline method to generate dynamically feasible trajectories to connect waypoints. The authors believe once a library of flight modes is built (presumably off-line), their motion planning approach has real-time implementation capability.

In (Hwangbo et al., 2007), a two-phase motion planning approach is proposed, which consists of a coarse global planner and a fine local planner. The coarse global planner computes kinematically feasible obstacle-free path in a discretized three-dimensional space, while the fine local motion planner is used to compute a dynamically feasible trajectory using motion primitives. Ultimately, this algorithm uses a pre-computed (off-line) library of motion primitives to achieve real-time motion planning; demonstrated by performing an air slalom task in simulation. This work is extended in (Hwangbo and Kanade, 2013), in which the task is performed in flight, while introducing a Markov decision process to compute the most probable path to a target in the presence of wind gusts and imperfect control.

In (Paranjape et al., 2015), two families of motion primitives, three-dimensional circular paths and aggressive-turnarounds, are used in a receding horizon control-based motion planner to enable fast flight through a dense obstacle field with an agile, fixed-wing, aircraft. The motion planning results are presented in simulation, but control of the maneuvers is experimentally demonstrated in a motion capture environment.

In (Levin et al., 2018), an agile, fixed-wing, UAV motion planning framework is presented which utilizes RRTs in conjunction with a trajectory library, similar to the method in (Frazzoli et al., 2002). The authors achieve real-time motion planning in cluttered environments, all while utilizing the full capabilities of the aircraft, such as an aggressive turnaround and hovering maneuvers. The algorithm is validated in simulation in (Levin et al., 2018), and in flight tests in (Levin et al., 2019). During the flight tests all of the computation is done on-board, but there is no on-board sensing, and instead a map of the environment is provided to the motion planner at run time.

In (Bry and Roy, 2011), the rapidly-exploring random belief trees motion planning algorithm is proposed. Unlike the previously mentioned sampling-based methods, this algorithm accounts for uncertainties in the sensed environment, and converges to an optimal path. The methodology is tested in a simulation of a two-dimensional system with a Dubins-type vehicle dynamics, and is intended to be later implemented on a fixed-wing aircraft in real-time. Later in (Bry et al., 2015), the authors couple the search process with optimization in the output space of a differentially flat vehicles to find aggressive trajectories that utilize the full maneuvering capabilities of a quadrotor. This is extended to fixed-wing vehicles with a novel trajectory representation called a "Dubins-Polynomial trajectory". This planning strategy is used in conjunction with the work in (Bry et al., 2012), where an IMU and planar laser range finder are used in an extension of the Gaussian particle filter to localize a fixed-wing aircraft within a predetermined map of the environment. These algorithms

enable an autonomous fixed-wing aircraft traveling at over $11ms^{-1}$ to avoid obstacles in a GPS-denied parking garage.

A motion planning framework is presented in (Majumdar and Tedrake, 2017), where an agile, fixed-wing, aircraft avoids obstacles utilizing the full aerobatic capabilities of the aircraft. The algorithm runs in real-time and is guaranteed to succeed despite uncertainty. The approach utilizes a precomputed library of funnels along different maneuvers, where the system state is guaranteed to remain inside the funnel throughout the maneuver. While these results are very impressive, they are obtained in a motion capture environment where the obstacle locations are given to the planner at run-time, and the algorithm is run off-board.

Receding horizon path planning with implicit safety guarantees is presented in (Schouwenaars et al., 2004), and demonstrated in a simulation of a fixed-wing aircraft. In (Basescu and Moore, 2020), direct nonlinear model predictive control is used with a minimalistic dynamics model to maneuver an agile, fixed-wing, UAV in constrained spaces in real-time at 5 Hz. Randomized motion planning is used to avoid local minima and local-linear feedback is used to compensate for model inaccuracies between updates. The methodology is validated with flight tests through a virtual narrow corridor, where a motion capture system is used for sensing and the computation is done off-board.

Genetic algorithms have also been proposed in UAV path planning in (Nikolos et al., 2003). Two methods are proposed, an off-line planner in a known environment, and an on-line planner in an unknown environment. The off-line planner generates a single B-Spline curve that connects the starting and target points, while the on-line planner produces a trajectory comprised of multiple smaller B-Spline curves that are smoothly connected together. While no experimental results are presented, the on-line algorithm is meant to be used in conjunction with a radar.

A simple and computationally efficient class of reactive obstacle-avoidance algorithms are potential field methods, in which obstacles produce virtual repelling forces while the goal produces a virtual attractive force, ultimately avoiding obstacles and steering towards the goal. A popular potential field method, the vector field histogram (Borenstein and Koren, 1991), has been extended to three dimensions and applied to a quadrotor in simulation in (Zhao, 2015). Although not demonstrated in any simulation or flight test environment, a potential field based reactive controller for an agile, fixed-wing, UAV is presented in (Barry et al., 2012), which generates a desired yaw rate to steer around obstacles.

While the majority of active research focuses on autonomous flight through cluttered environments with static (or slowly-moving) obstacles, there has also been research on UAVs flying in a relatively open sky, where the other obstacles are other aircraft. Most commercial aircraft transmit an Automatic Dependent Surveillance-Broadcast (ADS-B) message, which allows a UAV with an ADS-B receiver to receive the position and velocity information of other nearby aircraft. These ADS-B messages are utilized in a dynamic obstacle-avoidance framework for fixed-wing UAVs presented in (Lin and Saripalli, 2015; Lin and Saripalli, 2017). Closed-loop RRT is used to generate intermediate avoidance waypoints to ensure collision-free flight. The methodology is validated using recorded ADS-B messages nearby an airport within a hardware-in-the-loop simulator.

2.2. Motion Planning Strategies Validated in Unknown Environments with On-board Sensing and Computation

We now turn our attention to motion planning strategies that have been validated experimentally in more realistic operating environments, but in most cases using aerial vehicles other than fixed-wing aircraft.

The Vector Field Histogram is experimentally validated on a quadrotor equipped with a two-dimensional LiDAR in (Van Breda, 2016). Two-dimensional (constant height) obstacle-avoidance around two large panels is demonstrated.

High-speed ($3ms^{-1}$) collision avoidance in unknown environments using on-board sensing and computation is achieved with a quadrotor in (Liu et al., 2016). A short range planner uses a local map to generate a dynamically feasible collision-free trajectory with a safe stopping policy. Collision

avoidance is guaranteed since the quadrotor is always able to come to an emergency stop if needed. After an emergency stop, a long range planner can be used for completeness.

A computationally efficient collision avoidance strategy using instantaneous perception data for high-speed quadrotor flight is presented in (Lopez and How, 2017a). By exploiting the differential flatness of quadrotors, minimum-time motion primitives are generated in real-time and a cost for each primitive is assigned based on angle differences between heading and heading to the goal, and the angle difference between heading and previously selected heading. The lowest cost collision-free primitive is selected to execute. Safety is guaranteed by restricting the motion primitives to remain within the sensor's field-of-view and ensuring a stop maneuver is possible if no collision-free path exists. Experimental results are demonstrated in a cluttered environment with a quadrotor flying at $3ms^{-1}$ with on-board perception, planning, and control. A motion capture system is used, but only for vehicle state information. This work is extended in (Lopez and How, 2017b) by using a relaxed constraint Model Predictive Control framework. A main component of the methodology is the ability to safely use a previously generated motion primitive which enables motions outside the perception field-of-view to guarantee safety.

The standard method of guaranteeing safety in unknown environments is by ensuring the ability to stop within the known free space. This imposes limitations on the speed of the vehicle. The work in (Tordesillas et al., 2019; Tordesillas et al., 2020) aims to ensure safety without sacrificing speed. The proposed methodology, FASTER (Fast and Safe Trajectory Planner), enables a quadrotor to autonomously fly in unknown cluttered environments at speeds up to $7.8ms^{-1}$. The approach optimizes trajectories in both the free-known and unknown spaces while ensuring safety by always having a feasible, safe back-up trajectory in the free-known space at the start of each replanning step.

A methodology presented in (Ryll et al., 2019) enables a quadrotor to autonomously fly in flights totaling 22 km in unknown urban environments at speeds up to $9.4ms^{-1}$. A sparse global path to the goal is found using an occupancy grid. A position along the global path is chosen as the local goal for the local planner. A set of minimum-jerk motion primitives are computed and evaluated based on the distance to local goal, dynamic feasibility, depth image, and occupancy grid. The best motion primitive is followed until the next depth image arrives.

In (Scherer et al., 2008), a two-stage three-dimensional motion planner for UAVs is presented, which uses a potential field based method for local planning and a Laplacian-based planner for global planning. The algorithm is applied to a large unmanned RC helicopter equipped with a three-dimensional LiDAR. The methodology was demonstrated in over 700 successful runs of obstacle-avoidance in unchartered cluttered environments traveling at speeds up to $10ms^{-1}$, all using on-board sensing and computation.

A vision-based reactive obstacle-avoidance strategy called R-ADVANCE (Rapid Adaptive Prediction for Vision-based Autonomous Navigation, Control, and Evasion) is presented in (Escobar-Alvarez et al., 2018). In this work, a quadrotor autonomously avoids obstacles at speeds up to $19ms^{-1}$, using only on-board sensing and computation.

Obstacle-avoidance with a 20 g flapping-wing aircraft using on-board sensing and computation is demonstrated in (Tijmons et al., 2017). The "Droplet" strategy is proposed which uses a droplet shaped region such a that a future circular trajectory remains within the current field-of-view of the stereo cameras. If an obstacle enters this droplet region, the aircraft turns along that circular trajectory until a new collision-free droplet region is found. In the event a collision-free droplet does not exist while turning, the aircraft can remain in this circular collision-free trajectory indefinitely, guaranteeing collision-free flight (in the absence of sensor and motor noise).

Insect-inspired optic-flow based collision avoidance is demonstrated in an unknown environment using on-board sensing and computation with an agile, fixed-wing, UAV in (Green and Oh, 2008). The approach is used to avoid lateral collisions but fails when obstacles are directly in front of the aircraft. In the failure scenario, the authors propose using a distance sensor to initiate a transition to hover to avoid the collision.

The most significant research in fixed-wing obstacle-avoidance in unstructured and unknown environments is presented in (Barry et al., 2018; Barry, 2016). A fixed-wing aircraft detects and

avoids trees flying up to $14ms^{-1}$ using all on-board sensing and computation, without the prior knowledge of a map of the environment. The obstacle-avoidance algorithm relies on a library of precomputed trajectories, and the trajectory that is furthest from the point cloud of obstacles is selected. While this work is extremely impressive, it is open to improvement. The author mentions two failure scenarios with this planning algorithm: one associated with an insufficiently rich maneuver library, and the other with the trajectory initial state. The first failure mode occurs when there is no trajectory in the library that can avoid the obstacle. The second failure mode is best illustrated through an example. If the aircraft needs to bank right, and is already slightly banked right, but the time-varying 'bank right' maneuver starts from level flight, the aircraft will initially bank left to first return to level flight. The aircraft will ultimately not turn as quickly, potentially causing a collision. In addition to the failure modes, the planning algorithm does not have a goal: the aircraft is commanded to fly straight, unless it needs to avoid obstacles.

3. Modelling, Control, and State Estimation

Our obstacle-avoidance methodology relies on a dynamics model of the vehicle, and must be used in conjunction with a feedback controller and state estimator. In prior work, we develop a dynamics model and feedback controller for an agile, fixed-wing, aircraft. For state estimation, we utilize an open-source algorithm. In this section, we overview the modelling, control, and state estimation, and cite the articles in which they are originally presented.

3.1. Modelling

This subsection discusses the mathematical modeling of agile, fixed-wing, UAV motion. The model is based on first principles and can be used for various agile, fixed-wing, aircraft. A complete detailed description of the model can be found in (Khan, 2016), and the components of the model are presented in various articles which are cited throughout this summary.

3.1.1. Coordinate Frames

We use two reference frames to describe the dynamics of a UAV. $\mathcal{F}_i$ is the ground-fixed inertial reference frame with north-east-down basis vectors and $\mathcal{F}_b$ is the body-fixed reference frame. A third frame, which is utilized in Sec. 4, is fixed to the obstacle detection sensor and denoted by $\mathcal{F}_c$. All of these frames are depicted in Figure 3.

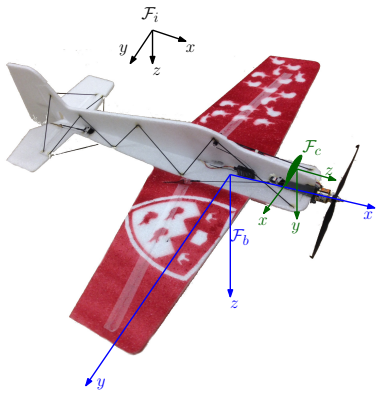


Figure 3. Coordinate Frames

3.1.2. UAV Kinematics and Dynamics

The translational and rotational dynamics of a UAV can be derived from the Newton-Euler equations for a single rigid body, which can be succinctly stated in the following first-order form:

$$\dot{\mathbf{p}}_i = \mathbf{C}_{bi}^T \mathbf{v}_b \tag{1}$$

$$\dot{\mathbf{v}}_b = \frac{\mathbf{f}_b}{m} - \boldsymbol{\omega_b} \times \mathbf{v}_b \tag{2}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \odot \boldsymbol{\omega}_b \tag{3}$$

$$\dot{\boldsymbol{\omega}}_b = \mathbf{I}_b^{-1}\left(\mathbf{m}_b - \boldsymbol{\omega}_b \times \mathbf{I}_b\boldsymbol{\omega}_b\right) \tag{4}$$

where $\odot$ is the Hamilton quaternion product, $\mathbf{p}_i$ is the absolute position of the UAV centre of mass expressed as components in $\mathcal{F}_i$ (designated with subscript i) and $\mathbf{q}$ is the aircraft orientation, expressed as a unit quaternion, $\mathbf{q} = \left[q_0, \mathbf{q}_{1:3}^T\right]^T$. Analogously at the velocity level, $\mathbf{v}_b$ is the translational velocity of the centre of mass and $\boldsymbol{\omega}_b$ is the rotational velocity of the UAV, both expressed in the body-fixed frame (designated with subscript b). The direction cosine matrix $\mathbf{C}_{bi}$ describes the orientation of the body frame relative to the inertial frame and can be formed from $\mathbf{q}$. The mass of the UAV is denoted by m, and $\mathbf{I}_b$ is the moment of inertia relative to the center of mass, resolved in the body frame.

The cumulative forces and moments acting on the UAV are denoted by $\mathbf{f}_b$ and $\mathbf{m}_b$, respectively. These forces and moments stem from gravity, aerodynamics, and the thruster. While the gravity force is trivial to obtain, the rest of these generalized forces are difficult to model. We use three lower-level models to model these forces and moments: a thruster model, a propeller slipstream model, and an aerodynamics model.

3.1.3. Thruster Model

The thruster model (Khan and Nahon, 2013) uses the propeller geometry to predict the total force (3-axis) and moment (3-axis) exerted by the propeller as a function of the incoming airflow and propeller rotational speed. The thruster model is based on blade element momentum theory, coupled with an inflow model to predict the thruster generalized forces. The model also includes the modelling of the thruster gyroscopic effects, and can account for various incoming airflow conditions that can occur in an aerobatic flight of an agile, fixed-wing, aircraft. These incoming airflow conditions can be: stationary (when the aircraft is hovering), pure axial flow (flow aligned with the propeller rotational axis, i.e. level flight), oblique flow conditions (flow at an angle to its rotation axis, due to aerobatic flight or wind gusts), and in reverse flow conditions.

3.1.4. Propeller Slipstream Model

The propeller slipstream has an important impact on the ability of the control surfaces to generate control forces. The slipstream provides additional flow over the control surfaces, enabling the aircraft to maintain control authority at low speeds. The novel slipstream model (Khan and Nahon, 2015a) includes both the acceleration and diffusion phenomena that occur in the slipstream. The model has been shown to accurately predict the axial slipstream velocity up to 6 propeller diameters downstream of the propeller.

3.1.5. Aerodynamics Model

The aerodynamics model (Khan and Nahon, 2015b) uses the aircraft geometry to predict the aerodynamic forces exerted on the aircraft as a function of the incoming airflow (which includes the propeller slipstream). These aerodynamic forces are calculated based on a component breakdown approach that partitions each component of the aircraft (wing, tail, rudder etc.) into small segments that produce lift, drag, and moment about the aircraft's center of mass. The aerodynamics model includes modelling of: the full flight envelope (i.e $\pm$ 180° angle of attack and sideslip range), partial flow conditions over the aerodynamic surfaces, low aspect ratio surfaces, and large control surfaces with large deflections.

3.1.6. Model Validation

The individual components of the physics-based model, including thruster, slipstream, and aerodynamics have been individually validated in (Khan, 2016) through wind-tunnel tests and static bench

tests. In addition to the validation of the component models, the complete flight simulator was qualitatively validated through pilot-in-the-loop simulations, in which an experienced professional RC pilot found the simulator to be accurate (Khan, 2016). In addition to qualitative validation, the complete flight simulator was validated quantitatively by comparing the simulated and in-flight translational and angular acceleration. A good match was found between the predicted and experimental translational acceleration, while there were larger discrepancies in angular acceleration (Bulka, 2021).

3.2. Controller

Our obstacle-avoidance technique algorithm provides a reference trajectory for the aircraft control system to follow, which it achieves by adjusting the aircraft's control inputs: aileron (u_2^s), elevator (u_3^s), rudder (u_4^s) and thrust (u_1^s). This is a challenging control problem, because the system is under-actuated, and the actuators' effectiveness varies with the aircraft speed. We present a *single* controller capable of tracking a wide range of maneuvers in (Bulka and Nahon, 2019a), and provide a summary of the control logic in this section. The nonlinear control system is based on the physics of the aircraft, allowing simple control laws to achieve precise tracking of highly non-linear dynamics. The physics used in the control algorithm is based on simplified models that require little computation, as opposed to using the detailed, but computationally expensive model discussed in Sec. 3.1. The controller does not rely on any plant linearization, to ensure that the controller will remain effective throughout the entire flight envelope of the agile aircraft.

The control architecture is modular, containing a position controller, force controller, attitude controller, and control allocation, as shown by the block diagram in Figure 4. The controller receives estimates of the pose and twist of the aircraft ($\mathbf{p}_i$, $\mathbf{q}$, $\mathbf{v}_b$, $\boldsymbol{\omega}_b$) from the state estimator, and receives the reference pose and twist ($\mathbf{p}_i^{ref}$, $\mathbf{q}^{ref}$, $\mathbf{v}_r^{ref}$, $\boldsymbol{\omega}_r^{ref}$) from the obstacle-avoidance algorithm. First, the position controller uses the translational position and velocity errors to modify the reference orientation. The rationale is that the control surfaces of a fixed-wing aircraft can control orientation but cannot directly control position, but the orientation of an aircraft *can* indirectly control the aircraft's position. This new augmented orientation, $\bar{\mathbf{q}}^{ref}$, is similar to the reference orientation $\mathbf{q}^{ref}$, but modified to correct translational motion errors. The attitude controller then generates control moments that track this augmented orientation. The force control is decoupled from the position and attitude controllers, and its goal is to counteract gravity and the aerodynamic forces, as well as track the height and velocity of the UAV. The control-allocation step uses simplified models to predict the thrust force produced by the propeller and the control moment generated by a control surface deflection, and ultimately maps the commanded control moment and force ($\mathbf{m}_b^c$ and f^c) to a given set of actuator commands ($\mathbf{u}^s$). We have validated the feedback control strategy in extensive flight tests, where a wide variety of maneuvers spanning a large portion of the aircraft's flight envelope were demonstrated in (Bulka and Nahon, 2019a). In addition to experimental validation, the feedback controller is shown to be asymptotically stable using a Lyapunov stability analysis in (Bulka and Nahon, 2021).

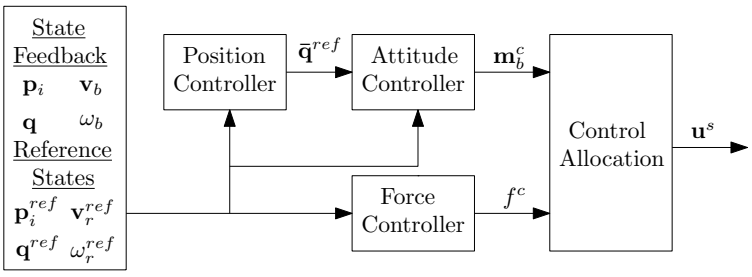


Figure 4. Control Architecture

3.3. State Estimation

Our obstacle-avoidance methodology relies on obtaining estimates of the aircraft's position, velocity, orientation, and angular velocity from the typical on-board sensors of a UAV, an IMU and GPS. We utilize the open-source flight stack, PX4 (Meier et al., 2015), to obtain these state estimates. The flight stack uses an Extended Kalman Filter, named EKF2, to estimate the UAV states by fusing on-board measurements from an accelerometer, gyroscope, magnetometer, barometer, and GPS.

4. Obstacle-Avoidance

Navigating a robot through an unknown environment is a complex problem that spans multiple research topics, including perception, motion planning, control theory, state estimation, and dynamics modeling. The typical approach consists of a robot equipped with sensors moving through an environment which simultaneously map the environment and locate itself within it (SLAM). A motion planner then uses this map to generate a collision-free reference trajectory that reaches the goal, and the controller outputs actuator signals to track this reference trajectory.

This problem is significantly more difficult in the context of unmanned aerial vehicles, due to their limited payloads and complex, six-DoF dynamics. The small payload restricts the robot's computational power and sensing capabilities. Complex dynamics complicate motion-planning when considering real-time implementation with limited computational power. The computational demands of executing a SLAM algorithm while generating motion plans on an on-board processor results in update rates too slow to avoid obstacles during high speed flight.

These issues are the reason autonomous flight through cluttered, previously unknown environments is an active research topic. Most progress in this area has been demonstrated with rotorcraft. There the issue of limited computational power has been addressed by separating the motion-planning problem into two parts: a relatively slow, global planner that requires a dynamically-constructed map of the environment to be built while flying and a faster, local, reactive planner that uses only real-time sensor data (Scherer et al., 2008; Ryll et al., 2019).

In the general model, the global planner is responsible for finding a collision-free trajectory to the goal, if one exists, and the local motion planner handles avoiding collisions with newly-perceived obstacles. Figure 5 illustrates the complete motion-planning architecture. The global planner utilizes the map of the environment and gives the obstacle-avoidance algorithm a local goal, which is a position along the global path. The primary objective of the obstacle-avoidance is to avoid obstacles, and reaching the local goal is secondary, since the global planner will eventually update this local goal. Furthermore, it is not possible to guarantee the obstacle-avoidance will reach the local goal, since it only operates on instantaneous, point cloud data.

Autonomously flying through unknown, cluttered environments with fixed-wing aircraft is even more complicated than with rotor-craft, due to the nature and complexity of their dynamics. The majority of researchers tackling this problem simplify it by assuming a *known* map of the environment. Previous work within the research group (Levin, 2019) makes this assumption and focuses on trajectory generation and global motion-planning with agile, fixed-wing, UAVs in static, known environments. The global planner utilizes a library of motion primitives built off-line,

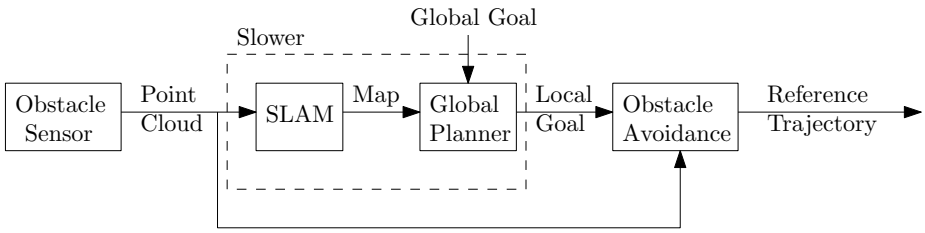


Figure 5. Block Diagram of Complete Motion Planner

which are then used in a real-time RRT motion planning framework to generate collision-free, dynamically-feasible trajectories to a goal region.

To the best of our knowledge, (Barry, 2016) is the only work describing autonomous, fixed-wing flight in an unknown, cluttered environment while using only onboard sensing and control. While their work is impressive and the most advanced to date, during their outdoor flight testing campaign their aircraft successfully avoided obstacles 16 times, but failed to do so 10 times.

4.1. Obstacle-Avoidance Overview

We limit the scope of this article to reactive obstacle-avoidance, as is done in (Barry, 2016), with the notion that it would be combined with a SLAM implementation and the global planner in (Levin, 2019) in the future. The scenario addressed in this article is represented by a block diagram in Figure 6, where the obstacle-avoidance algorithm generates reference trajectories based on an instantaneous point cloud and a goal location.

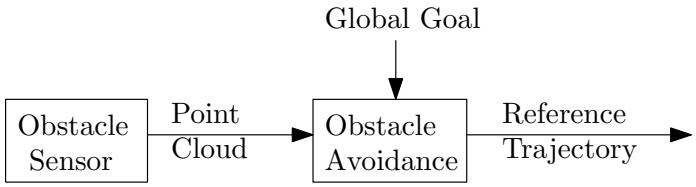


Figure 6. Block Diagram of Obstacle-Avoidance

In this situation, the primary purpose of the obstacle-avoidance algorithm is to avoid obstacles, and reaching the goal is secondary, since it only operates on instantaneous point cloud data. We base our methodology on a point cloud representation of obstacles because most obstacle detection sensors output a point cloud, such as a LiDAR or stereo camera. Our experimentation uses a stereo camera since it is the only sensor that fits within the aircraft's payload capacity.

As mentioned previously, the work in (Levin, 2019) utilizes a library of motion primitives built off-line, which are selected in real-time to navigate through a *known* environment. In this work, we utilize the same library of motion primitives, which is briefly discussed in Sec. 4.2. Unlike in (Levin, 2019), in this work trajectories are selected from the library in conjunction with on-board obstacle detection, which ultimately enables the aircraft to autonomously avoid collisions and reach the goal location in *unknown* environments. This selection process is detailed in Sec. 4.3.

4.2. Trajectory Generation

Many motion planning algorithms propagate the robot's dynamics model several times before obtaining a reference trajectory (LaValle, 2006), but this is not feasible in real-time, with limited computational power and a complex dynamics model. The most common approaches are to reduce the complexity of the dynamics model so that it can be executed in real-time or to run the computationally expensive dynamics model off-line, and store it in a library for real-time use. The former approach involves making various simplifying assumptions that reduce the flight mechanics to analytical equations, which is a common approach with conventional, fixed-wing, aircraft (Russell, 1996). These simplifying assumptions enable the motion planner to propagate the dynamics as needed, but at the cost of inaccurate modeling due to the approximation. The latter approach enables the motion planner to use the complex dynamics model, but limits the motion plan to only those options that have been stored in the trajectory library.

We have at our disposal a highly accurate, but computationally expensive, dynamics model developed in our research group (Khan, 2016). As well as existing methodologies pertaining to agile, fixed-wing, aircraft have successfully utilized a pre-computed trajectory library for real-time global motion planning in known environments in (Levin, 2019), and real-time obstacle-avoidance in

unknown environments in (Barry, 2016). We therefore elect to use a pre-computed trajectory library for our obstacle-avoidance methodology. The library of motion primitives computed off-line enables the strategy to exploit a large portion of the aircraft's flight envelope without solving complex equations of motion in real-time.

The reference trajectories used for collision avoidance should be dynamically feasible with respect to the modelling discussed in Sec. 3.1. We build our library by solving trajectory optimization problems, using the method presented in detail in (Levin et al., 2019; Levin, 2019), and briefly discussed in this section. We obtain our trajectory library using a MATLAB optimal control software, GPOPS-II (Patterson and Rao, 2014). Each trajectory in the library is the solution of a trajectory optimization problem that is constrained to the vehicle dynamics model in Sec. 3.1 and physical actuator limits. In (Levin et al., 2019; Levin, 2019), trajectories are classified into two classes: trim primitives, which are steady-state maneuvers that can be held indefinitely, and agile primitives, which are of finite-time. In this work, we utilize trim primitives to maneuver through cluttered environments, and utilize one agile maneuver, the cruise-to-hover, when either reaching the goal or executing an emergency stop.

4.2.1. Trim Primitives

In addition to the constraints mentioned above, trim primitives are constrained to constant speed, roll, pitch, and control inputs. They are also constrained to having no sideslip, and the optimization objective is to minimize the control effort. Each trim primitive is defined by its speed, yaw rate and climb rate, which are all constant throughout the trajectory.

Trim primitives make up all of the trajectory library, except for the cruise-to-hover maneuver discussed later in Sec. 4.2.2. Specifying various combinations of yaw rate and climb rate result in straight and level flight, climbs, descents, banked turns, and helical banked turns. For this work, we limit the collision-avoidance strategy to constant speed flight, except during the cruise-to-hover, and build a library of trim primitives. In (Levin, 2019), all of the validation is performed using a constant speed of $7ms^{-1}$, and motion primitives are obtained with yaw rates of -110 to $110°s^{-1}$ in increments of $10°s^{-1}$, and climb/descent rates from -2 to $2ms^{-1}$ in increments of $1ms^{-1}$ to form a library of 115 trim trajectories consisting of straight and level flight, 2 climbs, 2 descents, 22 banked turns, and 88 helical banked turns. A top-down view of the motion primitives for different yaw rates with a zero climb rate are depicted in Figure 7, and a side view showing different climb/descent rates with a zero yaw rate are depicted in Figure 8. Figure 9 shows the particular case of a helical turn trim primitive, with a yaw rate of $110°s^{-1}$ and a decent rate of $2ms^{-1}$.

The work in this paper considers operation at various speeds, and thus a trim primitives library is generated for speeds of $7ms^{-1}$, $9ms^{-1}$, $11ms^{-1}$ and $13ms^{-1}$. Solving these trajectory optimization problems at multiple speeds shows that, as the aircraft is flying faster, the maximum yaw rate decreases. Thus, the full trajectory library of yaw rates from -110 to $110°s^{-1}$ and climb/descent rates from -2 to $2ms^{-1}$ could not be found at higher speeds. Further investigation shows that this

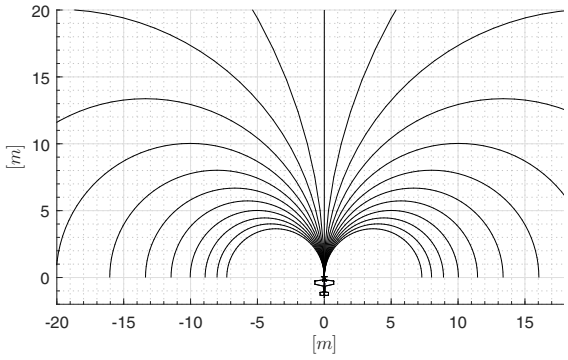


Figure 7. Trim Primitive Top-down View (Levin, 2019)

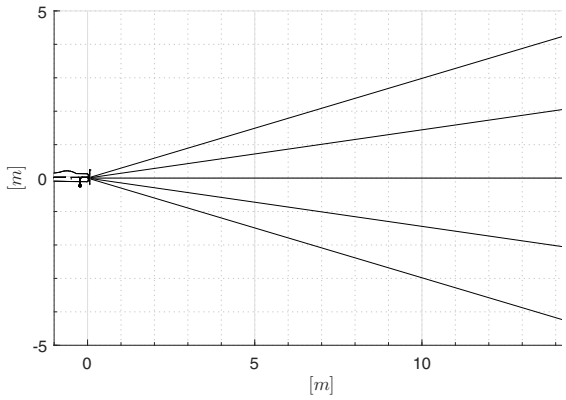


Figure 8. Trim Primitive Side View (Levin, 2019)

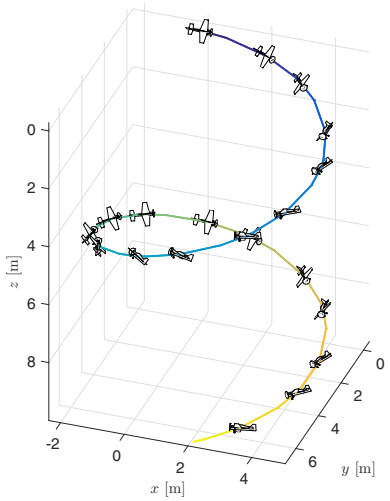


Figure 9. Helical Turn (Levin, 2019)

reduction in maximum yaw rate is caused by throttle saturation, since increasing the maximum thrust by 50% enables the trajectory optimization to find a full trajectory library at all the speeds tested. This phenomenon is consistent with intuition since, at constant speed, a turning aircraft requires more thrust than flying straight, due to the loss of lift while banking. Table. 1 summarizes the feasible combinations of yaw rate and climb/descent rate, based on speed. While Table. 1 contains various yaw rates, climb rates, and speeds, only one speed is used for the duration of a flight (i.e. one column of Table. 1).

4.2.2. Cruise-to-Hover

The only agile motion primitive from (Levin et al., 2019; Levin, 2019) used in this work is the cruise-to-hover maneuver. Agile motion primitives are obtained by solving a trajectory optimization problem which are only constrained to the vehicle's dynamics model, physical actuator limits, and specified boundary conditions. The objective function of the optimization is to minimize the duration of the trajectory and the time derivative of the control inputs throughout the trajectory. This results in aggressive trajectories with smooth control inputs. Varying the boundary conditions results in different agile maneuvers. For the specific case of the cruise-to-hover maneuver, the initial boundary condition requires the aircraft to be in steady level flight, and the final boundary condition requires the aircraft to be in a stationary hover. The cruise-to-hover maneuver is shown in Figure 10.

Table 1. Trajectory feasibility with varying speed

$\dot{\psi}^{ref}$ $(^{\circ}s^{-1})$	$v_{i,z}^{ref}$ (ms^{-1})	$\|\|\mathbf{v}_i^{ref}\|\| = 7ms^{-1}$	$\|\|\mathbf{v}_i^{ref}\|\| = 9ms^{-1}$	$\|\|\mathbf{v}_i^{ref}\|\| = 11ms^{-1}$	$\|\|\mathbf{v}_i^{ref}\|\| = 13ms^{-1}$
0	-2	✓	✓	✓	✓
0	-1	✓	✓	✓	✓
0	0	✓	✓	✓	✓
0	1	✓	✓	✓	✓
0	2	✓	✓	✓	✓
10	-2	✓	✓	✓	✓
10	-1	✓	✓	✓	✓
10	0	✓	✓	✓	✓
10	1	✓	✓	✓	✓
10	2	✓	✓	✓	✓
20	-2	✓	✓	✓	✓
20	-1	✓	✓	✓	✓
20	0	✓	✓	✓	✓
20	1	✓	✓	✓	✓
20	2	✓	✓	✓	✓
30	-2	✓	✓	✓	✓
30	-1	✓	✓	✓	✓
30	0	✓	✓	✓	✓
30	1	✓	✓	✓	✓
30	2	✓	✓	✓	✓
40	-2	✓	✓	✓	
40	-1	✓	✓	✓	
40	0	✓	✓	✓	
40	1	✓	✓	✓	✓
40	2	✓	✓	✓	✓
50	-2	✓	✓		
50	-1	✓	✓	✓	
50	0	✓	✓	✓	
50	1	✓	✓	✓	✓
50	2	✓	✓	✓	✓
60	-2	✓	✓		
60	-1	✓	✓	✓	
60	0	✓	✓	✓	
60	1	✓	✓	✓	
60	2	✓	✓	✓	✓
70	-2	✓	✓		
70	-1	✓	✓		
70	0	✓	✓	✓	
70	1	✓	✓	✓	
70	2	✓	✓	✓	✓
80	-2	✓			
80	-1	✓	✓		
80	0	✓	✓		
80	1	✓	✓	✓	
80	2	✓	✓	✓	
90	-2	✓			
90	-1	✓	✓		
90	0	✓	✓		
90	1	✓	✓	✓	
90	2	✓	✓	✓	
100	-2				
100	-1	✓			
100	0	✓	✓		
100	1	✓	✓		
100	2	✓	✓	✓	
110	-2				
110	-1				
110	0	✓			
110	1	✓	✓		
110	2	✓	✓		

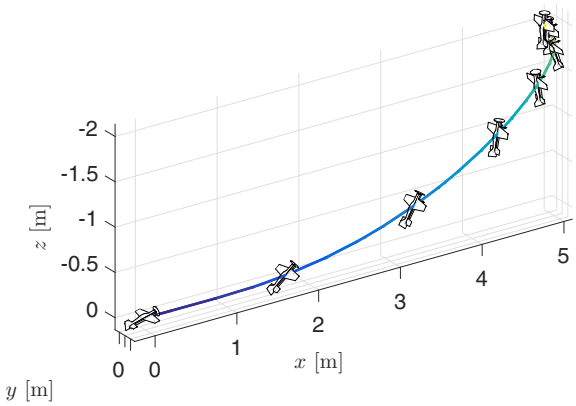


Figure 10. Cruise-to-Hover (Levin, 2019)

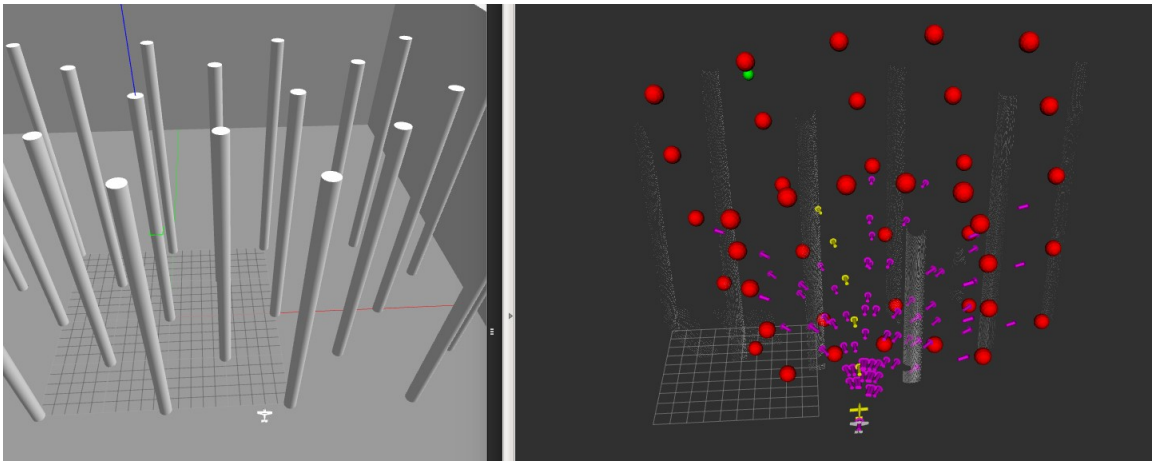


Figure 11. Example of one motion planning time-step in Gazebo (left) and RVIZ (right)

4.3. Trajectory Selection

While the aircraft is cruising, we use the trim primitives in our library to avoid obstacles and steer the aircraft towards the goal. Using the point cloud from a stereo camera we design a motion plan within the aircraft's field-of-view (FOV).

First we compute a set of trim primitives from the current aircraft location to various locations on the edge of the FOV. The target locations on the edge of the FOV are represented by the red dots in RVIZ in Figure 11, and the trim trajectories which end at those locations are represented by the purple arrows, where the direction of the arrow is the yaw along the trajectory. As shown in the figure, not all final target positions are reachable using dynamically feasible trim primitives. The final target positions are fixed to the edge of the FOV, and their location is therefore a function of the aircraft attitude (and position). The selection of the final target locations and computation of the trim primitive is discussed in detail in Sec. 4.3.1.

Next, the minimum distance between every point in the point cloud and each potential trajectory is computed and used to determine whether the potential trajectory will result in a collision. This computation is discussed in detail in Sec. 4.3.2. A cost is assigned to each collision-free trajectory, where this cost increases by: being near obstacles, steering the aircraft away from the goal, or changing the current heading rate. The trim trajectory with the lowest cost is selected for the aircraft to track. Referring to Figure 11, the point cloud is represented by the white dots, the goal

Algorithm 1. Reference Trajectory Selection.

```
Input: point cloud, aircraft state estimate, goal position, trajectory library
Output: reference trajectory
for all final target positions do
    compute the trim primitive that reaches final target position
    if the trim primitive exists in the library then
        compute minimum distance between trim primitive and all of the points in the point cloud
        if minimum distance is greater than half the aircraft wingspan then
            assign cost to trim primitive
if no trajectories have been assigned a cost then
    return cruise-to-hover
else
    return minimum cost trim primitive
```

location is represented by the green sphere, and the yellow arrows represent the minimum cost trajectory. The trajectory costs are discussed in detail in Sec. 4.3.3.

If there are no collision-free trim trajectories which lead to the edge of the FOV, the aircraft will execute a cruise-to-hover maneuver. The final target positions at the edge of the FOV are chosen such that the aircraft always has enough space to execute a collision-free cruise to hover, which theoretically guarantees collision-free flight under certain assumptions. This notion of safety is discussed further in Sec. 4.3.4.

In any scenario, the obstacle-avoidance will send the controller a reference trajectory. The reference states are extracted from the reference trajectory (discussed in Sec. 4.3.6) and the controller generates the actuator commands to track the reference trajectory as discussed in Sec. 3.2. The reference state is represented by the yellow aircraft in Figure 11.

We summarize this reference trajectory selection process using pseudocode in Algorithm 1.

4.3.1. Obtaining Trajectories to Evaluate

Evaluating too many trajectories increases the computation time and ultimately slows down the speed at which the obstacle-avoidance algorithm can run. By contrast, evaluating too few trajectories doesn't utilize all of the maneuvering capability of the aircraft, and could potentially cause the collision-avoidance algorithm to believe there are no collision-free trajectories, when in reality there are. We select 41 final target positions along the edge of the FOV, and compute the trim primitive which leads to each target position. These final positions at the edge of the FOV are chosen considering the safety analysis presented later in Sec. 4.3.4, which ensures that any trajectory chosen will always have enough space to perform an emergency hover and prevent a collision.

The 41 final target positions are made up of (a) 25 final positions in an equally spaced 5×5 grid at the end of the range of the depth sensor ($20m$), and (b) 16 final positions on the horizontal and vertical edges of the FOV at an intermediate exit distance ($10m$). Larger exit distances allow the aircraft more time to react to obstacles, but limit the aggressiveness of the maneuvers. On the contrary, smaller exit distances allow for more aggressive maneuvering which is necessary to fly through certain environments. We compute these final positions using Eqs. (5 & 6), which are derived from the geometry shown for an arbitrary final target position in Figure 12.

We use the camera coordinate system, $\mathcal{F}_c$, in Figure 12, where the z-axis points out of the camera shown in red. The axes are represented by black dotted lines, and the $x-z$ plane at $y=0$ is shown in red. The straight line vector from the aircraft's current position to the final position ($\mathbf{p}_c^0$ to $\mathbf{p}_c^f$), which has a magnitude of l, is shown using the solid black line. The projection of this vector onto the $x-z$ plane is shown with the dotted black line, and the angle between the vector and its projection is represented by γ_v. The angle between the projection and z-axis is denoted by γ_h.

In order to obtain the 25 final positions in an equally spaced 5×5 grid at the end of the range of the depth sensor, l is set to the camera range, while every combination of γ_v and γ_h are chosen from the

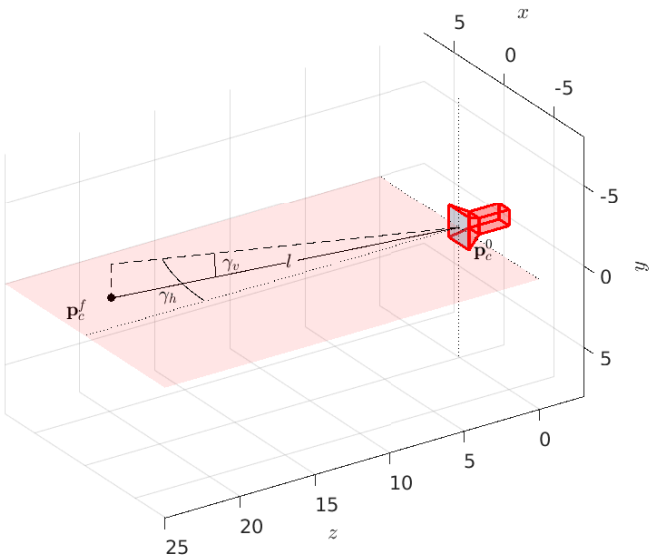


Figure 12. Computing final positions in the camera coordinate frame

sets of $\gamma_v = \{-\frac{VFOV}{2}, -\frac{VFOV}{4}, 0, \frac{VFOV}{4}, \frac{VFOV}{2}\}$ and $\gamma_h = \{-\frac{HFOV}{2}, -\frac{HFOV}{4}, 0, \frac{HFOV}{4}, \frac{HFOV}{2}\}$, where the $VFOV$ is the vertical field-of-view, and $HFOV$ is the horizontal field-of-view. Obtaining the 16 final positions at an intermediate exit distance can be obtained by setting l to a smaller value than the range of the depth sensor, but large enough to ensure a safe stopping maneuver is possible (see Sec. 4.3.4). At this intermediate exit distance the final positions must remain on the edge of the FOV, and thus only combinations of γ_v and γ_h are used when at least one angle is at its maximum or minimum value. In practice, we found it advantageous to set the value of the $HFOV$ in the algorithm to slightly less than the manufacturer's spec of the camera, to ensure the trajectories remain within the FOV even with some control or camera errors.

For arbitrary values of l,γ_h, and γ_v, we can compute the position vector from the aircraft's current position to the final position, resolved in the camera frame as

$$\mathbf{p}_c^f - \mathbf{p}_c^0 = \begin{bmatrix} l\cos\gamma_v \sin\gamma_h \\ l\sin\gamma_v \\ l\cos\gamma_v \cos\gamma_h \end{bmatrix}, \tag{5}$$

which ultimately is used to compute the final position resolved in the inertial frame as

$$\mathbf{p}_i^f = \mathbf{C}_{bi}^T \mathbf{C}_{cb}^T \begin{bmatrix} l\cos\gamma_v \sin\gamma_h \\ l\sin\gamma_v \\ l\cos\gamma_v \cos\gamma_h \end{bmatrix} + \mathbf{p}_i^0. \tag{6}$$

The direction cosine matrix from the body-fixed frame to the camera frame is denoted by $\mathbf{C}_{cb}$, where

$$\mathbf{C}_{cb} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}. \tag{7}$$

As we can see in Eq. (6), the final target positions are a function of both the aircraft's current position and orientation, since the camera is rigidly mounted to the aircraft. These final positions are shown in red in Figure 13. Evaluating up to 41 trim trajectories allows the algorithm to run at $5Hz$ using our on-board computer, the Odroid-XU4.

Using the geometry in Figures 14a & 14b, we can easily compute the trim trajectory that travels from the aircraft's current location, $\mathbf{p}_i^0(x^0, y^0, z^0)$, to the final position at the edge of the depth camera's FOV, $\mathbf{p}_i^f(x^f, y^f, z^f)$, which are the red dots in Figure 13.

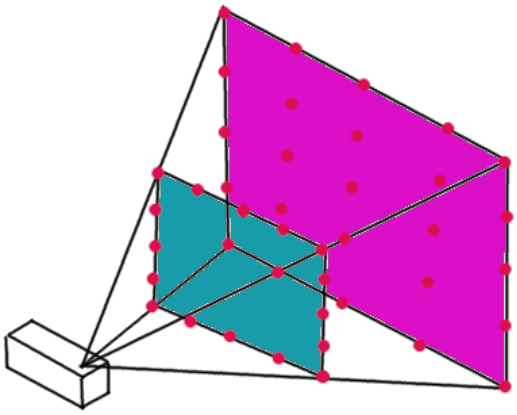
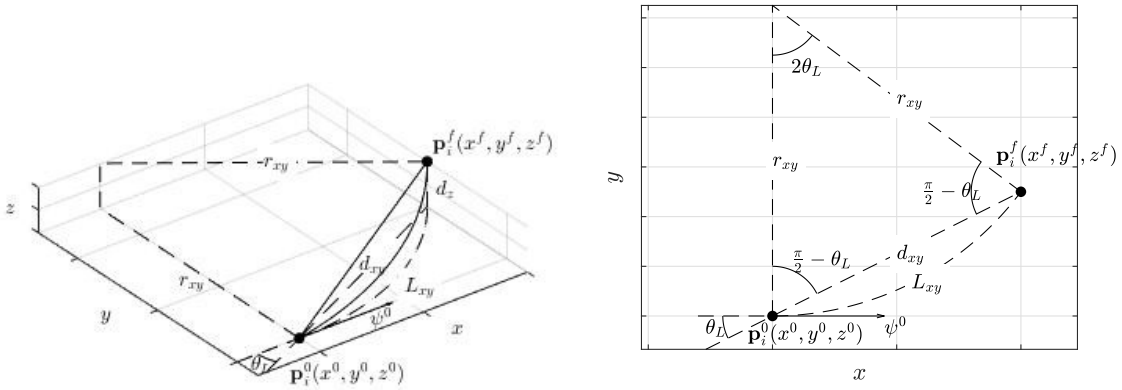


Figure 13. Final positions of one motion planning time-step



(a) 3D view **(b)** Top-View

Figure 14. 3D circular arc defining trim primitive geometry

Since the trim primitives have no sideslip, by assuming the wind is small we can assume that the aircraft's yaw, ψ, is the same as the aircraft's heading. With this assumption we can solve for the trim primitive defined by its yaw rate, $\dot{\psi}^{ref}$, climb/descent rate, $v_{i,z}^{ref}$, and coasting time, Δt.

The magnitude of the vector going from the aircraft to the final position, projected on the horizontal plane, is

$$d_{xy} = \sqrt{(x^f - x^0)^2 + (y^f - y^0)^2}, \tag{8}$$

and this vector projected on to the vertical plane is

$$d_z = z^f - z^0. \tag{9}$$

The angle between aircraft's current yaw, ψ^0, and vector from $\mathbf{p}_i^0$ to $\mathbf{p}_i^f$ projected onto the horizontal plane, is

$$\theta_L = \text{wrap2pi}(\arctan(\frac{y^f - y^0}{x^f - x^0}) - \psi^0), \tag{10}$$

where the wrap2pi function keeps the angle in parenthesis between $-\pi$ and π. Referring to Figure 14b, we can obtain the turn radius in the horizontal plane, r_{xy}, from

$$r_{xy} \sin(\frac{1}{2} 2\theta_L) = \frac{1}{2} d_{xy} \tag{11}$$

which reduces to

$$r_{xy} = \frac{d_{xy}}{2\sin\theta_L}. \tag{12}$$

We can then compute the arclength projected onto the horizontal plane from

$$L_{xy} = \int_0^{2\theta_L} r_{xy}\, d\Theta = r_{xy}(2\theta_L) = \frac{\theta_L d_{xy}}{\sin\theta_L}. \tag{13}$$

The coasting time is computed as

$$\Delta t = \frac{L_{xy}}{v_{i,xy}^{ref}} = \frac{d_z}{v_{i,z}^{ref}}, \tag{14}$$

where aircraft reference speed in the horizontal plane is denoted by $v_{i,xy}^{ref}$ and the reference climb/descent rate as $v_{i,z}^{ref}$. By definition,

$$||\mathbf{v}_i^{ref}||^2 = {v_{i,xy}^{ref}}^2 + {v_{i,z}^{ref}}^2, \tag{15}$$

and since $||\mathbf{v}_i^{ref}||$ is a constant known quantity, we can obtain the speed in the horizontal plane as

$$v_{i,xy}^{ref} = \frac{||\mathbf{v}_i^{ref}||}{\sqrt{1 + \frac{d_z^2}{L_{xy}^2}}}. \tag{16}$$

Now that $v_{i,xy}^{ref}$ is solved for, we can obtain the coasting time, Δt, from Eq. 14. We can then compute the climb/descent rate from

$$v_{i,z}^{ref} = \frac{d_z}{\Delta t}. \tag{17}$$

The yaw rate can then be computed as

$$\dot{\psi}^{ref} = \frac{v_{i,xy}^{ref}}{r_{xy}}. \tag{18}$$

The computed yaw rate and climb/descent rate are then rounded to the nearest $10^\circ s^{-1}$ and $1ms^{-1}$ to find the closest trim primitive in the library. For each trim primitive the associated roll, pitch, angular and linear body-frame velocity are all constant, and stored in memory. The reference yaw and position can be easily computed at a time t since the start of the maneuver. The aircraft's reference yaw at time t, $\psi^{ref}(t)$, is calculated by:

$$\psi^{ref}(t) = \psi^0 + \dot{\psi}^{ref} t. \tag{19}$$

The reference position at time t, $\mathbf{p}_i^{ref}(t) = (x^{ref}(t), y^{ref}(t), z^{ref}(t))$, is calculated by:

$$\begin{aligned}
x^{ref}(t) &= \begin{cases} x^0 + v_{i,xy}^{ref}\, t\cos\psi^0, & \text{if } \dot{\psi}^{ref} = 0 \\ x^0 + \frac{v_{i,xy}^{ref}}{\dot{\psi}^{ref}}(\sin(\psi^0 + \dot{\psi}^{ref} t) - \sin\psi^0), & \text{otherwise} \end{cases} \\
y^{ref}(t) &= \begin{cases} y^0 + v_{i,xy}^{ref}\, t\sin\psi^0, & \text{if } \dot{\psi}^{ref} = 0 \\ y^0 + \frac{-v_{i,xy}^{ref}}{\dot{\psi}^{ref}}(\cos(\psi^0 + \dot{\psi}^{ref} t) - \cos\psi^0), & \text{otherwise} \end{cases} \\
z^{ref}(t) &= z^0 + v_{i,z}^{ref}\, t
\end{aligned} \tag{20}$$

The ability to quickly compute the trim primitive is a major advantage, since it allows us to only use computational resources on evaluating trajectories that (a) remain within the FOV, and (b) represent the entirety of the FOV. While we only evaluate up to 41 trajectories in at given time-step, they are (up to) 41 trajectories selected from a much larger trajectory library, and increasing the

size of the trajectory library does not increase the computational burden on the algorithm. If we weren't using trim primitives, and therefore couldn't compute the trajectory which leads to a final position, we would have to evaluate trajectories regardless of where they end up. This would lead to evaluating trajectories that potentially: don't span the entirety of the FOV, don't end at the edge of the FOV, or end outside the FOV.

Another benefit of using trim primitives is their constant roll, pitch, and turn rate is advantageous when switching maneuvers frequently. In our preliminary work in (Bulka and Nahon, 2019b), we only used finite-time primitives. In order to keep the size of the library tractable, each maneuver started from level flight (zero roll), similar to (Barry, 2016). In order to effectively turn, these reference trajectories must be executed for a significant amount of time to allow the reference trajectory to go from level to banking, and then turn while banking. If the aircraft was initially banked, it would first roll towards level flight, and then back towards the desired bank angle, ultimately slowing down the turn. By using constant trim primitives, the need to start from level flight is avoided, and the aircraft is always directly commanded to the specified bank angle. This avoids the problem of initially turning the wrong way, and removes the need to execute the trajectory for a significant amount of time, which puts no limitation on the frequency at which reference trajectories are updated. This higher frequency enables the aircraft to fly through more cluttered environments at higher speeds.

4.3.2. Distance to Obstacles

The minimum distance from each primitive being evaluated to the point cloud representation of obstacles is used to determine if the primitive is collision-free, and to determine the cost associated with each primitive. The point cloud is made up of position vectors from the camera to the points in the point cloud, and are resolved in a coordinate frame fixed to the depth camera, $\mathcal{F}_c$. On the other hand, the positions along the trajectory are with respect to the NED origin, and are resolved in the inertial frame. In order to compute the distance from a point on a trajectory to the point cloud, we need to translate and rotate the positions along the trajectory. We can perform this transformation at a time t along the trajectory as follows:

$$\mathbf{p}_c^{ref/0}(t) = \mathbf{C}_{cb}\mathbf{C}_{bi}(\mathbf{p}_i^{ref}(t) - \mathbf{p}_i^0) \tag{21}$$

where $\mathbf{p}_c^{ref/0}(t)$ is the reference position with respect to aircraft, resolved in the camera frame, t seconds along the trajectory. The position t seconds along the trajectory, with respect to the origin and resolved in the inertial frame, $\mathbf{p}_i^{ref}(t)$, can be computed using Eq. (20). We assume that the camera is positioned at the aircraft's center of mass, and thus $\mathbf{p}_c^{ref/camera}(t) = \mathbf{p}_c^{ref/0}(t)$.

Every $25cm$ along the trajectory we compute the distance from that position to the closest point in the point cloud, which is stored in an octree for efficient distance computation. The minimum distance obtained is the distance to obstacle, d^{obs}. If this distance is less than half the aircraft's wingspan, the trajectory will result in a collision and is discarded. While the true wingspan of our vehicle is $0.86m$, we set the wingspan during collision checking to $2m$, to give a buffer for potential camera misalignment, state estimation errors, control errors, or modeling errors. Increasing the buffer size reduces the risk of collision in case of these aforementioned errors, but comes at the cost of potentially executing an emergency hover when the aircraft could have actually maneuvered around an obstacle.

4.3.3. Trajectory Cost

A cost function is constructed for each trajectory that specifies how competing objectives are combined. The cost of a potential trajectory (c) grows when being near obstacles (c^{obs}), the aircraft is steered away from the goal in both the horizontal (c^h) and vertical (c^v) plane, and when the trajectory selected significantly differs than the trajectory it's currently on (c^{diff}). The total cost associated with the potential trajectory is the sum of all these sub-costs:

$$c = c^{obs} + c^h + c^v + c^{diff}. \tag{22}$$

The cost for being near obstacles, c^{obs} is computed as follows:

$$c^{obs} = \begin{cases} -d^{obs} w^{obs}, & \text{if } d^{obs} < 10 \\ -10 w^{obs}, & \text{otherwise} \end{cases} \tag{23}$$

where w^{obs} is a positive weight and d^{obs} is the minimum distance from the potential trajectory to the point cloud. The rationale behind this cost function is the cost is reduced linearly the further the trajectory is from an obstacle, until a critical distance of $10m$ at which the trajectory is so far from an obstacle that it becomes irrelevant how far away it is. In our preliminary work in (Bulka and Nahon, 2019b), we simply discarded trajectories that result in a collision, but didn't add the distance to the obstacles within the cost function. Adding this distance into the cost function significantly improves performance because trajectories that are predicted to be collision-free but close to obstacles could actually result in a collision due to modeling inaccuracies and control errors. With trajectories being penalized for being near obstacles, the trajectories that end up getting selected are further away from obstacles.

The cost due to being steered away from the goal in the horizontal plane, c^h, is computed as follows:

$$c^h = w^h ||\text{wrap2pi}(\tan^{-1}(\frac{y^g - y^0}{x^g - x^0}) - \psi^f)|| \tag{24}$$

where the goal position is denoted $\mathbf{p}_i^g(x^g, y^g, z^g)$. The cost is proportional to the angle between the straight line from the aircraft to the goal, and the final yaw angle of the trajectory, which is depicted in Figure 15.

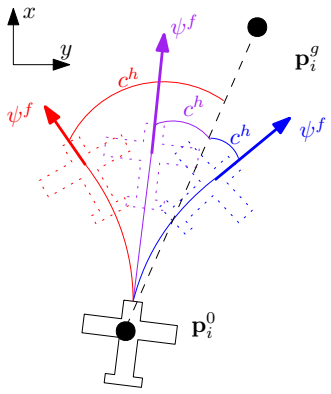


Figure 15. Depiction of horizontal cost

The cost due to being steered away from the goal in the vertical plane, c^v, is proportional to the distance from the altitude of the aircraft at the end of the trajectory and the altitude of the goal, and is computed as follows:

$$c^v = w^v ||z^g - z^f||. \tag{25}$$

The last portion of the cost comes from switching trajectories with large differences in yaw rate, because frequent switching reduces the efficiency of flight, the stability of the aircraft, and the ability for the aircraft to track the desired trajectory. In order to reduce the number of switches, as well the extremity of each switch, we add a penalty when switching to a trajectory with a different yaw rate. The cost is linearly proportional to the difference between the yaw rate of the trajectory being evaluated and the yaw rate of the primitive the aircraft is currently tracking:

$$c^{diff} = w^{diff} ||(\dot{\psi}^{ref} - \dot{\psi}^{ref,current})||. \tag{26}$$

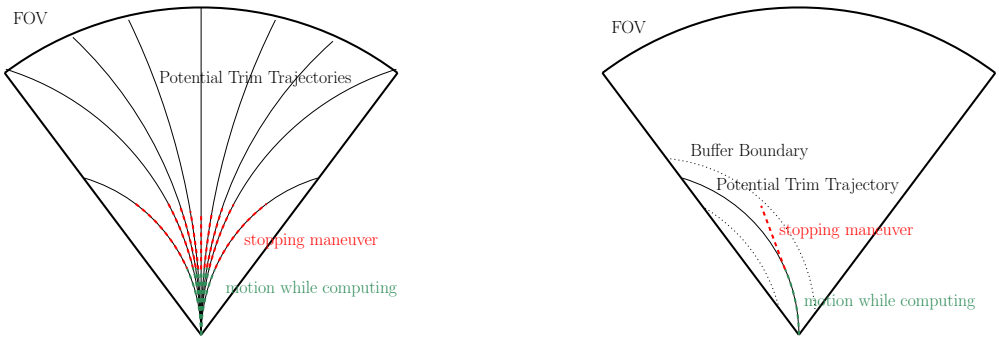
Ultimately, the trajectory with lowest cost, c, is sent to the controller to track.

4.3.4. Safety Guarantees

The obstacle-avoidance algorithm must ensure the aircraft never enters an inevitable collision state. In (Lopez and How, 2017a; Liu et al., 2016), safety is guaranteed with a quadrotor by restricting motion primitives to remain within the depth sensor's FOV in addition to ensuring a stop maneuver is possible if no collision-free path is found. In (Tijmons et al., 2017), safety is guaranteed by ensuring the flapping-wing MAV can always fly in a circle within the FOV.

A major advantage to developing collision avoidance strategies with agile, fixed-wing, aircraft, as opposed to traditional aircraft, is the ability to come to a complete stop while hovering. This allows us to guarantee safety using methods developed for rotorcraft. Since only collision-free motion primitives are selected, if we can ensure that at the next motion planning time-step, the aircraft can execute a stopping maneuver that remains within the current time-step's FOV, and travels along the current time-step's collision-free path, one can guarantee the aircraft can always stop to avoid a collision. This is demonstrated in Figure 16a, where the green dash represents the aircraft motion while the motion plan is being computed, and the red dash represents the motion if the aircraft was to execute a stopping maneuver on the following time-step, and that stopping maneuver followed the path of the previous time-step's trim primitive.

While it is potentially possible to generate a stopping maneuver associated with each trim primitive, it is much simpler to only have one stopping maneuver, such as the Cruise-to-Hover maneuver described in Sec. 4.2.2. During the collision checking step a buffer distance is added, so a trajectory will only be selected if the space occupied by the buffer is also collision-free. If the single cruise-to-hover maneuver remains close enough to any potential trim primitive, such that it remains in the buffer, the same logic as before still applies. This is the case for even fairly small buffer distances, since the duration of the cruise-to-hover is short, and the aircraft won't move too much laterally in a very short period of time. This is demonstrated in Figure 16b, where the cruise-to-hover maneuver does not perfectly follow the previous trim trajectory, but remains within the buffer distance, and within the FOV, guaranteeing collision-free flight. While we only show this for one particular primitive for simplicity, this must hold for all potential trim primitives.



(a) Multiple stopping maneuvers along trim trajectories **(b)** Single stopping maneuver: the Cruise-to-Hover

Figure 16. Top-down view of motion primitives within FOV

These safety guarantees are only valid assuming there is perfect sensing, perfect control, the obstacles are static, and the initial conditions are such that a cruise-to-hover is collision-free. Furthermore, this analysis neglects the transient effects of switching maneuvers.

We justify neglecting these transient effects in this analysis using our simulation and flight data. First, we simulate the aircraft using the $9ms^{-1}$ trajectory library, starting with the reference trim primitive of $\dot{\psi}^{ref} = 100°s^{-1}$ and $v_{i,z}^{ref} = 0ms^{-1}$. Every $10s$, we switch the reference trim primitive, keeping $v_{i,z}^{ref} = 0ms^{-1}$, and changing $\dot{\psi}^{ref}$ to $-90°s^{-1}$, then to $90°s^{-1}$, then to $-80°s^{-1}$, then to $80°s^{-1}$, then to $-70 \circ s^{-1}$, and continue this pattern until $0°s^{-1}$. The results from this simulation are shown in Figure 17, where we plot the roll (actual and reference), body-axis z angular velocity

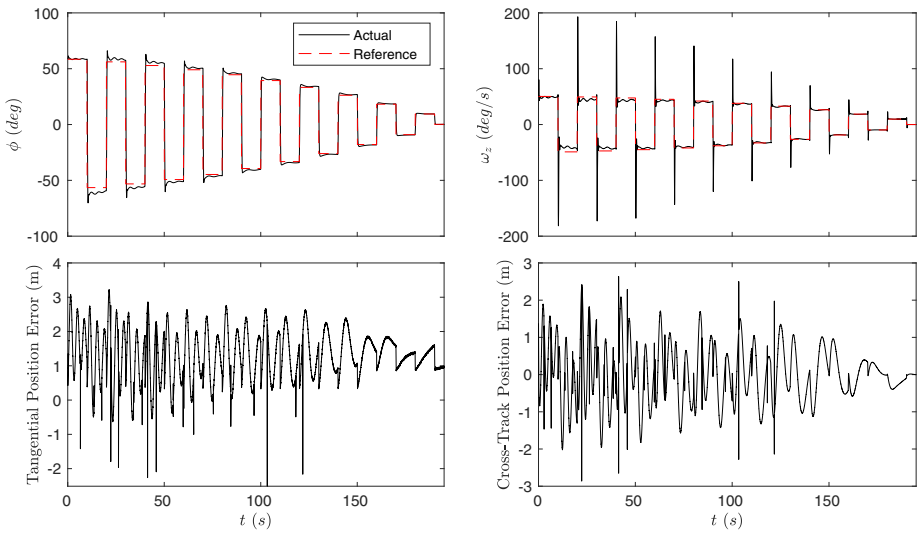


Figure 17. Transient effects of switching trim primitives

(actual and reference), as well as the position error, both tangent and perpendicular to the reference path.

We can see these transient effects lead to position errors that are small enough that they could be absorbed into the buffer distance in the collision-checking step, which justifies neglecting these transients in the analysis. Furthermore, as the magnitude of the reference yaw rate switch decreases (increasing time), these errors reduce in magnitude. Due to the term in the cost function that penalizes changes in reference yaw rate, the maneuver switches during flight tend to be smaller in magnitude (Bulka, 2021).

A similar argument can be made for a mismatch in initial conditions for the cruise-to-hover maneuver. We refer to simulated flight data presented later in Figure 21. We can see that at $t = 8s$, the aircraft is executing a banked turn when the cruise-to-hover maneuver is initiated, yet the aircraft remains close enough to the reference trajectory that it would remain within a reasonably sized buffer distance.

We also refer to experimental flight data presented later in Figures 46 & 50. In Figure 46, at $t = 18s$, the aircraft initiates the cruise-to-hover from level flight, and in Figure 50, at $t = 7s$, the aircraft initiates the cruise-to-hover from a large roll angle. In both cases, the errors in position are similar, but are both larger than in the simulation example, which likely concludes that the initial condition mismatch is less significant to the tracking performance in comparison to other disturbances such as wind.

4.3.5. Computational Breakdown

The computation of the trajectory selection is broken down into three components: obtaining the candidate trim primitive, computing the minimum distance between the candidate trim primitive and all of the points in the point cloud, and assigning a cost to the candidate trim primitive. Since obtaining the candidate trim primitive and assigning the cost are accomplished with a few algebraic computations, we expect the majority of the computational resources to be allocated to computing the minimum distance to the point cloud. We verify this using our simulation presented in Sec. 5 and our experimental test platform presented in Sec. 6.

In each simulation environment, we find that more than 99% of the computation time is used to compute the minimum distance to the point cloud. Since computing the minimum distance to the point cloud is a function of the point cloud itself, the computation time varies with the environment. In the case of flying in an open sky (with an empty point cloud), the computational time will

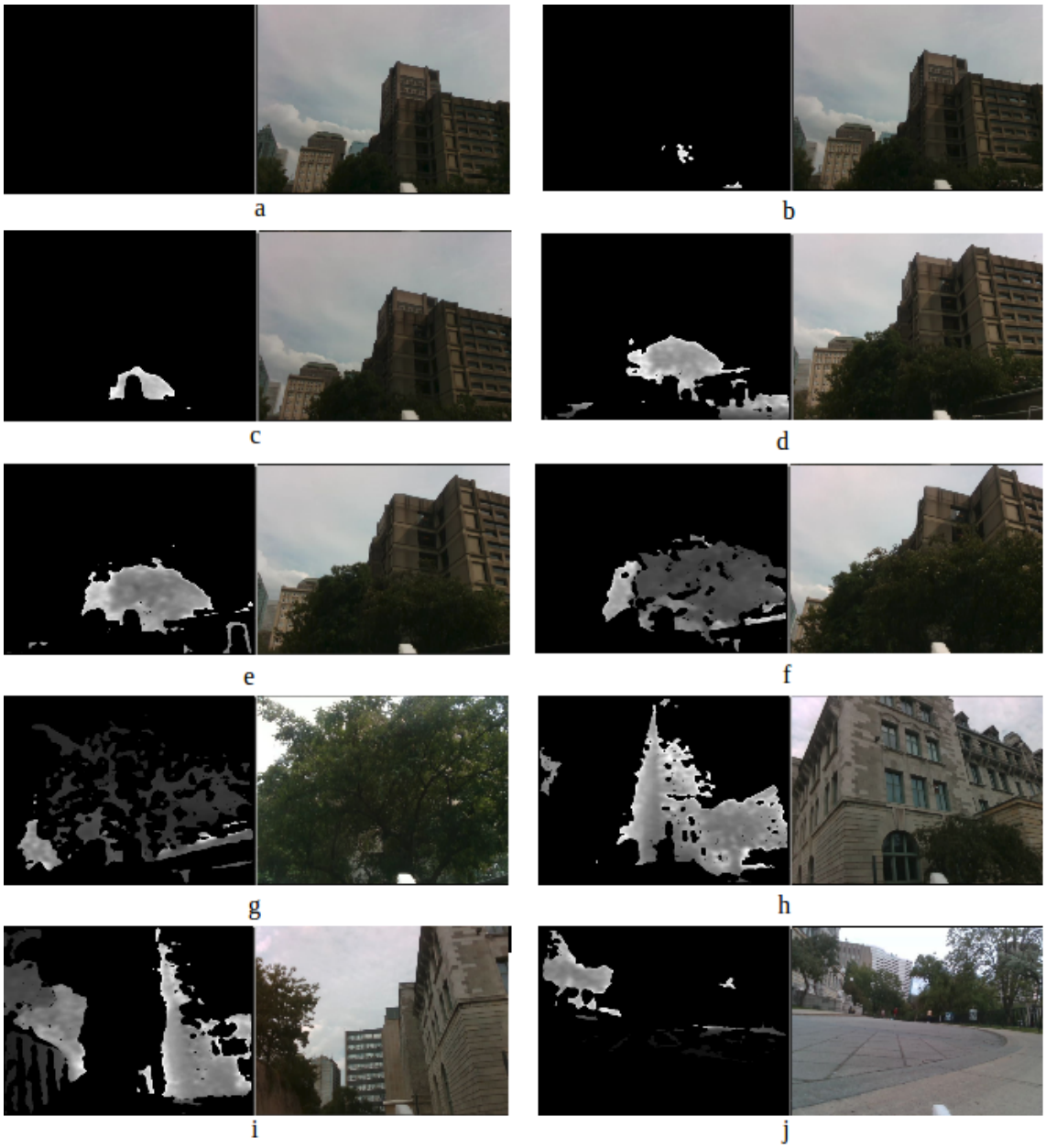


Figure 18. Computational Breakdown Example Images

drastically reduce, and the percentage of time allocated towards computing the minimum distance to the point cloud will decrease. However, since the need for an obstacle-avoidance algorithm is only relevant when there are obstacles present and the point cloud is not empty, we design our algorithm assuming the computation time will be that of flying in a cluttered environment.

We find similar results when running the obstacle-avoidance on our experimental test platform. The test platform uses an Intel Realsense D435 to detect obstacles and the obstacle-avoidance algorithm is running on an ODROID-XU4 companion computer. In Figure 18, we show the depth and color image of various scenarios, and show the computational breakdown of the obstacle-avoidance algorithm of the corresponding instances in Table. 2. In the depth image, black pixels corresponds to nothing in the point cloud, and grey-scale pixels correspond to points in the point cloud where the

Table 2. Computational Breakdown of Obstacle-Avoidance Algorithm Corresponding to Figure 18

Image	Compute Trim Trajectory (ms)	Compute Distance to Obstacle (ms)	Compute Trajectory Cost (ms)	Total (ms)	Compute Trim Trajectory %	Compute Distance to Obstacle %	Compute Trajectory Cost %
a	0.21	0.06	0.07	0.34	62.56	16.38	21.06
b	0.51	65.93	0.23	66.68	0.77	98.88	0.35
c	0.35	96.38	0.15	96.88	0.36	99.48	0.16
d	0.65	155.01	0.23	155.89	0.42	99.44	0.15
e	0.32	117.99	0.11	118.42	0.27	99.64	0.09
f	0.30	94.47	0.06	94.82	0.31	99.62	0.06
g	0.30	66.19	0.02	66.51	0.45	99.52	0.03
h	0.28	72.78	0.04	73.09	0.38	99.58	0.05
i	0.31	142.63	0.09	143.04	0.22	99.72	0.06
j	0.33	4.02	0.00	4.35	7.49	92.51	0.00

darker the pixel the closer the obstacle. Looking at Figure 18, we can see that there is an empty point cloud in (a) because all the obstacles are out of the camera's range. In this instance, the computation time is very small, and percentage of time dedicated computing the minimum distance between the candidate trim primitives and all of the points in the point cloud is correspondingly small. As the aircraft moves closer to the tree in images (b) - (g), and the tree always remains within the depth camera's range, the percentage of computation time corresponding to computing the distance to obstacles remains larger than 98% of the total computation time. The total computation time varies depending on the environment. The number of points in the point cloud, as well as location of each point will play a role in the computation time of computing the minimum distance to obstacles. The first reason for this is that we use an octree structure to find the nearest point in the point cloud to a point along the candidate trajectory. Using this structure is much more efficient than simply searching along every point in the point cloud, but also causes the computation time to depend not only on the number of points, but also their locations. The second reason is that, when we search for the nearest points along the trajectory, if a collision would occur early on, the computation time will decrease because the rest of the trajectory is not evaluated. We can see the computation time increases as the aircraft gets closer to the tree, and more points appear in the point cloud in (b), (c), and (d). In images (e), (f), (g), the computation time starts to decrease, because collisions are being detected early on. An extreme example of this is in (j), where the aircraft is placed on the ground, and straight ahead sees the ground very close to itself. This causes every candidate trajectory to result in a collision early on when computing the minimum distance to obstacles, and therefore the total computation time drastically decreases, and the percentage of time dedicated to computing this distance also decreases.

4.3.6. Controller Integration

The obstacle-avoidance algorithm sends the controller a reference motion primitive, as well as the pose of the aircraft at the time the reference primitive was selected at a rate of $5Hz$. The controller runs at $200Hz$ and extracts the reference states from the motion primitive sent from the obstacle-avoidance.

For the trim primitives, the reference roll and pitch angles, as well as reference translational and angular velocities, can be directly looked up in memory. The reference yaw and position can be computed using Eqs. (19 - 20), where $\mathbf{p}_i^0$ and ψ^0, come from the pose sent along with the trajectory, and t is the difference between the current time, and time stamp associated with $\mathbf{p}_i^0$ and ψ^0. Since the position and yaw reference is continuously being reset to its current value, there is no position or yaw error at the beginning of a new trajectory. We found it advantageous to add a tenth of a second to t, which pushes the reference state further downstream the reference trajectory, resulting in position and yaw errors that cause a control action at the beginning of the maneuver. For agile

primitives, all of the reference states are stored in memory, but the position reference is rotated by ψ^0 and translated by $\mathbf{p}_i^0$, and yaw reference is offset by ψ^0.

We validate this obstacle-avoidance methodology in simulation in Sec. 5, prior to implementing on hardware (Sec. 6) and conducting flight tests (Sec. 7).

5. Simulation

We initially utilize a simulation environment prior to testing on an actual aircraft. In simulation, we can quickly and easily tune parameter values; without a risk of damage to the aircraft. After converging on a set of parameter values, we can validate our obstacle-avoidance methodology in simulation. An advantage of validating in simulation in comparison to experiment is having control over many external factors that are uncontrolled in experiment. These external factors include the wind conditions, the sunlight, and the layout of obstacles in the environment. In addition to fine-tuning parameter values, we use our simulation environment to find the limits of the types of obstacle environments our algorithm succeeds in when there is an absence of wind and sensor noise.

As previously mentioned in Sec. 4, we limit the scope of this work to reactive obstacle-avoidance. Since all of the high-level planning is based on instantaneous point cloud data, we suspect the aircraft will fail to reach the goal in certain environments. While the primary objective of the obstacle-avoidance algorithm is to avoid obstacles, and reaching the goal is secondary, it is still useful to understand which types of environments the aircraft could find the goal location without a SLAM implementation and global motion planner. Examining this in simulation has two main advantages. First, it is beneficial to understand the algorithm limitations prior to testing on an actual vehicle. Second, obtaining permission to fly the aircraft is difficult, and we therefore have little choice over the type of environments we can experimentally fly at. The simulation gives us the ability to create different environments, and we can then test out the algorithm in any environment we create.

To realistically validate the obstacle-avoidance algorithm in simulation, the simulation environment must include the aircraft dynamics, feedback controller, obstacles, and a depth camera. We found it easiest to simulate all of these components using Gazebo (Koenig and Howard, 2004), where we can accurately simulate our experimental platform described in Sec. 6. The aircraft dynamics are simulated using a custom plug-in that corresponds to the aerodynamics, thruster, and slipstream modeling presented in Sec. 3.1. For obstacle detection, we simulate the RealSense D435 depth camera used on our experimental platform (Sec. 6) in Gazebo by modifying the parameters in the existing Kinect depth camera plug-in.

We evaluate the obstacle-avoidance methodology in four environments, each with different types of obstacles and of varying difficulty, that we believe are representative of potential obstacle fields that an aircraft would need to fly through for various applications. While the methodology presented is in three dimensions and the aircraft can climb and descend to avoid obstacles, we found it clearer to display results using the horizontal plane, and therefore use very tall obstacles in each simulation environment, which forces the aircraft to go around, and not over them. In each test environment, we run 10 simulations that use a $9ms^{-1}$ trajectory library, and 10 simulations that use a $13ms^{-1}$ trajectory library. For each test environment, we select 10 positions to initialize the aircraft from zero velocity, zero angular velocity, 0° roll, 0° pitch, and 90° yaw. We allow one second to pass to allow the aircraft to reach cruise conditions, and show the simulation results from then. For each test environment, we use the same 10 initial positions for both trajectory library speeds. Each simulation is executed until either the aircraft reaches the goal and hovers, the aircraft executes an emergency hover, or the aircraft collides with an obstacle. For every simulation run, the goal is located at $(x, y, z) = (30, 0, -10)m$ in the NED frame, and if the aircraft enters within $5m$ of this position it is commanded to hover.

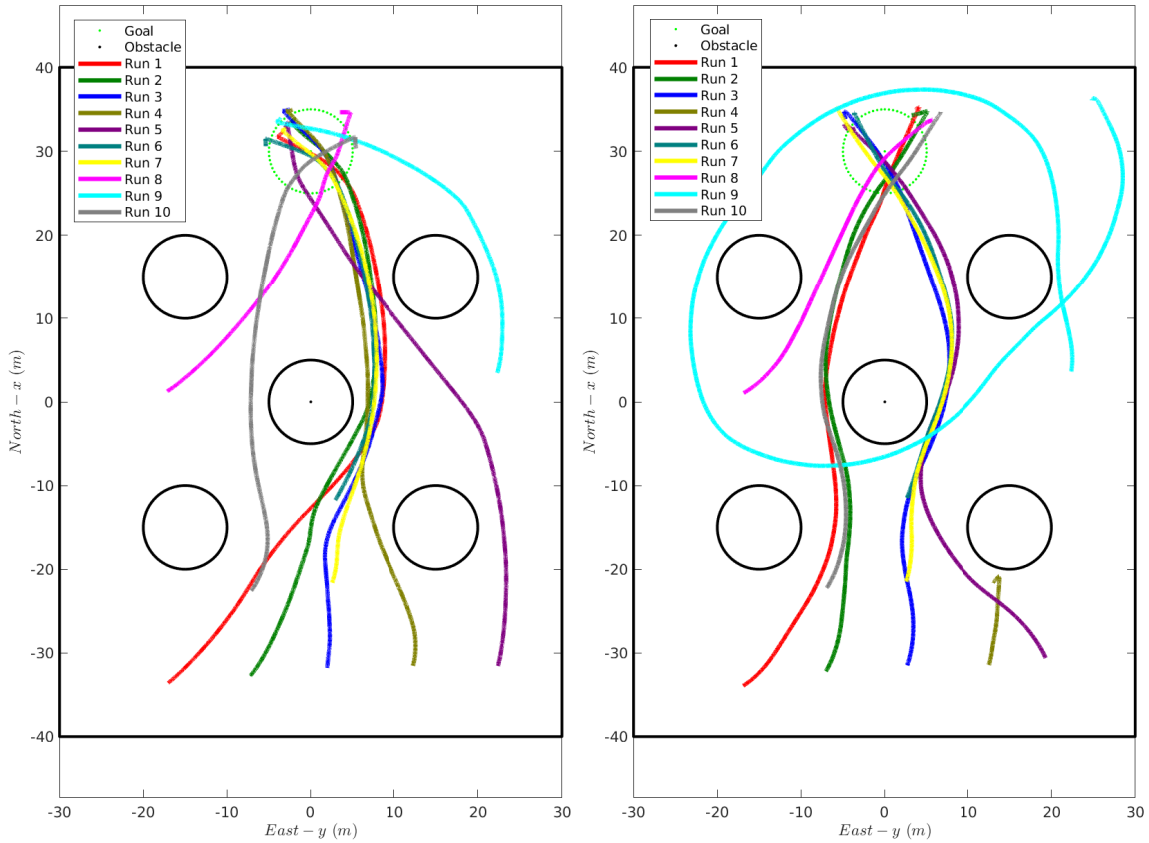
In these simulation runs, we end the simulation once the aircraft performs an emergency hover, since the aircraft cannot perceive the environment with its camera pointing upward, and has no memory of the environment it was just flying through. If a SLAM algorithm was implemented, the

aircraft would build a map of the environment while flying, and if an emergency hover is needed, the aircraft could compute a motion plan using that map while hovering, and continue flying to the goal.

5.1. Environment 1

We first test the obstacle-avoidance methodology in a relatively easy environment with only a few obstacles and large gaps between them. This environment consists of 5 tall poles with a $5m$ radius which are enclosed in a $60m \times 80m$ rectangle. A top-down view of the environment and flight trajectories is shown for the trajectories using the $9ms^{-1}$ trajectory library in Figure 19a and for the trajectories using the $13ms^{-1}$ trajectory library in Figure 19b. As shown in Figure 19a, the aircraft reaches the goal for all 10 runs using the $9ms^{-1}$ trajectory library.

Figure 19b shows a top-down view of the environment using the $13ms^{-1}$ library. The aircraft reaches the goal 8 out 10 times. In the two runs where the aircraft does not reach the goal, collisions are avoided with the emergency hover. In run 4, the aircraft simply does not have enough space to maneuver around the pole at $13ms^{-1}$. In run 9, the aircraft is not able to turn sharply enough to enter the goal region, and ends up looping back through poles, and ultimately corners itself and performs an emergency hover. The aircraft reaches the goal more often at lower speed because it can turn with a smaller turning radius. Not only is the turning radius larger when flying faster with the same heading rate, the maximum heading rate is decreased at higher speed, due to throttle saturation, as mentioned in Sec. 4.2.



(a) $9ms^{-1}$ **(b)** $13ms^{-1}$

Figure 19. Environment 1

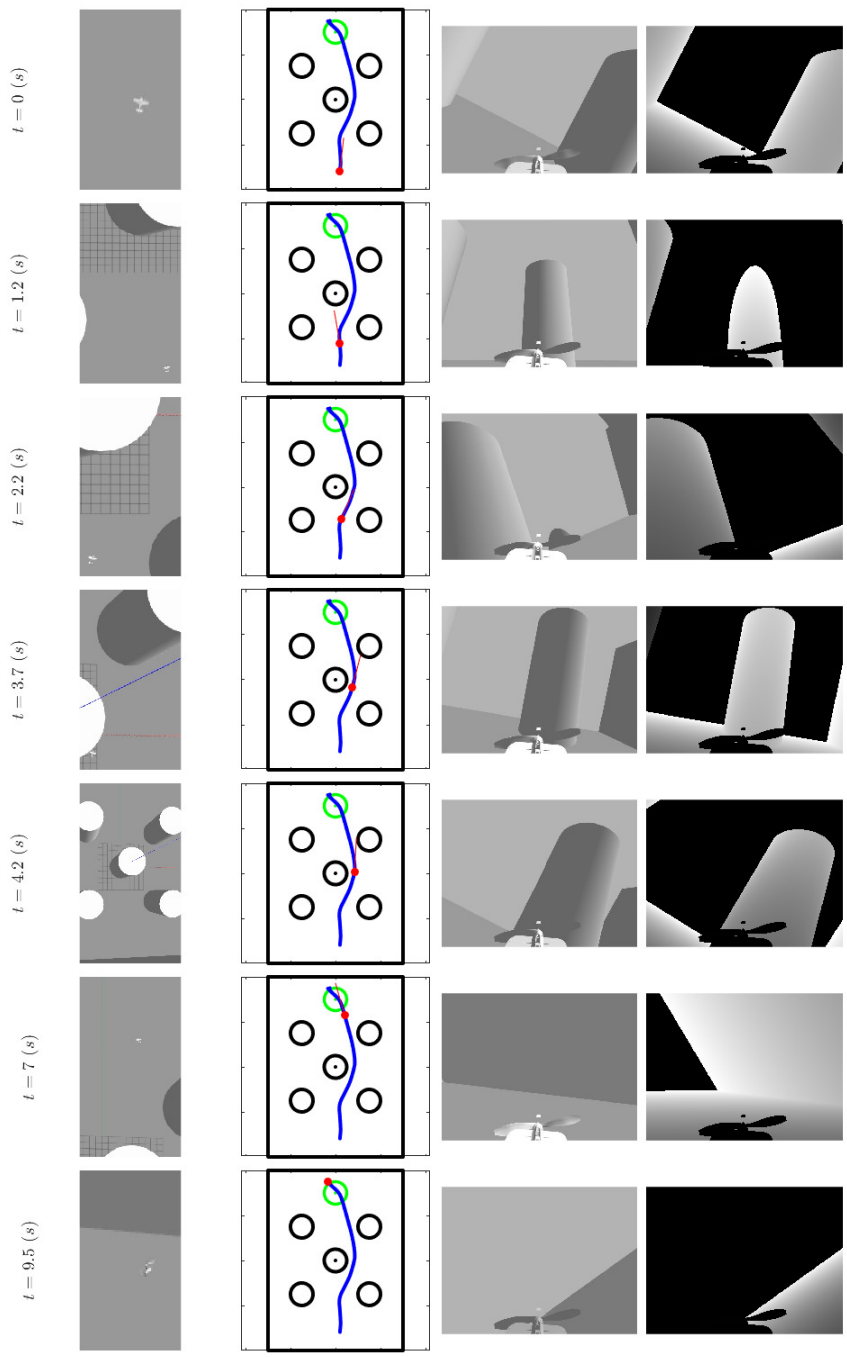


Figure 20. Environment 1 at $9ms^{-1}$, Run 3: aerial image, flight trajectory, on-board color image, and on-board depth image

We examine run 3 of the $9ms^{-1}$ trajectory library in more detail using Figures 20 & 21. At selected times throughout the flight, Figure 20 shows from left to right: a 3rd-person aerial image; the location and yaw (red dot and red line) with respect to the top-down view of the trajectory and environment; the on-board color image; and the on-board depth image. In the depth image, black corresponds to nothing being detected, and obstacles are represented in gray-scale where the darker pixels correspond to closer obstacles. In Figure 21 we plot the aircraft states, references, and

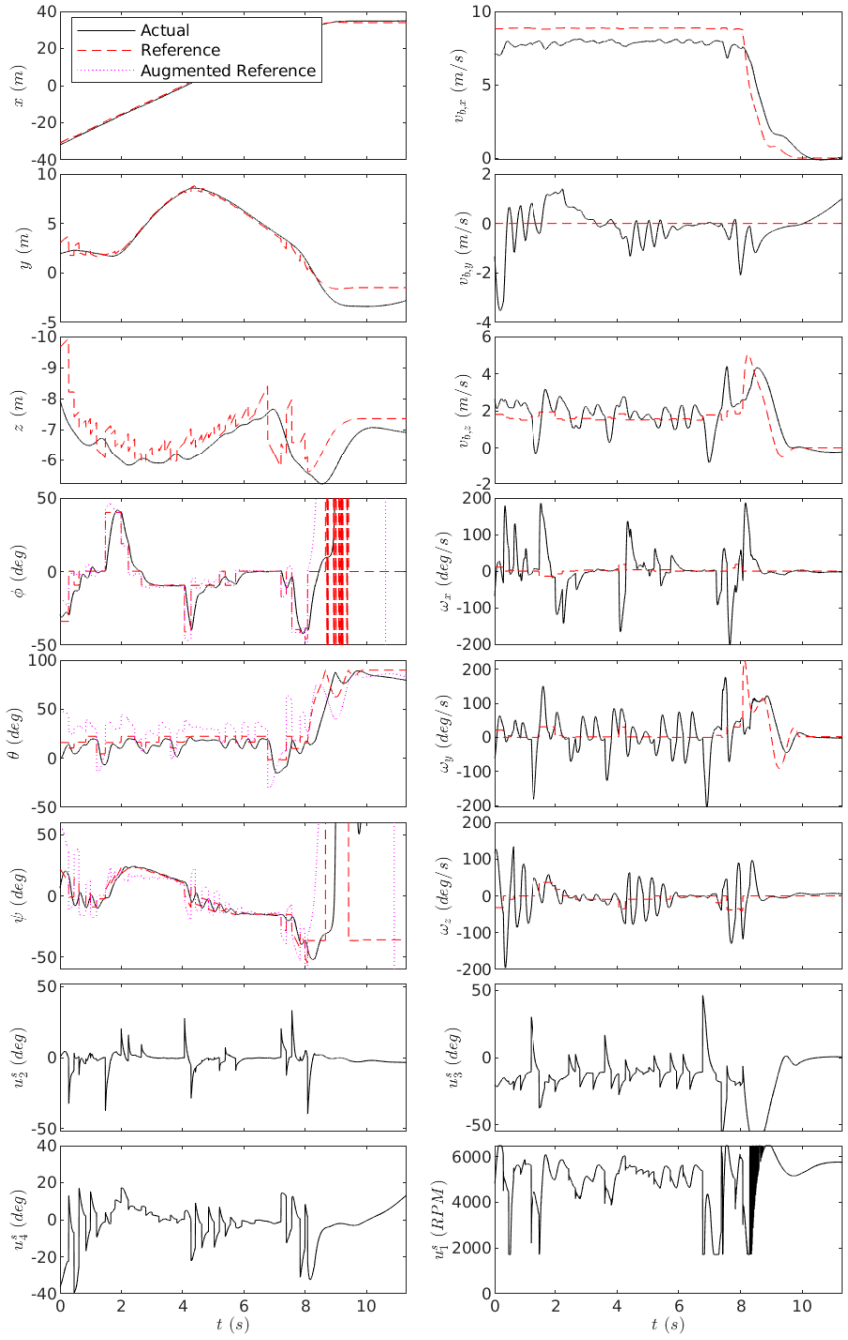


Figure 21. Environment 1 at $9ms^{-1}$, Run 3: state estimates, reference states, and control inputs

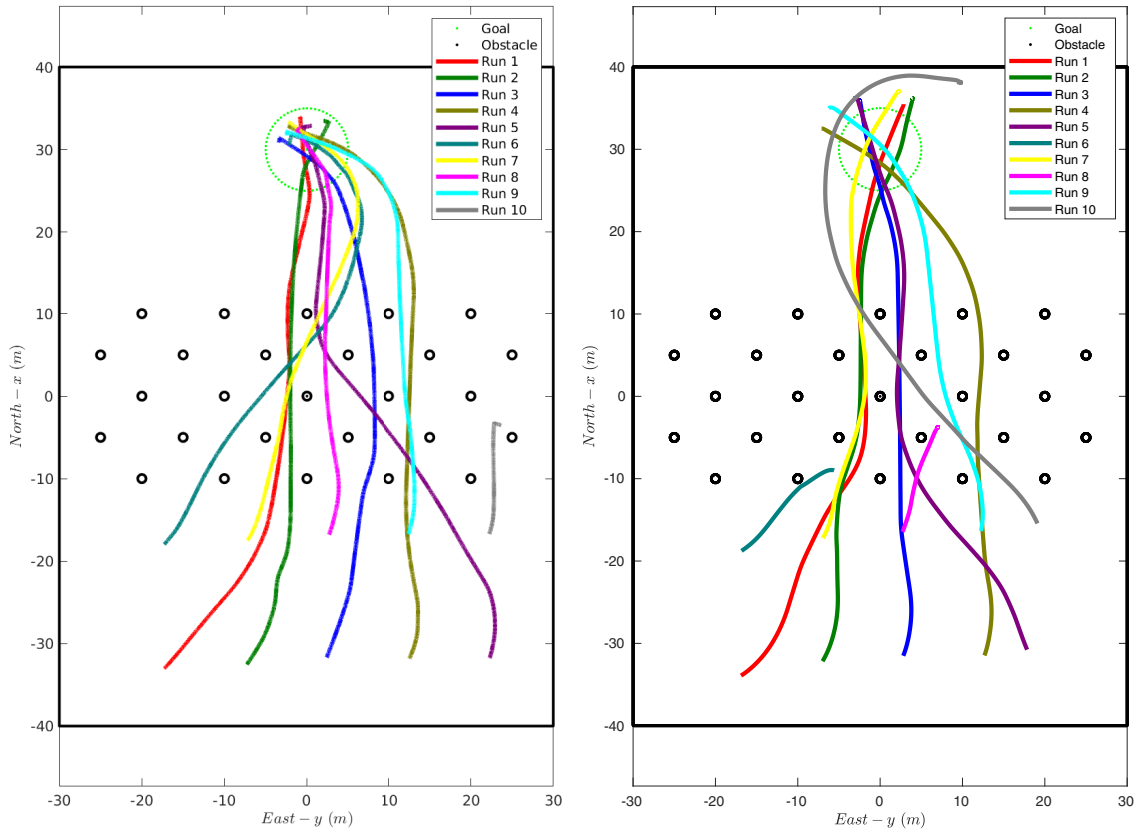
control inputs (throttle (u_1^s), aileron (u_2^s), elevator (u_3^s), rudder (u_4^s)) for the duration of the flight. For clarity, we only show the roll and yaw angles from $-50°$ to $50°$ and $-60°$ to $60°$ respectively, as the motion in cruising flight remains within these limits. They exceed these limits when hovering, but roll and yaw become unintuitive variables when the pitch is near $90°$.

As we can see in Figure 20, at the start of the trajectory the aircraft sees the pole at $(x, y) = (-15, 15)$ and turns slightly left. The aircraft then continues straight, and at $t = 1.2s$ the aircraft sees the pole at $(x, y) = (0, 0)$. The aircraft subsequently banks right, as seen in the large positive

roll and reference roll angle shown in Figure 21. By $t = 3.7s$ the pole at $(x, y) = (0, 0)$ is to the left and mostly out of sight of the aircraft. However, the aircraft now sees the pole at $(x, y) = (15, 15)$ directly in front. To avoid this pole, and head towards the goal, the aircraft turns left. At $t = 4.2s$ the aircraft is in the middle of the left bank turn, as seen by the negative roll and reference roll. The aircraft continues to fly straight, and by $t = 7s$ the aircraft has flown past all of the obstacles. The aircraft enters the goal region at $t = 8s$ and executes the cruise-to-hover maneuver, and holds the hover for the remainder of the flight.

5.2. Environment 2

Our second test environment is more challenging, containing an area with densely spaced tall thin poles, resembling a forest. The environment consists 5 rows of tall poles spaced $10m$ apart in the East direction. Alternating rows are shifted by $5m$ in the East/West direction and spaced $5m$ apart in the North/South direction. Cumulatively there are 27 poles, each with a $0.5m$ radius, and enclosed in a $60m \times 80m$ rectangle. A top-down view of the environment and flight trajectories is shown for the trajectories using the $9ms^{-1}$ trajectory library in Figure 22a, and for the trajectories using the $13ms^{-1}$ trajectory library in Figure 22b. Collisions were successfully avoided in all 10 runs at both speeds. Using the $9ms^{-1}$ trajectory library, 9 runs reach the goal while 1 run ends with an emergency hover within the rows of poles. The run that ends in an emergency hover, run 10, is an example of where a dynamically feasible collision-free trajectory most likely existed prior to executing the emergency hover, but due to the limited number of final target positions used to compute the potential reference trajectories (to save on computation) no collision-free reference



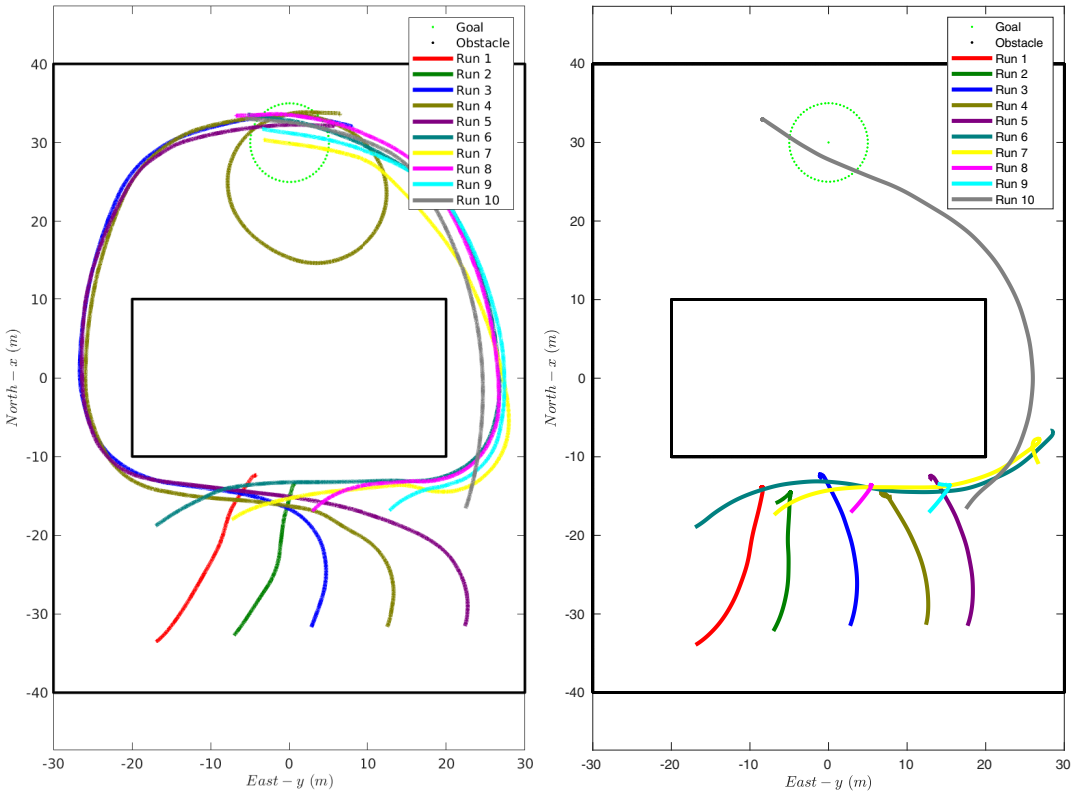
(a) $9ms^{-1}$ **(b)** $13ms^{-1}$

Figure 22. Environment 2

trajectory was found. Using the $13ms^{-1}$ trajectory library, 7 runs reach the goal while 3 runs ends with an emergency hover. Out of the emergency hovers, 2 runs hover within the rows of poles, while 1 run makes it through the rows of poles, but cannot turn sharply enough to reach the goal, and ends up hovering near the wall.

5.3. Environment 3

The third test environment is used to test the aircraft approaching a wall. Environment 3 consists of a $40m \times 20m$ rectangular obstacle enclosed by a $60m \times 80m$ rectangular wall. This environment would be a relatively simple environment to maneuver through if planning with a known map of the environment, but is more difficult when only planning locally. At $9ms^{-1}$, shown in Figure 23a, all 10 flights avoid collisions, but only 8 reach the goal. In runs 1-5, the aircraft initially heads toward rectangular obstacle since that is the same direction towards the goal, and the obstacle has yet to be detected. In runs 3-5, the aircraft turns before reaching the rectangle, and eventually reaches the goal. In runs 1 and 2, by the time the aircraft starts turning right to avoid the rectangle, there is not enough room to complete the turn, and the aircraft hovers to avoid a collision. In runs 6-10, the aircraft immediately sees the rectangular obstacle, and initially flies parallel to the front side. The aircraft can then turn sharply enough to avoid hitting the outer wall, and eventually reaches the goal. While it is unintuitive that the runs starting closer to the obstacle have better success in maneuvering around the obstacle, the reason is the aircraft is not initially facing the obstacle (in all runs), and because the aircraft is closer to the obstacle, it can detect the obstacle before turning directly towards it, which enables the aircraft to maneuver around it. In runs 1-5, the aircraft cannot detect the rectangle before it turns toward it.



(a) $9ms^{-1}$ **(b)** $13ms^{-1}$

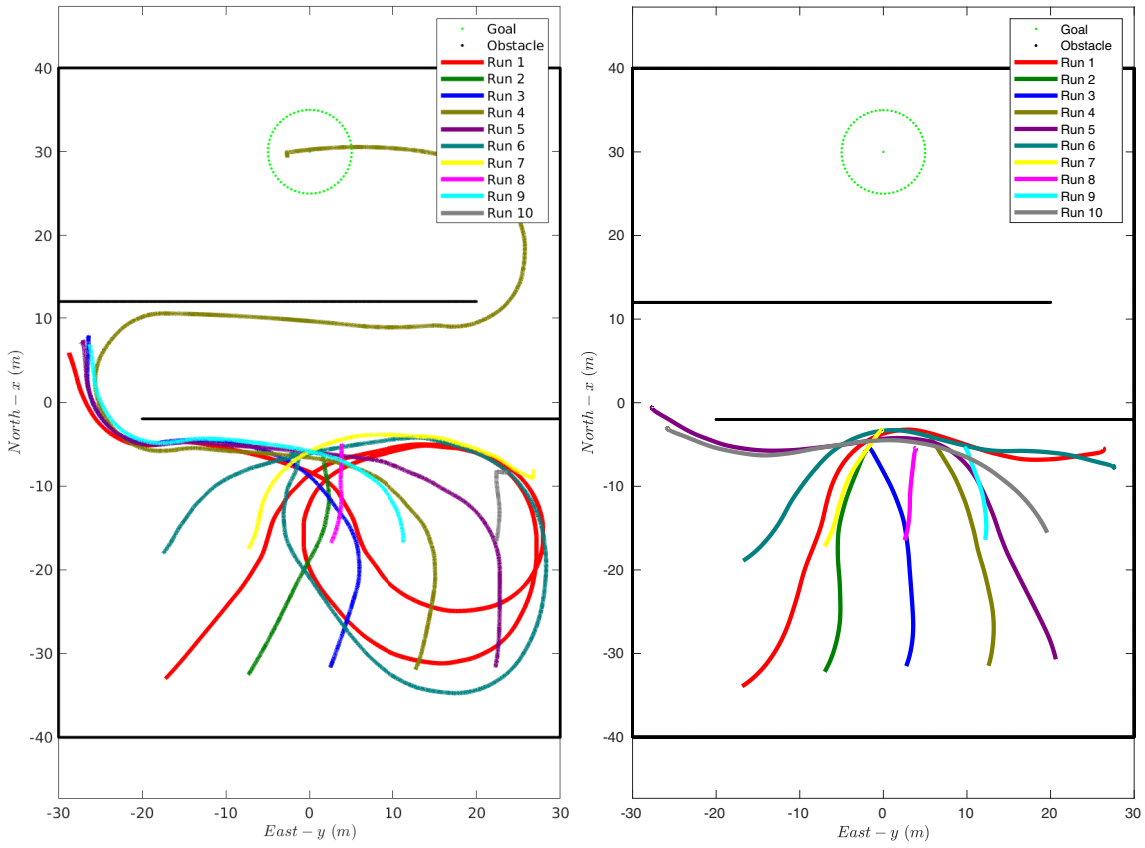
Figure 23. Environment 3

At $13ms^{-1}$, shown in Figure 23b, the aircraft has much less success reaching the goal. While it does avoid collisions in all 10 flights, it only reaches the goal once. Due to the larger turning radius at higher speed flight, the aircraft cannot turn sharply enough to avoid hitting the rectangle and outer walls, and thus hovers to avoid collisions. In run 10, where the aircraft does reach the goal, the aircraft is not required to make any sharp turns due to its initial pose.

5.4. Environment 4

Environment 4 is a labyrinth type environment. Similarly to the third environment, this environment is much harder to navigate through using local planning than if the entire obstacle-map was known a priori. Looking at the $9ms^{-1}$ trajectories, shown in Figure 24a, all of the runs avoid collisions, but only one reaches the goal. When trajectories are only planned locally, they don't factor in the distant future motion. As we can see in multiple runs (runs 1,3,5,9), the aircraft has difficulty completing the U-turn to pass through the opening and avoid the walls. The aircraft has difficulty making this turn because it is unaware of the second east-west wall until passes through the opening, giving the aircraft little room to plan around. During the one run which completes the U-turn (run 4), the aircraft initiated the U-turn slightly south in comparison to the runs which ended in an emergency hover, which enabled the aircraft to finish the U-turn.

Turning our attention to the $13ms^{-1}$ trajectories, shown in Figure 24b, none of the trajectories reach the goal, but all avoid collisions. In comparison to the $9ms^{-1}$ flights, the aircraft has a larger turning radius–and can never execute a U-turn in through the opening. An environment like this



(a) $9ms^{-1}$ **(b)** $13ms^{-1}$

Figure 24. Environment 4

demonstrates the need for a global planner and SLAM implementation, while still showcasing the obstacle-avoidance's capability to avoid collisions in difficult environments.

6. Platform Description

The experimental test platform is depicted in Figure 1. The airframe is an off-the-shelf RC aircraft, the WM Parkflyers McFoamy, which is retrofitted with additional carbon fiber reinforcements and a custom 3D printed motor mount. The aircraft is made of EPP foam, and is equipped with a 50C 3S 850mA LiPo battery, 3 HiTEC HS-65MG metal gear feather servos, a T-Motor AT2312 Long Shaft 1150KV Brushless Motor, the Turnigy Plush-32 30A Speed Controller, and the Master Airscrew MR Series - 10 x 4.5 propeller. The aircraft's computing and sensing suite contains a Pixracer flight controller (Meier et al., 2012) running the PX4 flight stack (Meier et al., 2015), an ODROID-XU4 companion computer, and an Intel Realsense D435 depth camera. The equipped aircraft has a total mass of 660g, and shown in Figures 25 & 26 with labeled hardware components.

The Pixracer/PX4 combination was chosen due to the low cost, ease of use, and ability to customize. The Pixracer contains an internal IMU (gyroscope, accelerometer, and magnetometer), and the ability to easily connect a GPS module. The PX4 flight stack fuses the sensor measurements and provides a state estimate of the aircraft. We chose the ODROID-XU4 companion computer because of its low mass (58g) and high computational abilities, which include a 2GHz quad-core

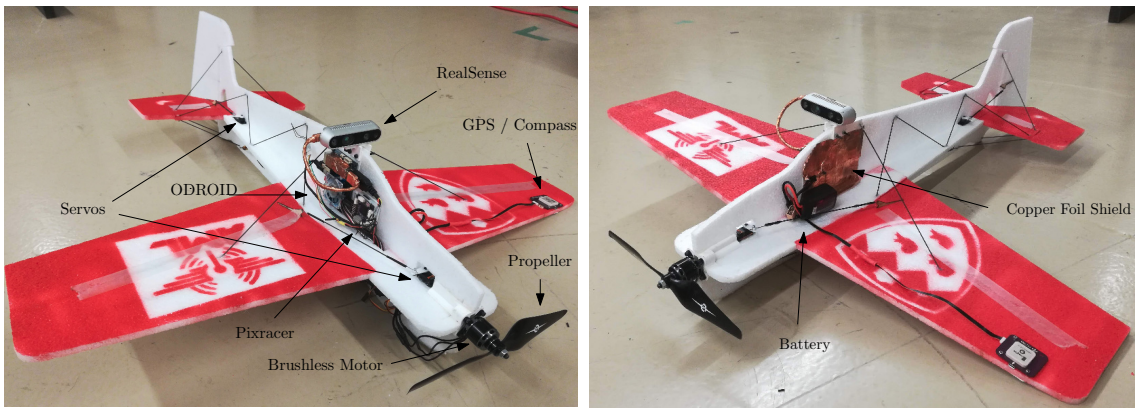


Figure 25. Mcfoamy Hardware: Top

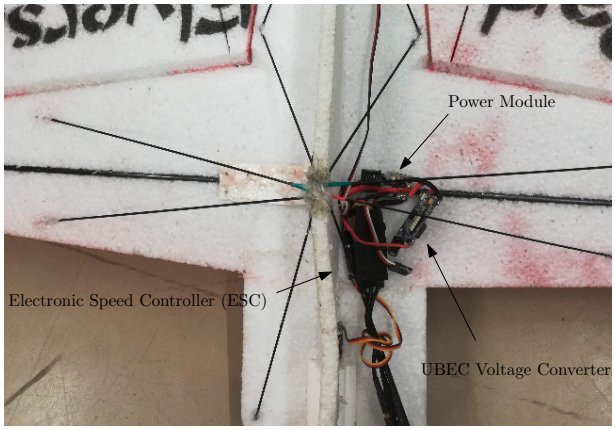


Figure 26. Mcfoamy Hardware: Bottom

processor and 2 GB RAM. In addition, it is capable of running the Ubuntu 16.04 operating system along with Ros-Kinetic, and can easily communicate with the Pixracer.

We chose the Intel Realsense D435 for obstacle detection due to its weight, power consumption, range, field-of-view, and ease of use. The RealSense contains two global shutter imagers, and RGB camera, and an IR projector. The system is 72 grams, has a field-of-view of $86° \times 57°$ (horizontal $\times$ vertical), and a maximum range of over $10m$ that varies with lighting conditions, according to (Intel, 2020). We found the system provides reliable depth measurements outdoors at much larger ranges, and thus we rely on the system having a $20m$ range. We use the camera at 30 frames per second, although it can work as fast at 90 fps. The depth camera is placed such that part of the image is obstructed by the propeller. When the propeller is spinning, the depth image is unaffected with a disabled IR projector, but affected when the IR projector is enabled. Thus we disable the IR projector for this reason, as well as the fact that it is not needed for outdoor operation and consumes power. For the rest of the camera settings we refer to (Pohl et al., 2019), which provides RealSense settings for drone collision avoidance applications.

6.1. System Communication

The ODROID is powered by a 5V/3A UBEC voltage converter from the Pixracer's power module, and communicates to the Pixracer from a USB port on the ODROID to the telemetry (serial) port of the Pixracer through an FTDI cable. The ODROID also provides power and communication with the RealSense using a USB3 cable.

The communication between various hardware and software components is shown in Figure 27. As shown in the diagram, all control and obstacle-avoidance computation is done on the ODROID, while the Pixracer, which runs in 'off-board' mode, is only used for state estimation and to send PWM signals. The MAVLink protocol is used to communicate between the Pixracer and ODROID, and all internal communication on the ODROID, as well as communication between the ODROID and the RealSense, is done using ROS. We also note that MAVROS, a package used to convert between MAVLink and ROS, changes the inertial coordinate frame to East-North-Up (ENU) and the body frame to Front-Left-Up (FLU), so we create a bridge node to convert back to North-East-Down (NED) and Front-Right-Down (FRD), which is sent to the obstacle-avoidance and controller nodes.

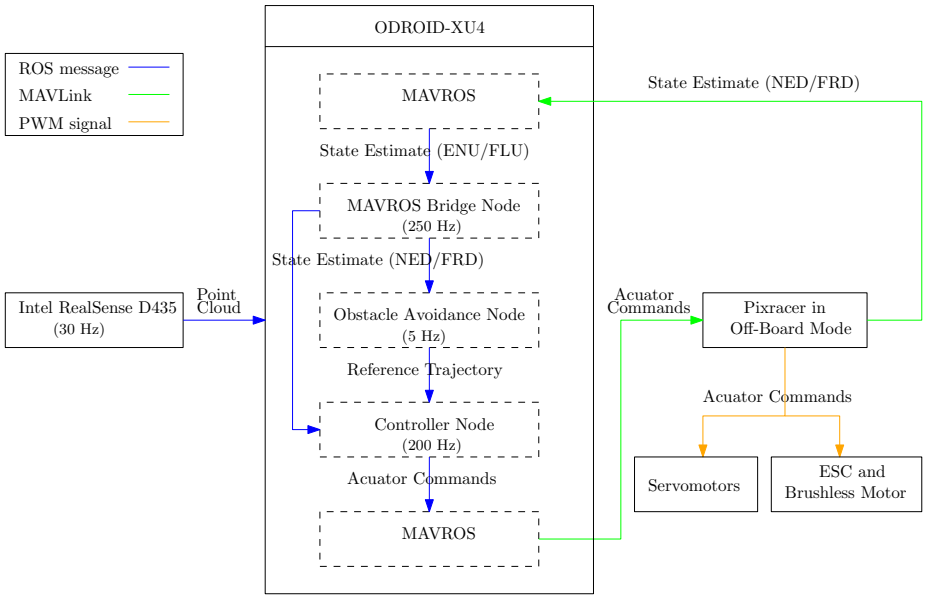


Figure 27. Hardware and Software Communication Diagram

6.2. USB3 Interference

The RealSense D435 requires a USB3 connection, which interferes with the GPS signal. Without adequate shielding this inference can make the GPS so noisy that it is not usable. We were able to significantly reduce the interference by moving the GPS unit to the end of the wing (further from the RealSense and ODROID), lining the side of the fuselage (between the ODROID and GPS unit) with copper foil, and wrapping the USB3 cable in copper foil. These modifications are apparent in Figure 25.

6.3. Parameters

For both the simulated and actual aircraft, we show the values of the physical properties in Table 3, and the values of the parameters used in the obstacle-avoidance algorithm in Table 4. For the true camera parameters, we used the manufacturer's specifications for the experiment column, and the values specified in the Gazebo plugin for the simulation column.

Table 3. Aircraft Physical Properties

Parameter	Symbol	Simulation Value	Experiment Value	Unit
Mass	m	0.660	0.660	kg
Moments of Inertia	I_x	3.922×10^{-3}	3.922×10^{-3}	kgm^2
	I_y	1.594×10^{-2}	1.594×10^{-2}	kgm^2
	I_z	1.934×10^{-2}	1.934×10^{-2}	kgm^2
Non-zero Products of Inertia	I_{xz}	3.03×10^{-4}	3.03×10^{-4}	kgm^2
Wing Area		.143	.143	m^2
Wing Span		.864	.864	m
Mean Aerodynamic Chord		.21	.21	m
Maximum Thruster Rotational Speed	$u^s_{1_{max}}$	6800	6800	RPM
Maximum Aileron Deflection	$u^s_{2_{max}}$	52	52	deg
Maximum Elevator Deflection	$u^s_{3_{max}}$	55	55	deg
Maximum Rudder Deflection	$u^s_{4_{max}}$	40	40	deg

Table 4. Obstacle-Avoidance Parameters

Parameter	Symbol	Simulation Value	Experiment Value	Unit
Camera Range in Algorithm		20	20	m
True Camera Range		20	10+	m
Field-of-View in Algorithm		65×58	65×58	$^\circ \times ^\circ$
True Field-of-View		86×58	86×58	$^\circ \times ^\circ$
Aircraft Wingspan in Algorithm		2	4	m
Intermediate Exit Distance		12	10	m
Weight Used to Compute c^{diff}	w^{diff}	0.03	0.03	$\frac{s}{^\circ}$
Weight Used to Compute c^h	w^h	15	15	rad^{-1}
Weight Used to Compute c^{obs}	w^{obs}	2	2	m^{-1}
Weight Used to Compute c^v	w^v	2	2	m^{-1}

7. Outdoor Flight Tests

Our obstacle-avoidance flight testing campaign began in August 2019 and continued through November 2019, at which time flight testing was suspended for the colder months and the subsequent COVID-19 lockdown. We resumed flight testing in June 2020 through November 2020. Due to the various regulations regarding COVID-19, our access to flight testing locations was highly variable. Flight tests were conducted at the Macdonald Campus of McGill University, the West Island Model Aeronautics Club (WIMAC), the Montreal Area Thermal Soarers Club (MATS), and the cottage

of a generous McGill Professor. All of these locations are either in or near Montreal, Canada, and contain large plots of land with a mix of open space, sparse trees, and dense forest.

A great deal of time was spent at the flight testing sites diagnosing and fixing hardware and software issues unrelated to the actual obstacle-avoidance methodology. Initially, the Intel RealSense D435 depth camera output too many false-positive points within the point cloud, which was largely resolved by using the settings presented in (Pohl et al., 2019). Another issue arose due to USB3 connection to the Intel Realsense, which interfered with the GPS signal. This issue was mostly resolved by shielding the electronics with copper foil and moving the GPS further from the electronics. The last major issue was a bug in the trajectory generation code, which was easily fixed once diagnosed.

We completed 6 days of flight testing with all the aforementioned issues resolved, where we were able to focus on gathering experimental results with the completely autonomous aircraft. We executed 35 runs, where each run consists of hand launching the aircraft; manually flying the aircraft to a region where a tree must be avoided in order to reach the goal location; switching the control authority from manual to autonomous; autonomously avoiding trees; and then concluding by either (a) hovering at the goal, (b) executing an emergency hover, or (c) colliding with a tree. Once the aircraft enters a hover, control is given back to the pilot, who manually lands the aircraft. At any instant during these flights, any error could cause the aircraft to crash into a tree. Videos from flight testing can be found in the Youtube playlist: https://www.youtube.com/playlist?list=PLAqQI_mCMFP7HriBt5VIMf7xZmXUyikAV.

The 35 autonomous runs take place in six different test environments. While we are satisfied with experimental validation in six environments, the multiple environments are more a consequence of where permission is obtained to fly the aircraft, rather than an intentional testing strategy. We believe that all of the environments tested in experiment are of either similar difficulty, or slightly easier than that of simulation Environment 1. While we would have liked to also test experimentally in environments resembling simulation Environments 2 - 4, our options for field testing were limited due to the regulatory permissions required.

In Figures 28 - 32, we show the testing environments using Google Earth, and approximately mark the start and goal locations on the images. For Environments 1 & 6, which are both at WIMAC, three-dimensional renderings are available. For the remaining environments, only top-down two-dimensional images are available.

The start and goal location in each environment are selected to force the aircraft to avoid obstacles in order to reach the goal. As we can see in Figure 28, in experimental Environment 1 the aircraft is always forced to avoid atleast one large tree. Depending on the initial heading, the aircraft may be forced to avoid additional trees. In the second experimental environment shown in Figure 29, the

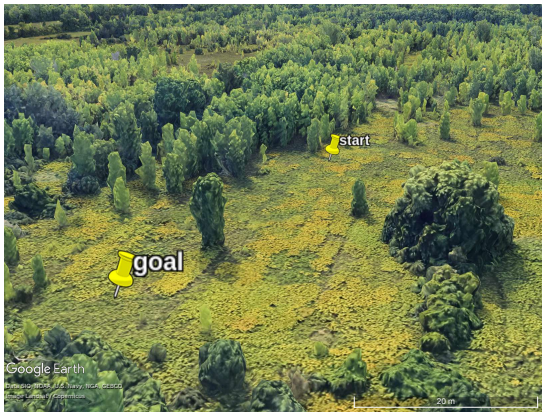


Figure 28. Environment 1 (at WIMAC)

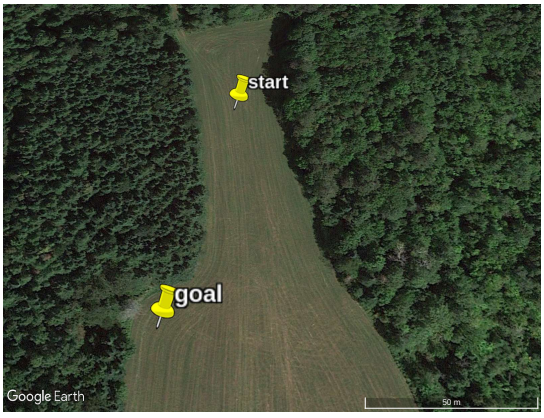


Figure 29. Environment 2 (at cottage)

Figure 30. Environment 3 (at MATS)

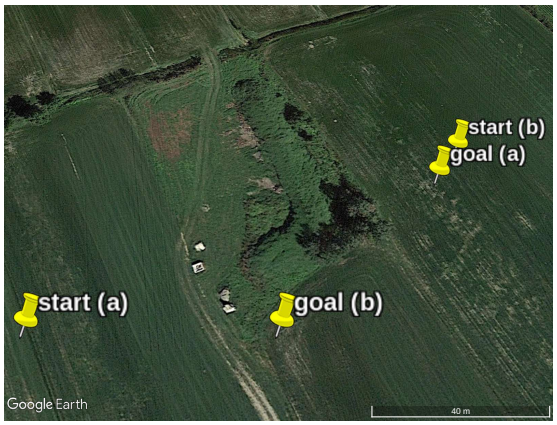


Figure 31. Environment 4 (start/goal (a)) & 5 (start/goal (b)) (at MATS)

aircraft is forced to fly beside a wall of trees, and turn towards the goal once the space widens, as the direct line from start to goal would go through the trees. In the third experimental environment, shown in Figure 30, the aircraft is always forced to avoid one large tree, and depending on the path chosen, would potentially need to avoid more smaller trees. In the fourth and fifth experimental environments, shown in Figure 31, the aircraft is forced to avoid a large mound and one large tree. In the sixth experimental environment, in Figure 32, the aircraft is forced to avoid multiple trees.

Figure 32. Environment 6 (at WIMAC)

Thoroughly explaining the aircraft's motion during an autonomous run requires a large volume of data, including state estimates, reference states, external video images, on-board color images, on-board depth images, and trajectories overlaid onto satellite images. For this reason, we summarize the results for all of the runs using various statistics in Sec. 7.1, followed by a brief analysis of all the runs individually using only top-down satellite imagery in Sec. 7.2, and finally a detailed analysis of four selected runs in Sec. 7.3.

7.1. Summary

Table 5 summarizes the 35 autonomous runs where, for each run, we specify the environment (Env.), date, the trajectory library speed (Lib.), the configuration of the control and obstacle-avoidance

Table 5. Summary of Runs

Run	Env.	Date	Lib.	Conf.	Dist.	Time	Avg.Spd	Max.Spd	Outcome
1	1	15/09/20	9	a	78.00	10.20	7.64	7.91	reached goal
2	1	15/09/20	9	a	53.79	7.17	7.51	8.22	emergency hover
3	1	15/09/20	9	a	108.05	14.23	7.59	8.23	reached goal
4	1	15/09/20	9	a	134.90	18.03	7.48	9.02	reached goal
5	1	24/09/20	9	a	219.19	33.83	6.48	8.85	emergency hover
6	1	24/09/20	9	a	66.33	8.48	7.82	11.85	emergency hover
7	1	24/09/20	9	a	135.00	19.45	6.94	9.48	emergency hover
8	1	24/09/20	9	a	123.77	17.07	7.25	9.98	emergency hover
9	1	24/09/20	11	a	221.35	26.02	8.51	11.13	emergency hover
10	1	24/09/20	11	a	34.15	3.95	8.64	9.84	emergency hover
11	1	24/09/20	11	a	55.67	7.43	7.49	9.56	emergency hover
12	1	24/09/20	11	a	121.89	14.59	8.36	10.50	emergency hover
13	1	24/09/20	11	a	101.92	11.20	9.10	12.56	emergency hover
14	1	24/09/20	11	a	283.24	36.58	7.74	11.78	emergency hover
15	2	31/10/20	11	b	129.32	15.99	8.09	11.04	reached goal
16	2	31/10/20	11	b	102.23	12.90	7.92	14.42	reached goal
17	2	31/10/20	11	b	155.95	20.22	7.71	13.11	collision
18	2	31/10/20	11	b	117.20	13.23	8.86	11.00	collision
19	3	9/11/20	9	c	160.25	17.96	8.92	10.25	reached goal
20	3	9/11/20	9	c	58.58	6.94	8.44	9.35	emergency hover
21	3	9/11/20	9	c	39.25	4.96	7.91	9.83	emergency hover
22	4	13/11/20	9	d	153.97	17.98	8.56	10.63	reached goal
23	4	13/11/20	9	d	250.37	29.93	8.37	11.11	reached goal
24	4	13/11/20	9	d	142.09	16.97	8.37	10.94	reached goal
25	4	13/11/20	9	d	175.45	21.47	8.17	13.49	reached goal
26	4	13/11/20	9	d	131.25	15.57	8.43	11.59	reached goal
27	4	13/11/20	9	d	205.21	24.29	8.45	11.01	collision
28	5	13/11/20	9	d	106.25	11.90	8.93	10.39	emergency hover
29	5	13/11/20	9	d	115.59	12.85	8.99	10.75	reached goal
30	5	13/11/20	9	d	113.28	12.16	9.31	9.95	reached goal
31	5	13/11/20	9	d	138.25	15.65	8.84	9.36	reached goal
32	5	13/11/20	9	d	90.43	11.23	8.05	12.96	emergency hover
33	1	19/11/20	9	d	102.76	11.44	8.98	9.93	reached goal
34	6	19/11/20	9	d	119.07	13.00	9.16	9.68	reached goal
35	6	19/11/20	9	d	46.43	7.82	5.93	8.75	collision

algorithm (Conf.), the total distance traveled in m (Dist.), the duration of the run in s, the average speed in ms^{-1} (Avg.Spd), the maximum speed in ms^{-1} (Max.Spd), and whether the run ended because the aircraft reached the goal, executed an emergency hover, or collided with a tree. During the flight testing campaign we made a few modifications to the control and obstacle-avoidance algorithm, which are denoted by configuration a,b,c and d, and discussed in Sec. 7.2.

In total, the aircraft autonomously flew $4.4km$ over $543s$ while avoiding trees, and maintained an average speed of $8.1ms^{-1}$ and a top speed of $14.4ms^{-1}$. Out of the 35 runs, 16 reached the goal, 15 ended with an emergency hover, and 4 collided with a tree. Given the difficulty of the research problem at hand, we are very satisfied with these results. For comparison, the only existing research that attempts to autonomously fly a fixed-wing aircraft through an unknown and unstructured environment using only on-board computation and sensing is in (Barry et al., 2018; Barry, 2016). In that work, the aircraft crashed into a tree 10 times out of 26 runs. Thus, our 4 collisions out of 34 runs is significantly better than the state-of-the-art. Furthermore, none of the collisions were the fault of the obstacle-avoidance algorithm; one collision was caused by a mechanical failure resulting in loss of aileron control, and the other three were a result of a camera failure.

7.2. High-Level Analysis

We show top-down views of the flight trajectories of each run overlaid onto a Google satellite images of the test environments in Figures 33 - 41. Looking at Figure 33, we can see Runs 1-4 all avoid the tree in the middle of the image, and Run 4 also avoids the trees in the bottom right of the image. While Runs 1,3 & 4 all reach the goal, Run 2 ends with an emergency hover.

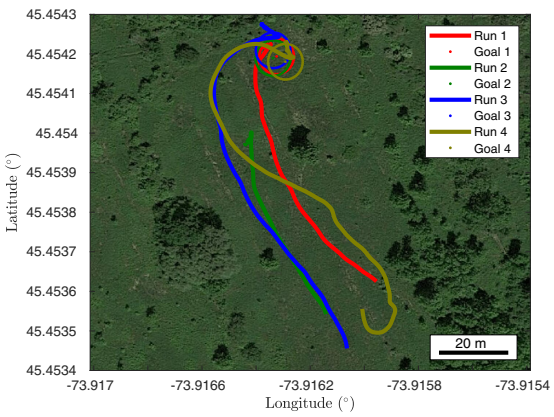


Figure 33. Top-Down View from Runs 1-4

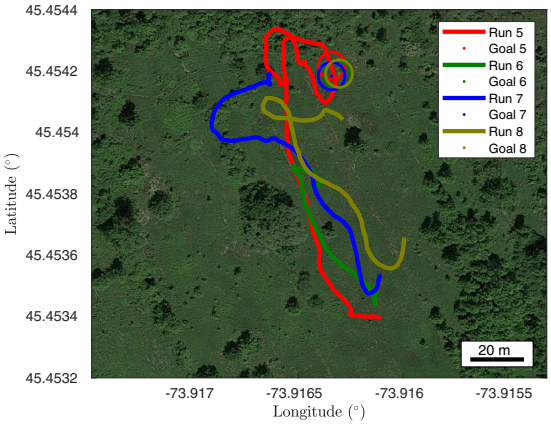


Figure 34. Top-Down View from Runs 5-8

As shown in the image, the emergency hover is executed in free space. The reason for this hover is as follows: the aircraft significantly pitches up (likely because of a wind gust) for a short period of time. During that time, the obstacle-avoidance algorithm computes the trim trajectories that lead to the edge of the FOV. Since, the camera is fixed to the aircraft, and the aircraft is significantly pitched up, all the positions along the edge of the FOV require a large climb rate to reach those positions. If all of the these climb rates are not feasible to achieve using trim primitives, the aircraft executes an emergency hover, to avoid flying in a space which it cannot see. Emergency hovering for this reason also occurs in Runs 5,6, & 12. Hovering in these scenarios is a conservative approach. For Runs 1-14, we use this conservative approach, which we label configuration 'a' in Table. 5. Another option in this scenario would be to continue on the previously selected reference trajectory until the next motion planning iteration. We use this approach for the remainder of the runs, which is the case for configurations 'b', 'c', and 'd' in Table. 5.

Runs 5-8, which use a $9ms^{-1}$ trajectory library, are shown in Figure 34 and Runs 9-14, which use an $11ms^{-1}$ trajectory library, are shown in Figure 35. An interesting phenomenon occurs when initializing the aircraft heading away from the goal. In Runs 4,6,7 & 8, all which use a $9ms^{-1}$ trajectory library, are able to turn and avoid the very large tree (right side of Figure 28, middle of Figure 34). Runs 10 & 11, which use an $11ms^{-1}$ trajectory library, both cannot turn in time to avoid that tree and execute an emergency hover, likely because of the fewer available turning radii in the higher speed trajectory library.

Runs 5-14 occur on the same day, which was windy. Strong winds affect the aircraft's motion beyond pitching up the aircraft and prematurely hovering. In many instances, the aircraft does not take a direct route to the goal. This can be attributed to two reasons. First, say a wind gust yaws the aircraft. Since the aircraft always selects trajectories starting from its current yaw angle, if a wind gust yaws the aircraft, the flight trajectory will completely change, since the aircraft will now select trajectories starting from its new yaw angle. Second, say there is a constant wind pushing the aircraft to the right. If the aircraft wants to turn right, the reference trajectory will originate from the aircraft location while turning right. However, during the time the motion plan is computing, the wind will cause the aircraft to be right of the initial part of the right-turning trajectory, so then the position controller will direct the aircraft to the left. This attempt to stay on the originally planned trajectory results in longer turns and ultimately an indirect path to the goal.

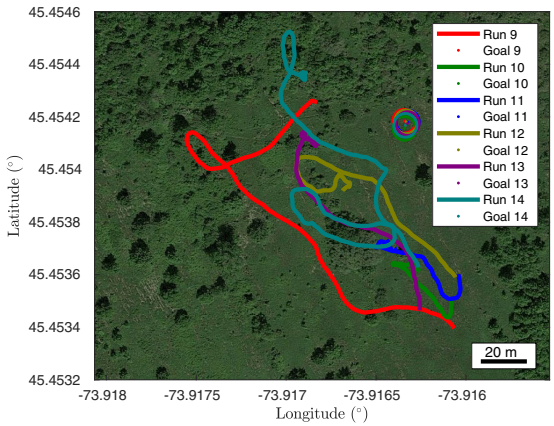


Figure 35. Top-Down View from Runs 9-14

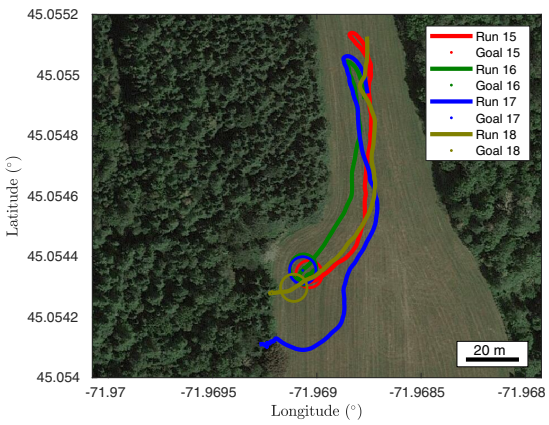


Figure 36. Top-Down View from Runs 15-18

Looking at Figure 36, the aircraft needs to avoid a wall of dense trees, and then turn after the wall to the reach the goal. The aircraft succeeds at reaching the goal in Runs 15 & 16, but collides with a tree in Runs 17 & 18. During both collisions, the depth camera failed to detect parts of the tree leading up to the collision, and the aircraft therefore flew into what it thought was free space, but was not.

Between Runs 18 & 19, a coding mistake was found in the controller. In addition to fixing this error, the control gains were also adjusted. These changes are denoted by configurations 'c' and 'd' in Table 5.

Looking at Figure 37, Run 19 reaches the goal, while Runs 20 & 21 end with an emergency hover. The satellite image is slightly deceiving, as this test occurred in November when the leaves had fallen. So while it appears that Run 19 goes either above or through the tree, it actually goes around. In all of these runs, a creek separating the grass fields (along the tree line), causes many false positive points to appear in the point cloud, which has a large impact on the aircraft's motion.

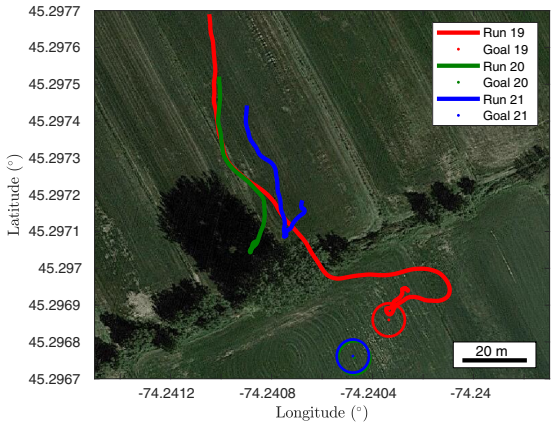


Figure 37. Top-Down View from Runs 19-21

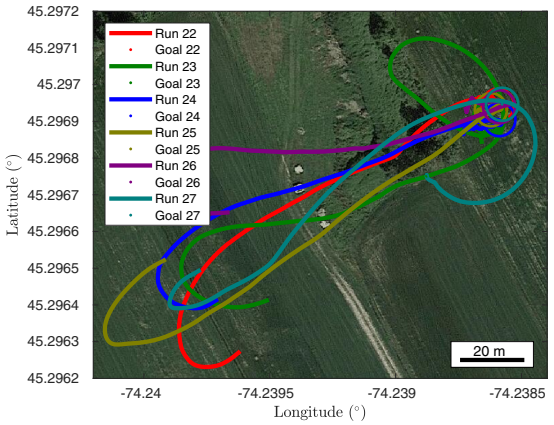


Figure 38. Top-Down View from Runs 22-27

The obstacle-avoidance algorithm will always rely on GPS to reach a desired GPS location. However, testing whether obstacles could still be avoided without GPS is useful, in the case of a temporary loss of GPS signal. The remainder of the runs are conducted with the position control gains set to zero, except when hovering. By doing this, the feedback controller no longer relies on position estimates. So while these runs do use GPS (and position estimates) to give priority to trajectories that reach the goal, the position estimate is not used to avoid obstacles. We also note

that the GPS is still being used in the velocity estimation, but in theory velocity estimates could be obtained without GPS. This modification is represented by configuration 'd' in Table 5.

The obstacles in Environment 4 & 5 (Figure 31) consist of a mound, which gradually increases altitude on the left side, and has a sharp drop on the right side. On this mound there is a tree on each end. We force the aircraft to approach the mound from each side by varying the start and goal, as shown in Figures 38 & 39. When approaching the obstacles from the left side (Figure 38), the aircraft has a tendency to go over the tree, as it starts to climb first because of the mound, and by the time it approaches the tree, it is high enough to either continue straight or slightly climb to fly over the tree. This is shown in Runs 24,25 & 27. In Run 27, the aircraft is too high above the goal's altitude to enter a hover, and ends up turning back towards the tree. Due to a camera malfunction, the camera fails to detect the tree in time to avoid crashing into it. This run is discussed in detail in Sec. 7.3.4. In Runs 22, 23 & 26, the aircraft goes around the tree and reaches the goal, even though it is hard to see due to outdated satellite imagery.

Referring to Figure 39, all of the runs go around the tree. Runs 29-31 reach the goal, while runs 28 & 32 execute an emergency hover. In these runs, the emergency hover was executed due to having false positive points in the point cloud which resulted in the aircraft thinking it had no collision-free options.

Looking at Figure 40, the aircraft reaches the goal by flying around the tree. Referring to Figure 41, in Run 34 the aircraft avoids several trees and then reaches the goal (although hard

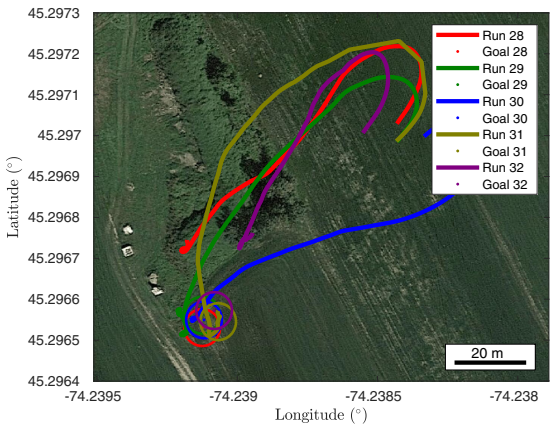


Figure 39. Top-Down View from Runs 28-32

Figure 40. Top-Down View from Runs 33

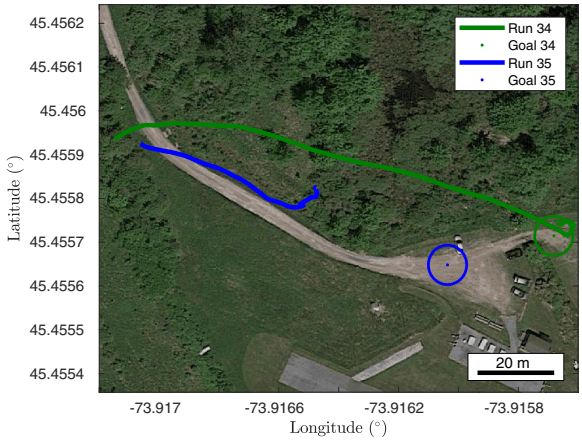


Figure 41. Top-Down View from Runs 34-35

to discern due to outdated satellite imagery). In Run 35, a mechanical failure causes a loss of control of the aileron. The aircraft loses control in roll and subsequently flies straight into a tree.

7.3. Detailed Analysis

While the flight trajectories overlaid on to satellite imagery briefly describe the runs, we now proceed to analyze four runs in more detail. We choose to analyze one successful run that includes position control (Run 4); one successful run that does not use position control to avoid obstacles (Run 22); one run that ends with an emergency hover (Run 20); and one run that ends with a collision (Run 27). At selected times, we show the image from a ground video, the on-board color image, and the on-board depth image. The depth image shows black pixels where there are no obstacles in the point cloud, and a grey-scale for obstacles, where the closer obstacles are represented by darker pixels. While the color and depth image are of roughly the same scene, they do have different fields-of-view. In addition to images, we overlay the flight trajectory onto a satellite view of the flight area. The trajectory line starts blue, and becomes lighter with time, and goal region is represented by the green circle. At the same instants where the images are shown, we display the aircraft's location with a red dot, and its yaw with a red line. In addition to these images, we show plots of the reference states, state estimates, and control inputs. The goal location is considered to be $(x, y, z) = (0, 0, -10)m$, and 'reaching' the goal occurs when the aircraft is within $5m$ of the goal location.

7.3.1. Successful Run using Position Control (Run 4)

We present a detailed analysis of Run 4, where the aircraft successfully avoids trees and reaches the goal. From left to right we show the image from the ground video, the satellite imagery, the on-board color image, and the on-board depth image in Figure 42, for the first seven seconds of the run, and in Figure 43 for the remainder of the run. The reference states, state estimates, and control inputs (throttle (u_1^s), aileron (u_2^s), elevator (u_3^s), rudder (u_4^s)) are shown in Figure 44 for the first seven seconds, in Figure 45 between seven and fifteen seconds, and in Figure 46 for the remainder of the flight. This run is also shown in https://youtu.be/GeFdZbb3yH8.

As shown in both Figures 42 & 44, the aircraft is initialized heading away from the goal. The aircraft banks left to turn around, and by $t = 2.45s$, the aircraft has completely turned around. At this instant, flying straight would be the most direct path to the goal, but the aircraft continues banking left since it sees the tree in front of it, which is shown in the second row of Figure 42. By $t = 2.7s$, the aircraft no longer sees the tree in front of it, and subsequently banks right in order to fly towards the goal. By $t = 3.33s$, the aircraft sees tree again, and banks left to avoid it.

At $5.6s$, the aircraft has avoided the first tree, and sees a new tree located in the middle of the satellite image, which is shown in the fifth row of Figure 42. The aircraft initiates a mild right turn, which steers the aircraft not too far from the goal, and would allow the aircraft to pass the right side of the tree. However, while the trajectory was being selected, the aircraft deviated far from the initial position and yaw in which the trajectory is being originated from since the aircraft had turned quickly in its previous time step. So although the trajectory was meant to turn right, which can be shown by the increasing yaw reference, the start of that increasing yaw reference is less than the current yaw. The position controller augments the reference yaw to be even more negative, which induces a large rudder deflection (u_4^s) and yaws the aircraft to the left (negatively).

By $6s$, the aircraft no longer sees the tree, and subsequently banks right to go towards the goal. At $6.5s$ the aircraft detects the tree, and subsequently banks even more aggressively to the right, as going right of the tree is the shortest collision-free path to the goal. At $7s$, the aircraft

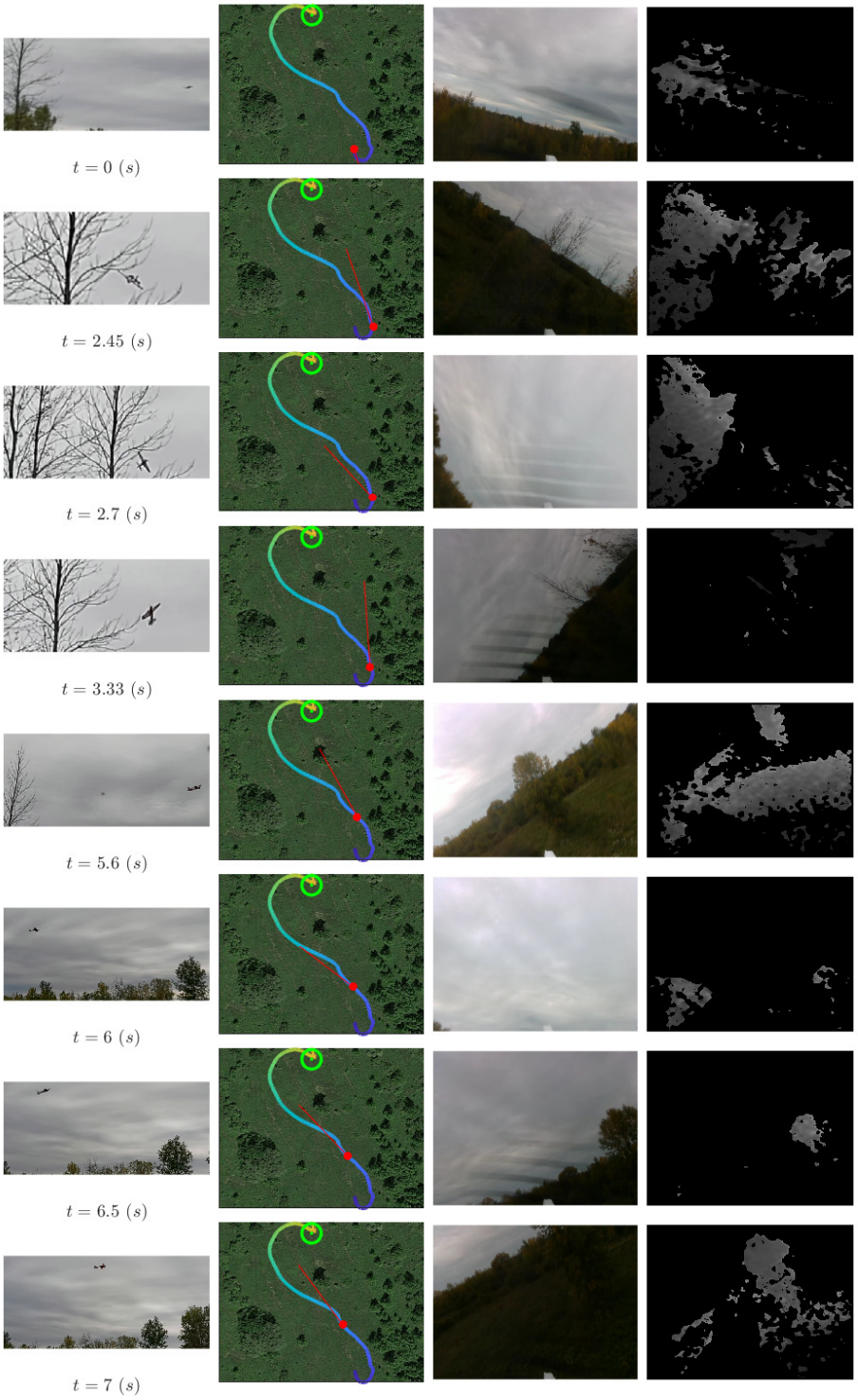


Figure 42. Run 4: ground image, flight trajectory, on-board color image, and on-board depth image ($t = 0 - 7s$)

Figure 43. Run 4: ground image, flight trajectory, on-board color image, and on-board depth image ($t = 7-20s$)

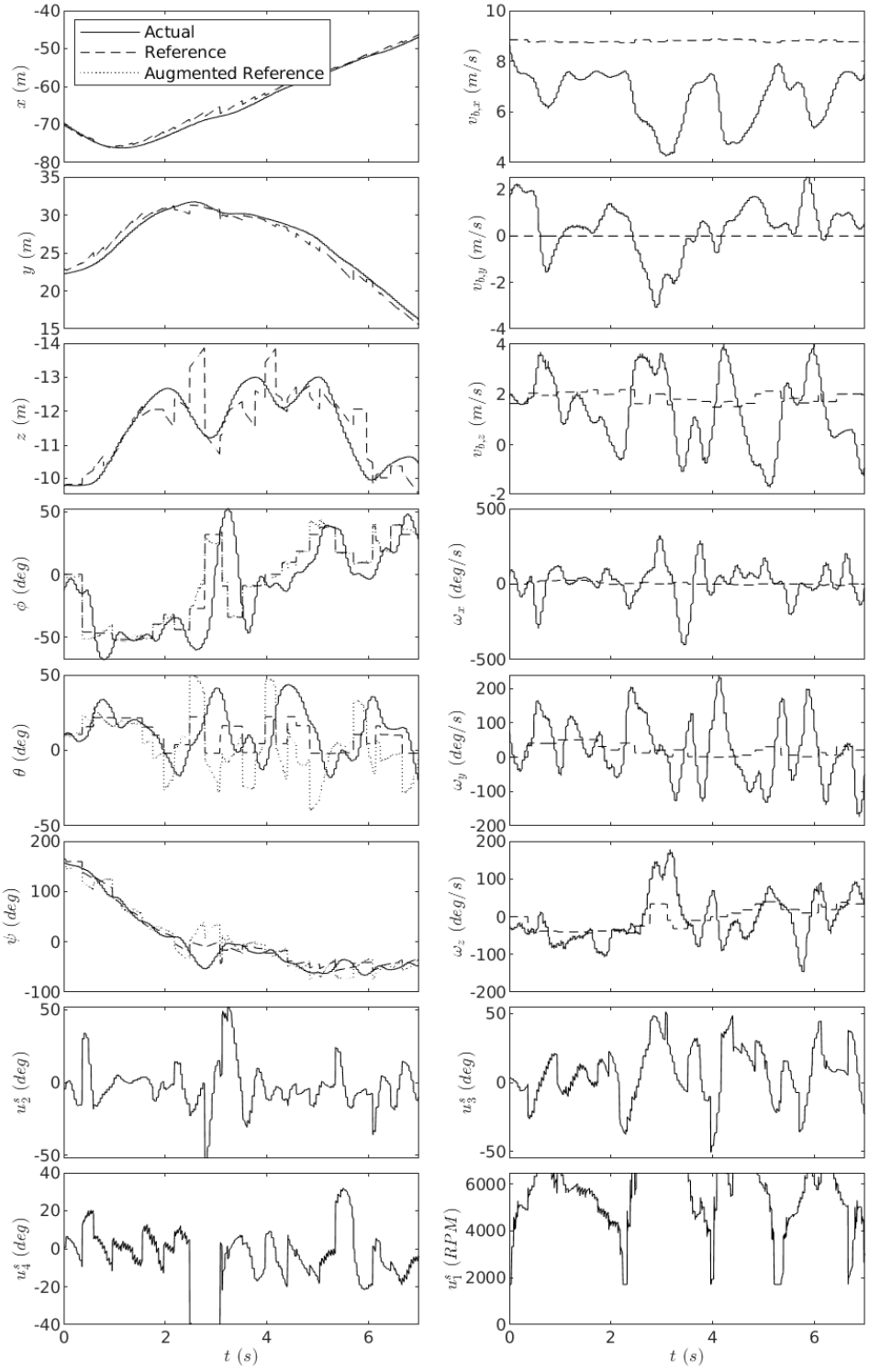


Figure 44. Run 4: state estimates, reference states, and control inputs ($t = 0 - 7s$)

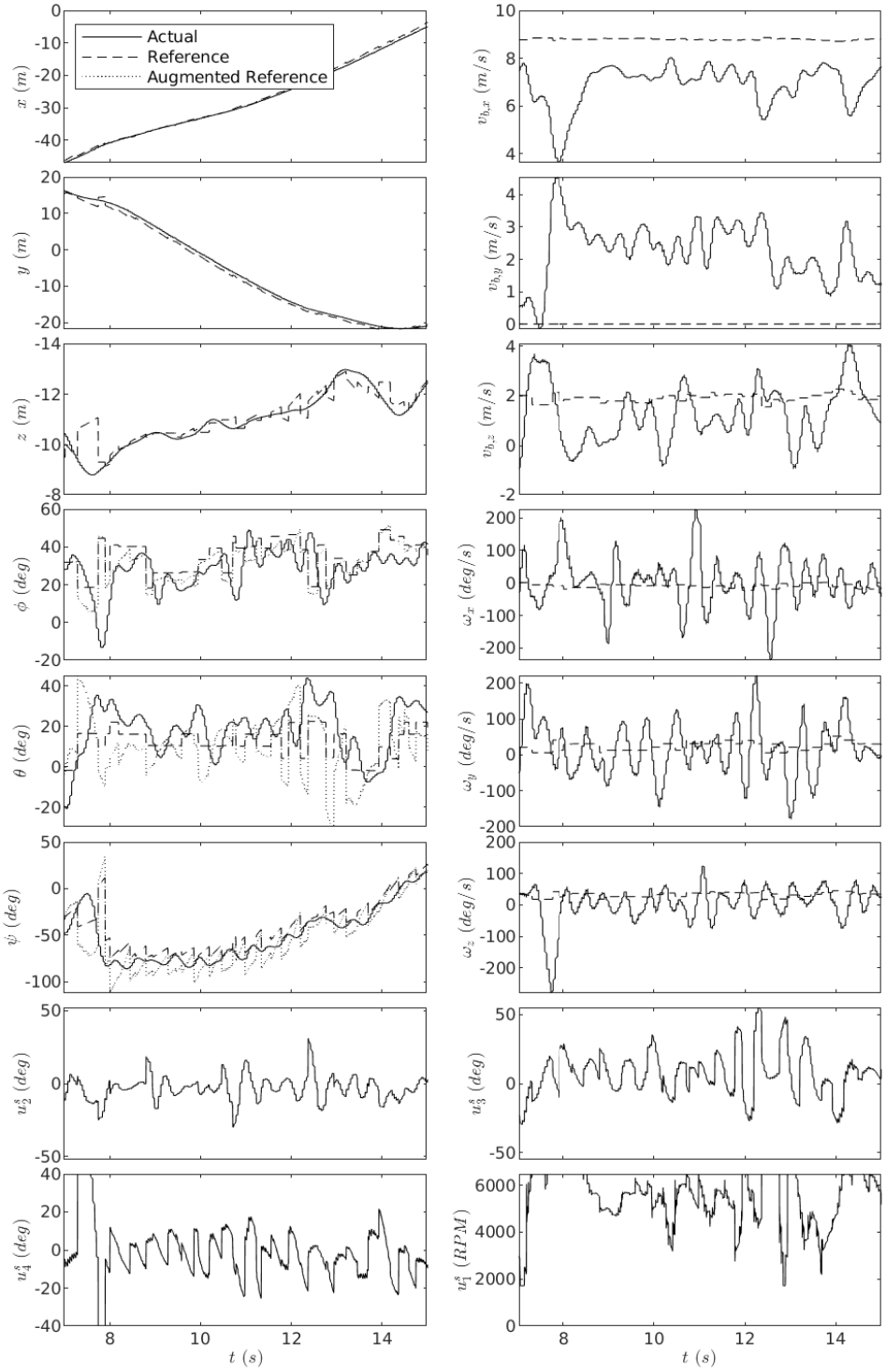


Figure 45. Run 4: state estimates, reference states, and control inputs ($t = 7 - 15s$)

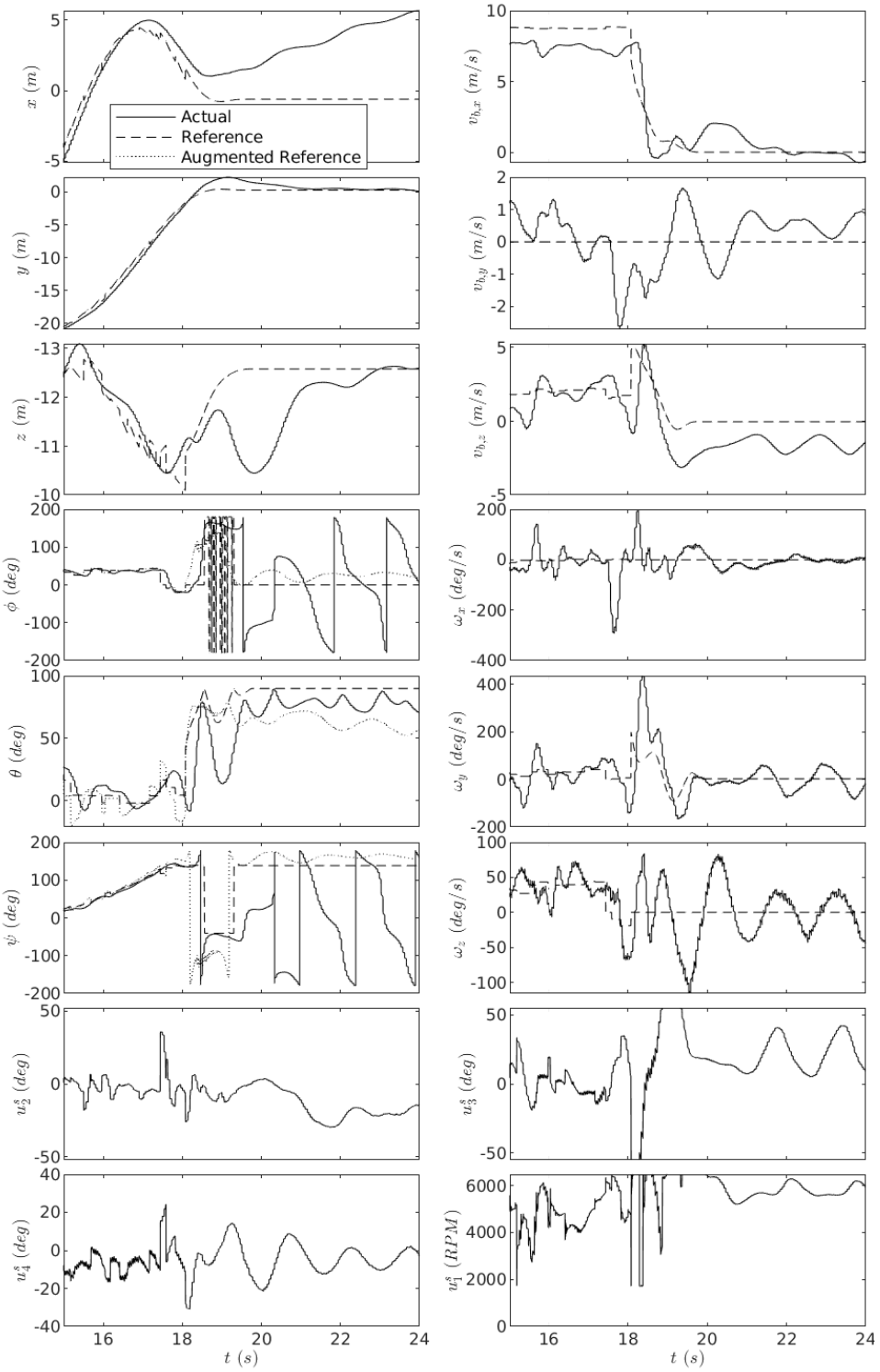


Figure 46. Run 4: state estimates, reference states, and control inputs ($t = 15 - 24s$)

still sees the tree, and continues to select right bank trajectory in order to fly to the right of the tree.

The same issue that occurs at $5.6s$ occurs again. As we can see in Figure 45 the first trajectory switch after $7s$ has an increasing yaw, but is initialized at the yaw of around $7s$ which is much less than the yaw at the time of the trajectory switch. In addition, since the position references are generated based on that 'out-dated' yaw, a large position error causes the augmented reference yaw to be even more negative. A large rudder deflection (u_4^s) negatively yaws the aircraft, and by $8s$ the aircraft is directed left of the tree (in Figure 43, the tree being avoided is on the right side of the ground video image in the first two rows). The aircraft continues to select straight and right turning trajectories as it passes the left side of the tree. By $13.7s$, the aircraft has passed the tree, and turns right until the aircraft reaches the goal at $17.8s$. At this instant, the aircraft enters a hover and holds the hover the remainder of the run.

As shown throughout the run, the aircraft frequently changes reference trajectories in order to avoid obstacles in real-time. These frequent changes in reference states cause oscillatory motion (see Figures 44 - 46). These oscillations are not caused by an unstable feedback controller, but because of the reference states themselves.

The issue of having an 'out-dated' initial position and yaw reference was apparent during this run. While it did lead to a longer path than necessary, the issue did not cause the aircraft to crash. The way to properly address this issue would be to predict the initial position and yaw based on the computation time of the obstacle-avoidance algorithm, and select the reference trajectories from that predicted position and yaw. However, making this prediction would require accurate wind sensing and an accurate dynamics model which can be run on-board in real-time, and both of these are difficult to achieve within our payload limitations.

7.3.2. Successful Run without using Position Control (Run 22)

We now present Run 22 in greater detail, which does not use position control to avoid obstacles. Images from the run can be found in Figure 47, while plots of the reference states, state estimates, and control inputs (throttle (u_1^s), aileron (u_2^s), elevator (u_3^s), rudder (u_4^s)) can be found in Figure 48. This run is shown in https://youtu.be/Ilz3YNOCXr0. Looking at Figure 47, the aircraft is initialized heading a away from the goal. By $t = 5s$, the aircraft has turned around, and heads toward the goal. The aircraft continues to execute level flight and mild banked turns as it flies toward the goal. Although at $t = 10s$ the aircraft sees part of the mound and at $t = 12s$ part of the tree, neither appear to cause a collision with the straight and mild bank turn trajectories, which the aircraft continues to execute. At $t = 13s$, the aircraft clearly sees the tree in front of it, and sharply banks left, which can be seen at $t = 13.6s$ in Figure 47, and in the roll plot in Figure 48. Once the aircraft passes the tree, mild right bank turns are executed until reaching the goal around $t = 18s$.

In comparison to Run 4, which uses position control to avoid obstacles, the trajectory in this run is smoother and more direct to the goal. A possible explanation for this relates to an issue previously mentioned, when the wind causes a mismatch with the initial position of the reference trajectory and the aircraft, and then the position controller corrects for this mismatch by turning in the opposite direction of where the reference trajectory eventually turns. A benefit of not using position control while avoiding obstacles is that this can never happen, since the position control gains are zero. Ultimately, the aircraft flies in a more direct route towards the goal.

A drawback to not using position control is that there is nothing to correct for deviations from the initially planned trajectory in position and velocity, which can cause the aircraft to fly along a slightly different position trajectory than the one that was deemed collision-free.

7.3.3. Emergency Hover (Run 20)

We now present a detailed description of a run that terminates in an emergency hover (Run 20), which can additionally be seen in https://youtu.be/00K866hfaVI. Images from the run are shown

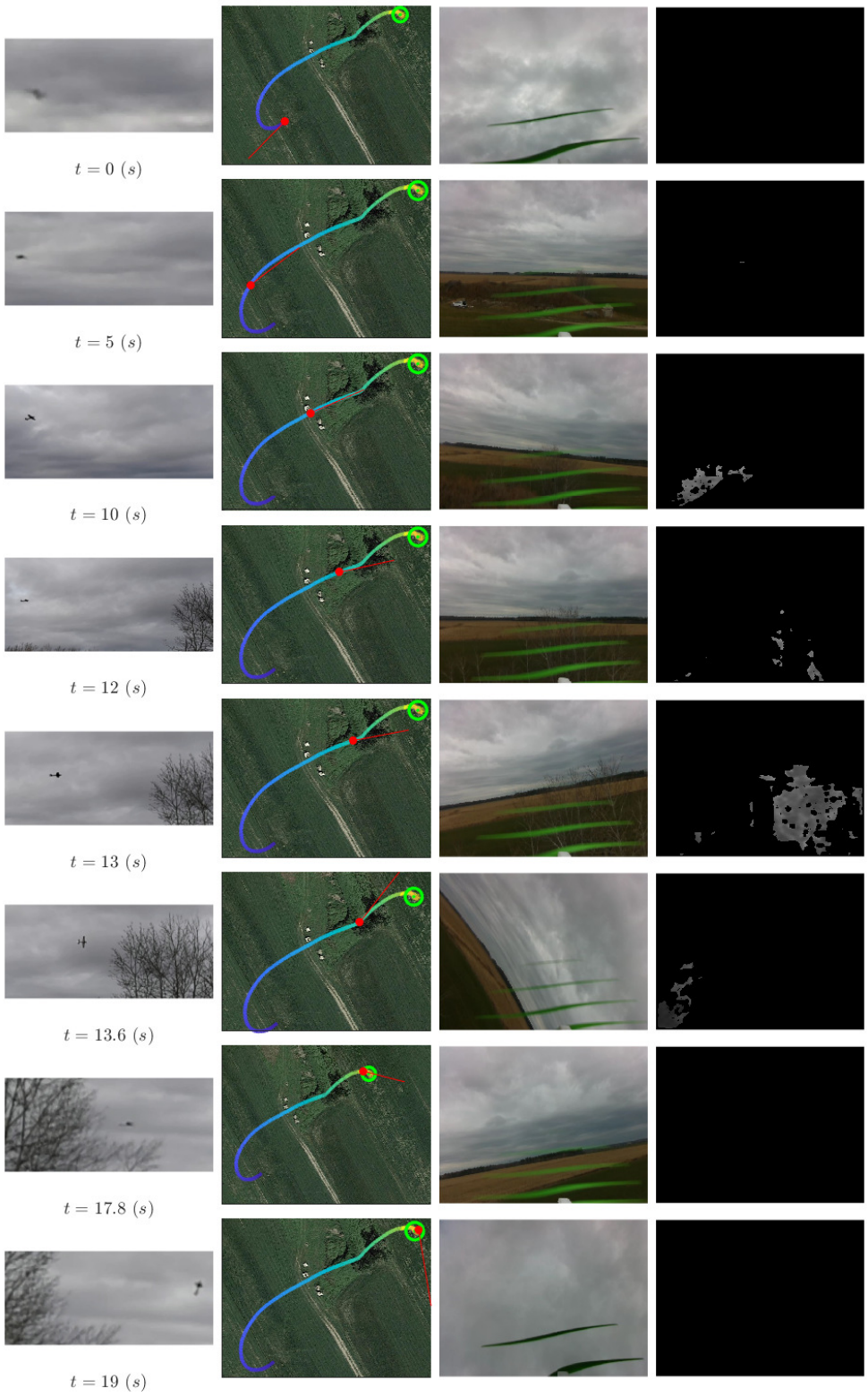


Figure 47. Run 22: ground image, flight trajectory, on-board color image, and on-board depth image

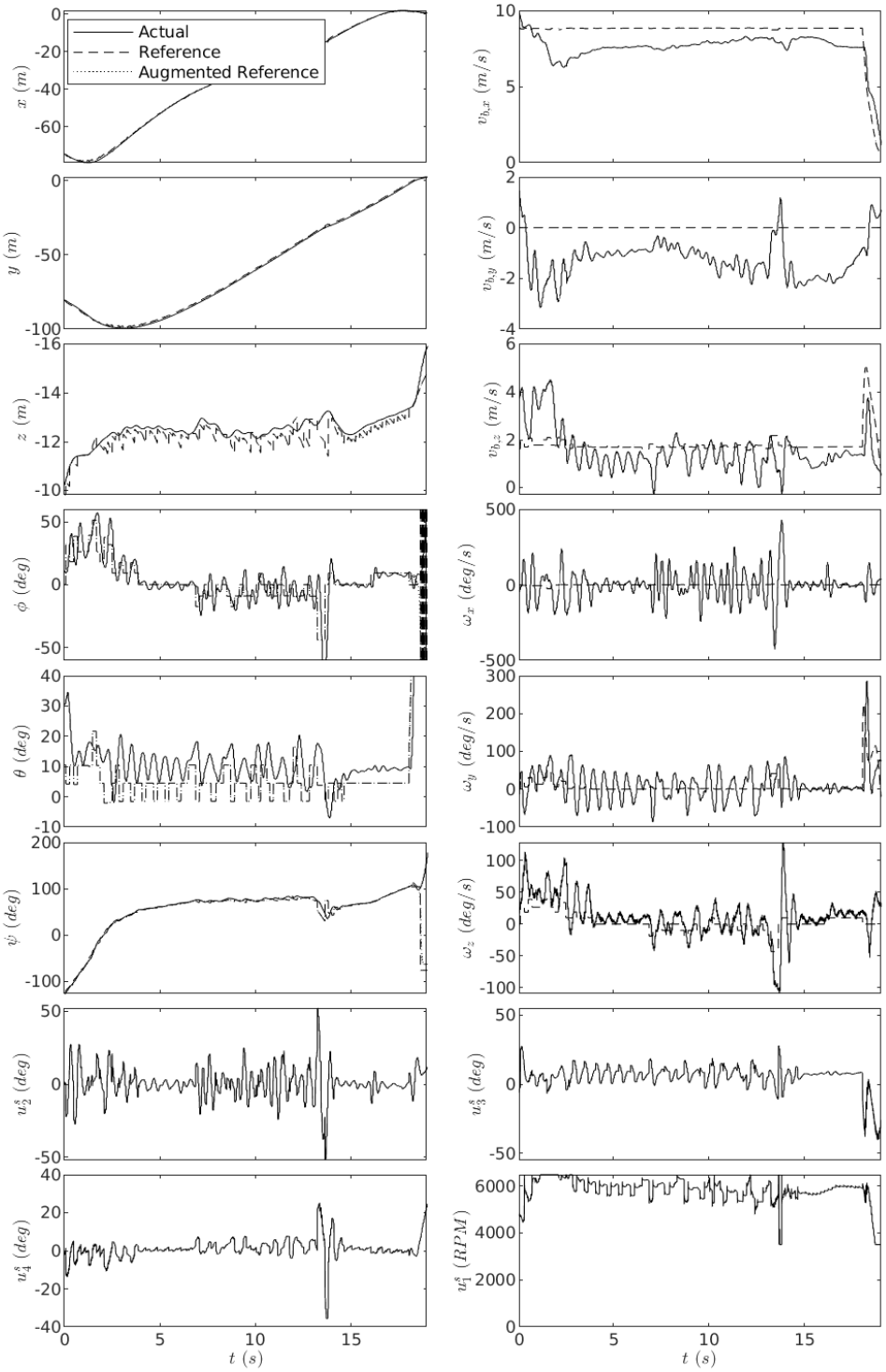


Figure 48. Run 22: state estimates, reference states, and control inputs

in Figure 49, and plots are shown in Figure 50. Looking at Figure 49, the aircraft initially detects the tree, and subsequently banks right, shown in the roll in Figure 50. Looking at Figure 49 when $t = 1.6s$, the creek separating the two grass fields causes false positive points in the point cloud. These false positive points cause the aircraft to bank left in order to avoid them, when looking at Figure 50. A large wind gust occurs around $t = 3s$, causing the roll angle to drop to $-100°$, which can be seen by the roll and sideslip speed ($v_{b,y}$) in Figure 50. By $t = 3.5s$, the aircraft recovers from the wind gust and banks right because the tree is not detected at that instant. Looking at Figure 49 at $t = 4.6s$, the aircraft stops banking right because part of the tree goes undetected, and the aircraft thinks flying straight is collision-free. At $5.1s$ enough of the tree is detected and the aircraft decides to bank right for a short period of time until part of the tree is undetected again (seen at $t = 5.8s$) where the aircraft executes a slight left turn. At about $t = 7s$, the aircraft realizes it can no longer maneuver around the tree using the trim trajectory library, and executes an emergency hover. The aircraft is seen hovering at $t = 9.9s$.

7.3.4. Collision (Run 27)

We now analyze in detail a run that ends in a collision caused by a camera failure (Run 27), and show the run in https://youtu.be/AkCVo2Pmp_g. In Figure 51, we show the image from a ground video, a top-down flight trajectory, the on-board color image, and on-board depth image, for the seconds leading up to the collision. For that same duration of time, we show the state estimate, reference states, and control inputs in Figure 52.

During this run, the aircraft flew over the goal but did not enter a hover because it was too high above the goal altitude. After flying over the goal, the aircraft executes a descending right turn to loop back to the goal. We would have expected the aircraft to turn right until it detects the tree, and at that point would then turn left once realizing the right turn is not collision-free. However, as shown in the color image at $22.33s$, the tree is starting to appear on the right side, middle height. As the aircraft progresses, more of that tree is shown in the color images at $22.66s$ and $23s$. However, during all this time, the depth image still does not detect the tree. Our algorithm relies on the depth camera detecting the tree at these times, as it is within the depth range and field-of-view (the field-of-view of the depth is larger than that of color). It is likely the camera does not detect the tree during these times because the branches and ground are a similar color. At $23.33s$ and $23.66s$, parts of the tree have now been detected, but there are still large chunks that are undetected, shown by the black spots in the right side of the depth image. Looking at Figure 52, the aircraft continues selecting banked right turns during that time, which means the undetected parts of the tree were large enough for the aircraft to deem those trajectories collision-free. The aircraft does not decide to enter an emergency hover until after $24s$, and at that point the collision is inevitable.

7.3.5. Dynamic Obstacles

Outside of the 35 runs presented in the preceding subsections, we briefly considered avoiding moving obstacles. The obstacle-avoidance methodology developed in this article is designed for static obstacles. If the main purpose of the algorithm was to avoid dynamic obstacles, the intelligent approach would be to factor in the motion of the obstacles into the collision checking process, and not simply treat a dynamic obstacle as static points in a point cloud. However, since our static obstacle-avoidance algorithm can be executed quickly, we have succeeded in avoiding dynamic obstacles in certain cases. First off, the dynamic obstacle must be approaching the aircraft from within the FOV. A dynamic obstacle approaching the aircraft from outside of its FOV will never be able to be avoided with any algorithm. Second, there are limitations on the speed of the dynamic obstacle. This speed limitation will be a function of where it enters the aircraft's FOV, and its trajectory relative to the aircraft. Finding this limitation is outside the scope of this work, but such a limitation does exist.

We demonstrate dynamic obstacle-avoidance with two examples, the first where we throw a football at the aircraft, and the second where we fly another UAV through the aircraft's intended flight

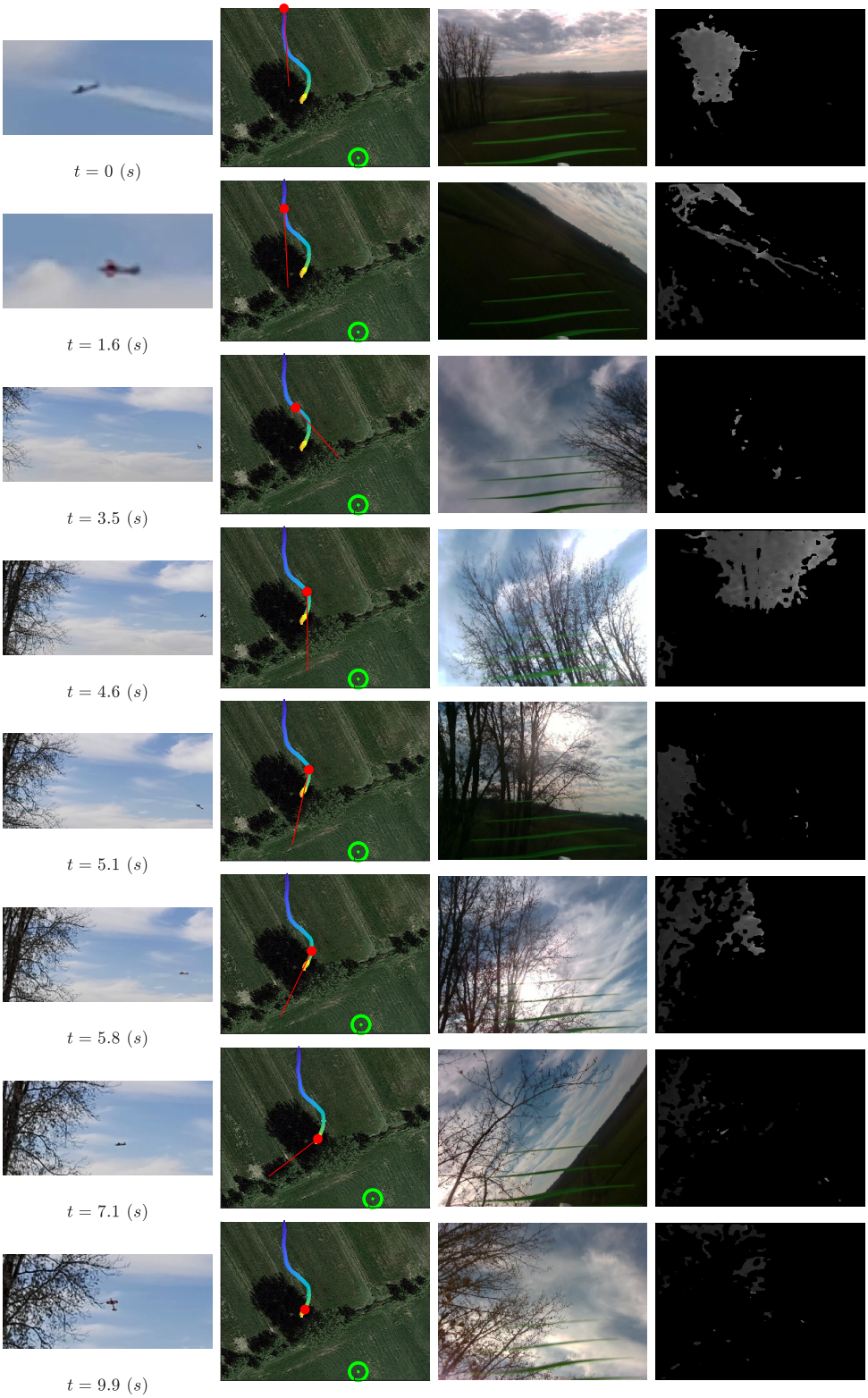


Figure 49. Run 20: ground image, flight trajectory, on-board color image, and on-board depth image

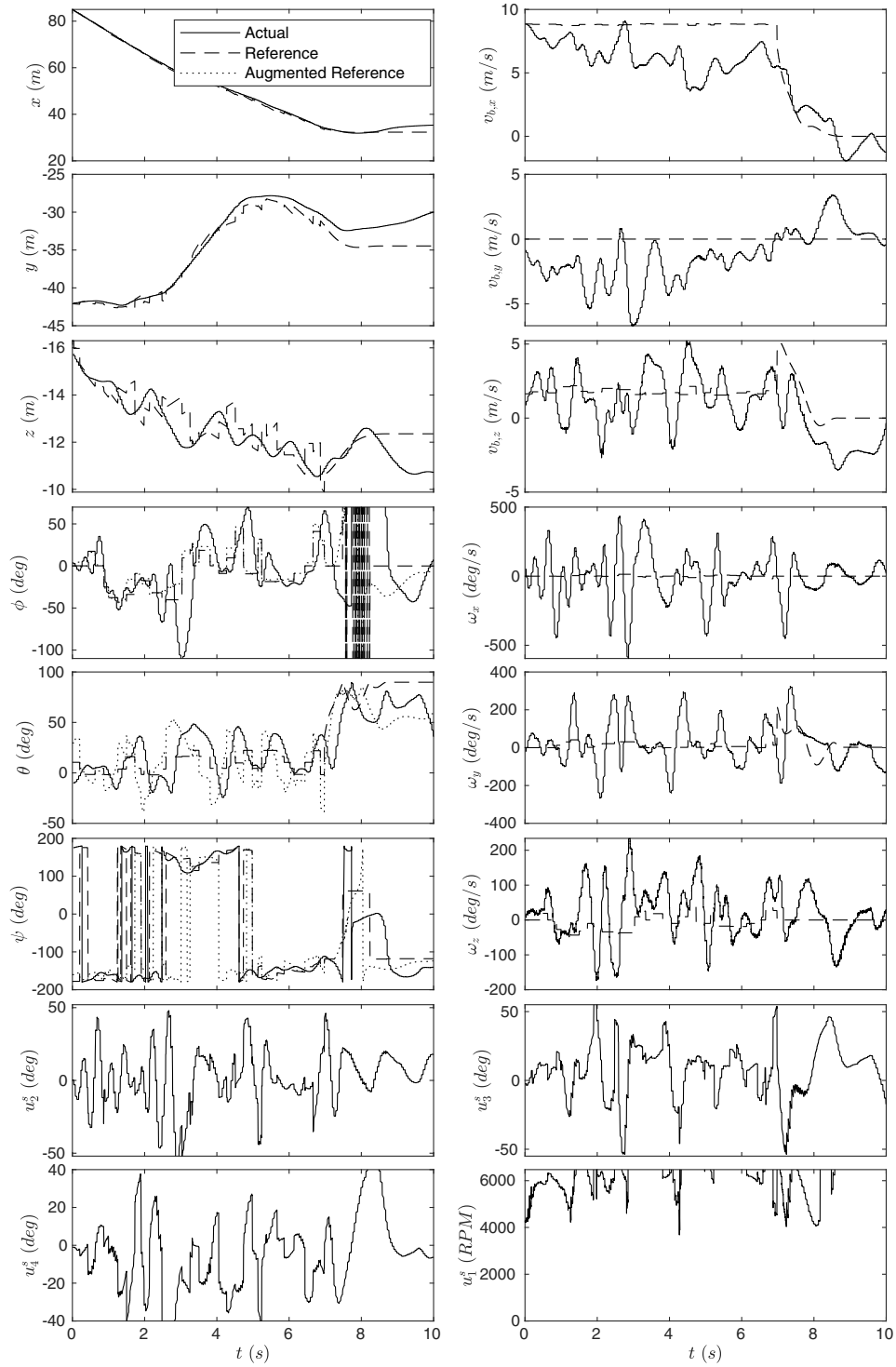


Figure 50. Run 20: state estimates, reference states, and control inputs

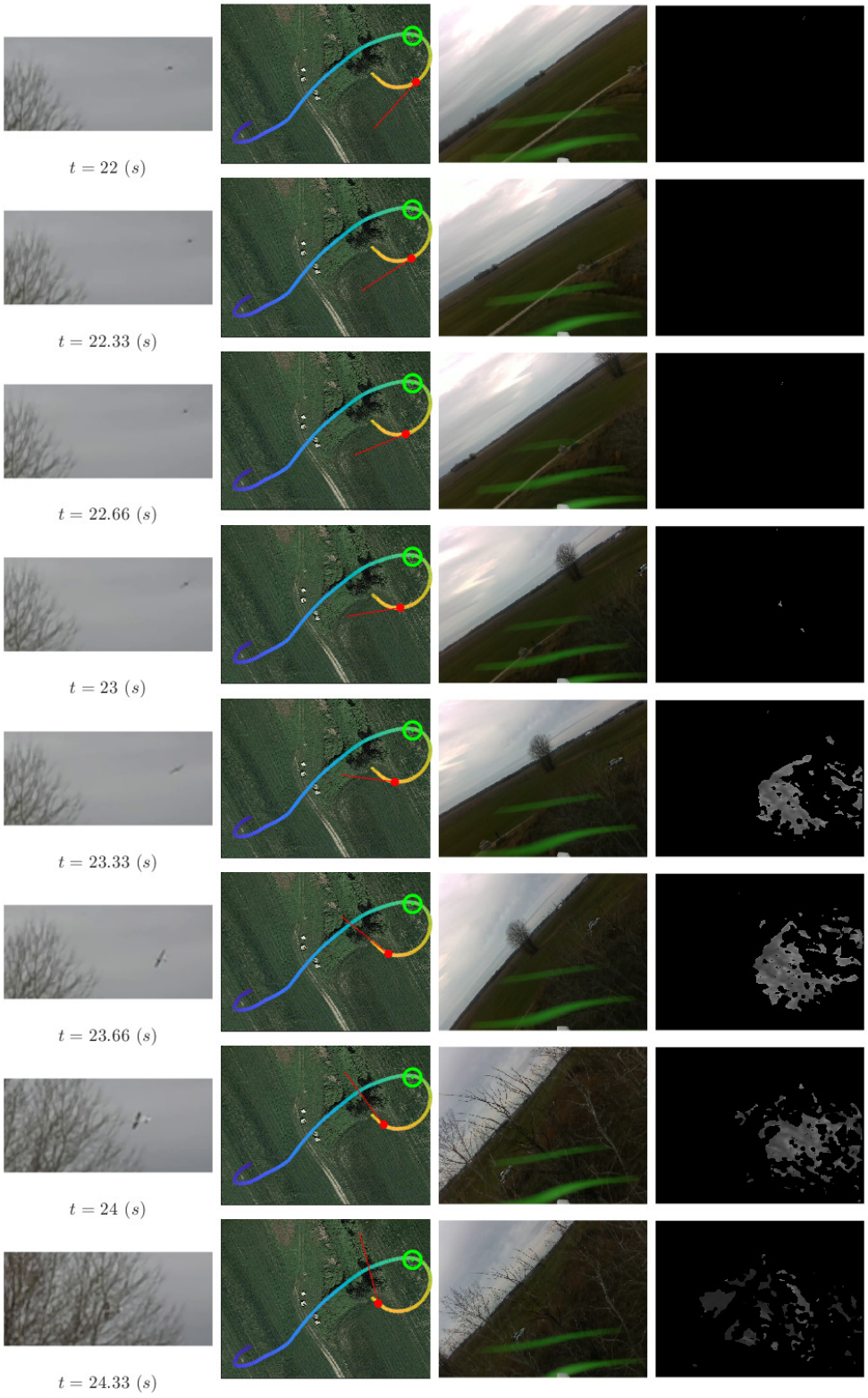


Figure 51. Collision: ground image, flight trajectory, on-board color image, and on-board depth image

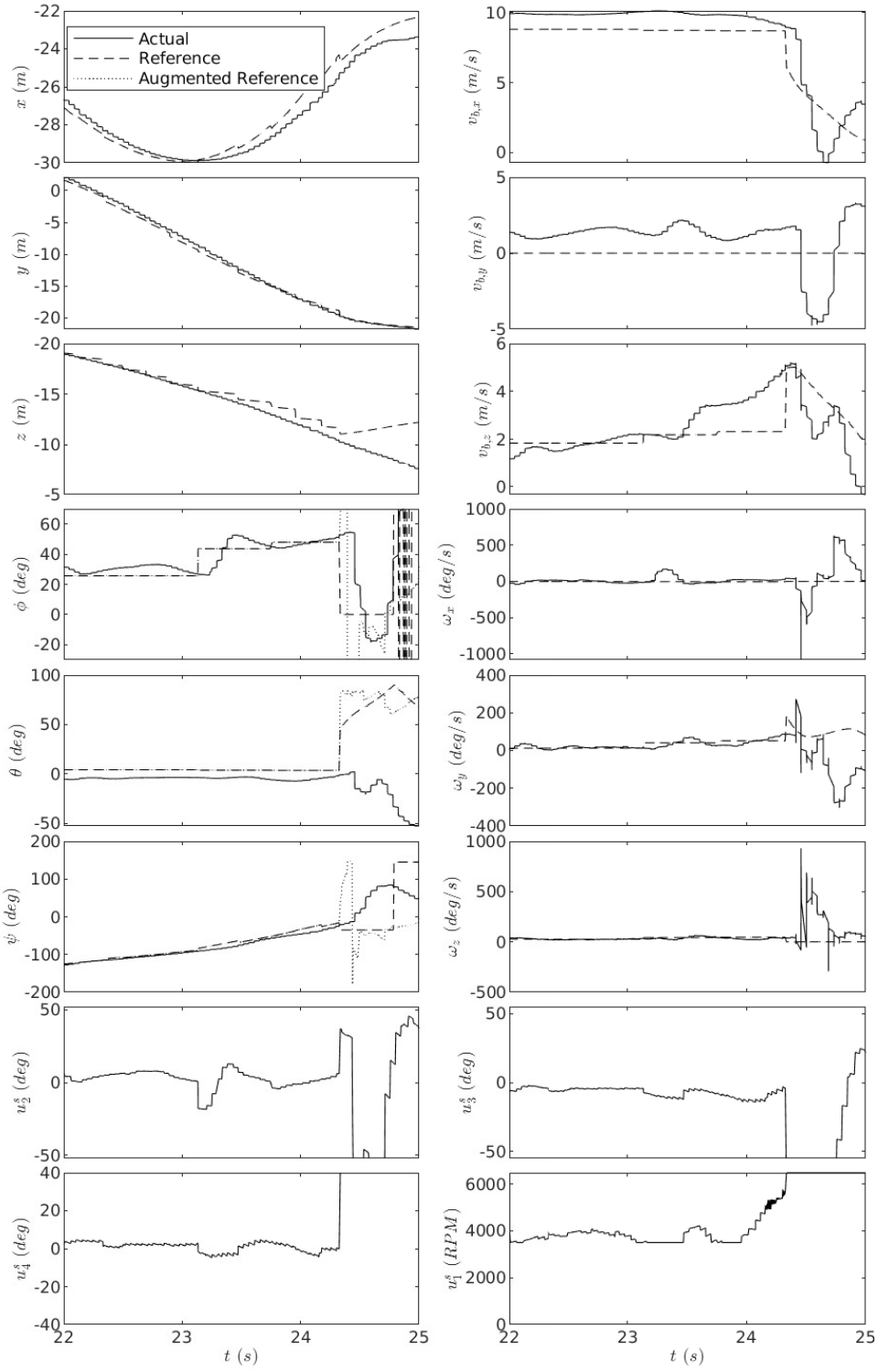


Figure 52. Collision: state estimates, reference states, and control inputs

Figure 53. Avoidance of a Football

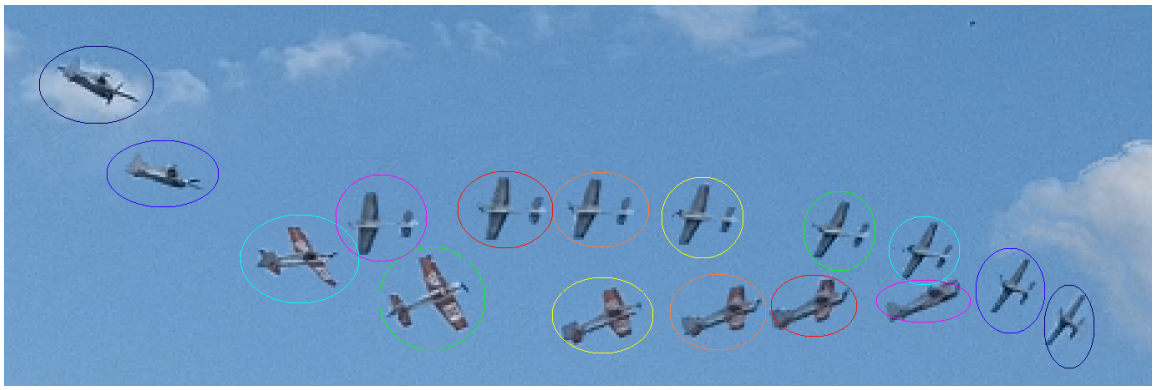


Figure 54. Avoidance of a fixed-wing aircraft

path. The avoidance of a football is shown in Figure 53 and in https://youtu.be/0EADDosTlP8. The football is thrown from the left while the aircraft is initially flying towards it on the right. The trajectories of both the football and aircraft are overlaid on to the image, where the same color circle around the football and aircraft represents the same instant in time. Between the green and yellow times the aircraft has seen the football and decides to turn right to avoid colliding with the football.

The avoidance of an intruder aircraft is shown in Figure 54 and in https://youtu.be/A_AweLnl2FY. The intruder aircraft is initially on the right, and the autonomous aircraft begins on the left. The trajectories of both aircraft are overlaid on to the image, where the same color circle around each aircraft represents the same instant in time. At an instant between the blue and cyan times the autonomous aircraft detects the intruder aircraft, and banks right to avoid it.

8. Conclusion

We have shown it is possible to formulate an obstacle-avoidance algorithm that enables autonomous, high-speed flight of an agile, fixed-wing, UAV in unknown, unstructured environments. Our method only relies on on-board computation and sensing; and can theoretically guarantee collision-free flight.

We first demonstrated autonomous flight through diverse, unknown cluttered environments in simulation. Then we implemented on a test platform, where the algorithm runs in real-time, using only on-board sensing and computation. Using the test platform, we demonstrated autonomous flight in unknown, cluttered, outdoor environments. During flight testing, the aircraft autonomously flew $4.4km$ over $543s$ while avoiding trees, and maintained an average speed of $8.1ms^{-1}$ and a top speed of $14.4ms^{-1}$. During some of these tests, the position controller was not used to avoid obstacles, demonstrating that it is potentially possible for the aircraft to avoid collisions with a loss of GPS signal.

During both simulation and flight testing, the ability for the aircraft to stop and hover in complex scenarios proved critical to avoiding collisions. We believe utilizing the hover is a significant reason we obtained such successful simulations and experiments. By contrast, had the work in (Barry et al., 2018; Barry, 2016) incorporated a similar stopping maneuver, we suspect that some of their collisions could have been avoided. While a hovering fixed-wing aircraft clearly has the flight efficiency similar to a rotor-craft and loses the benefits of the wing, the ability for a vehicle to benefit from a wing for the majority of the flight, and as a last resort, hover to avoid a collision, provides significant benefits for vehicles that are required to fly long distances and at times fly through cluttered environments.

In simulation, the aircraft never collided with an obstacle, while in field experiments the aircraft incurred four collisions out of 35 trials. Obstacle sensing failures were the main cause for collisions, showing that while stereo cameras do provide a obstacle detection solution in the majority of situations, a robust autonomous system should have redundant obstacle-detecting sensors.

Acknowledgments

This work was made possible with financial support from the Natural Sciences and Engineering Research Council (NSERC), the Fonds de Recherche du Quebec—Nature et technologies (FRQNT), a McGill Engineering Undergraduate Student Masters Award (MEUSMA) and by a McGill Engineering Doctoral Award (MEDA).

ORCID

Eitan Bulka https://orcid.org/0000-0002-8596-4428
Meyer Nahon https://orcid.org/0000-0001-6674-2208

References

Barry, A. (2016). *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo*. PhD thesis, Massachusetts Institute of Technology. Department of Electrical Engineering and Computer Science.

Barry, A. J., Florence, P. R., and Tedrake, R. (2018). High-speed autonomous obstacle avoidance with pushbroom stereo. *Journal of Field Robotics*, 35(1):52–68.

Barry, A. J., Majumdar, A., and Tedrake, R. (2012). Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates. In *2012 IEEE International Conference on Robotics and Automation*, pages 484–490.

Basescu, M. and Moore, J. (2020). Direct NMPC for post-stall motion planning with fixed-wing UAVs. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9592–9598. IEEE.

Borenstein, J. and Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288.

Bry, A., Bachrach, A., and Roy, N. (2012). State estimation for aggressive flight in GPS-denied environments using onboard sensing. In *2012 IEEE International Conference on Robotics and Automation*, pages 1–8.

Bry, A., Richter, C., Bachrach, A., and Roy, N. (2015). Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research*, 34(7):969–1002.

Bry, A. and Roy, N. (2011). Rapidly-exploring random belief trees for motion planning under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 723–730. IEEE.

Bulka, E. (2021). *Control and Obstacle Avoidance for Agile Fixed-Wing Aircraft*. PhD thesis, McGill University, Montreal.

Bulka, E. and Nahon, M. (2019a). Automatic control for aerobatic maneuvering of agile fixed-wing UAVs. *Journal of Intelligent & Robotic Systems*, 93(1-2):85–100.

Bulka, E. and Nahon, M. (2019b). High-speed obstacle-avoidance with agile fixed-wing aircraft. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 971–980. IEEE.

Bulka, E. and Nahon, M. (2021). A unified control strategy for autonomous aerial vehicles. *Autonomous Robots*, 45(6):859–883.

Escobar-Alvarez, H. D., Johnson, N., Hebble, T., Klingebiel, K., Quintero, S. A., Regenstein, J., and Browning, N. A. (2018). R-advance: Rapid adaptive prediction for vision-based autonomous navigation, control, and evasion. *Journal of Field Robotics*, 35(1):91–100.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, 25(1):116–129.

Green, W. E. and Oh, P. Y. (2008). Optic-flow-based collision avoidance. *IEEE Robotics & Automation Magazine*, 15(1):96–103.

Hwangbo, M. and Kanade, T. (2013). Maneuver-based autonomous navigation of a small fixed-wing UAV. In *2013 IEEE International Conference on Robotics and Automation, ICRA 2013, May 6, 2013 - May 10, 2013*, Proceedings - IEEE International Conference on Robotics and Automation, pages 3961–3968. Institute of Electrical and Electronics Engineers Inc.

Hwangbo, M., Kuffner, J., and Kanade, T. (2007). Efficient Two-phase 3d Motion Planning for Small Fixed-wing UAVs. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1035–1041.

Intel (2020). *Intel RealSense Product Family D400 Series Datasheet*. Rev. 9.

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

Khan, W. (2016). *Dynamics Modeling of Agile Fixed-Wing Unmanned Aerial Vehicles*. PhD thesis, McGill University, Montreal, Canada.

Khan, W. and Nahon, M. (2013). Toward an Accurate Physics-Based UAV Thruster Model. *IEEE/ASME Transactions on Mechatronics*, 18(4):1269–1279.

Khan, W. and Nahon, M. (2015a). Development and Validation of a Propeller Slipstream Model for Unmanned Aerial Vehicles. *Journal of Aircraft*, 52(6):1985–1994.

Khan, W. and Nahon, M. (2015b). Real-time modeling of agile fixed-wing UAV aerodynamics. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1188–1195.

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

Koyuncu, E., Ure, N. K., and Inalhan, G. (2008). A Probabilistic Algorithm for Mode Based Motion Planning of Agile Unmanned Air Vehicles in Complex Environments. *IFAC Proceedings Volumes*, 41(2):2661–2668.

Koyuncu, E., Ure, N. K., and Inalhan, G. (2010). Integration of Path/Maneuver Planning in Complex Environments for Agile Maneuvering UCAVs. *Journal of Intelligent and Robotic Systems*, 57(1-4):143.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LaValle, S. M. and Kuffner, J. J. (2001). Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, 20(5):378–400.

Levin, J. (2019). *Maneuver Design and Motion Planning for Agile Fixed-Wing UAVs*. PhD thesis, McGill University, Montreal.

Levin, J., Paranjape, A., and Nahon, M. (2018). Motion planning for a small aerobatic fixed-wing unmanned aerial vehicle. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8464–8470. IEEE.

Levin, J. M., Nahon, M., and Paranjape, A. A. (2019). Real-time motion planning with a fixed-wing uav using an agile maneuver space. *Autonomous Robots*.

Lin, Y. and Saripalli, S. (2015). Sense and avoid for unmanned aerial vehicles using ads-b. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6402–6407. IEEE.

Lin, Y. and Saripalli, S. (2017). Sampling-based path planning for UAV collision avoidance. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):3179–3192.

Liu, S., Watterson, M., Tang, S., and Kumar, V. (2016). High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1484–1491. IEEE.

Lopez, B. T. and How, J. P. (2017a). Aggressive 3-d collision avoidance for high-speed navigation. In *ICRA*, pages 5759–5765.

Lopez, B. T. and How, J. P. (2017b). Aggressive collision avoidance with limited field-of-view sensing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1358–1365. IEEE.

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

Meier, L., Honegger, D., and Pollefeys, M. (2015). Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE.

Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., and Pollefeys, M. (2012). Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Auton. Robots*, 33(1–2): 21–39.

Nikolos, I. K., Valavanis, K. P., Tsourveloudis, N. C., and Kostaras, A. N. (2003). Evolutionary algorithm based offline/online path planner for UAV navigation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(6):898–912.

Paranjape, A. A., Meier, K. C., Shi, X., Chung, S.-J., and Hutchinson, S. (2015). Motion primitives and 3d path planning for fast flight through a forest. *The International Journal of Robotics Research*, 34(3):357–377.

Patterson, M. A. and Rao, A. V. (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 Trans. Math. Softw.*, 41(1).

Pohl, D., Dorodnicov, S., and Achtelik, M. (2019). Depth map improvements for stereo-based depth cameras on drones. In *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 341–348. IEEE.

Russell, J. (1996). *Performance and stability of aircraft*. Butterworth-Heinemann.

Ryll, M., Ware, J., Carter, J., and Roy, N. (2019). Efficient trajectory planning for high speed flight in unknown environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 732–738. IEEE.

Scherer, S., Singh, S., Chamberlain, L., and Elgersma, M. (2008). Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, 27(5):549–574.

Schouwenaars, T., How, J., and Feron, E. (2004). Receding horizon path planning with implicit safety guarantees. In *Proceedings of the 2004 American control conference*, volume 6, pages 5576–5581. IEEE.

Tijmons, S., de Croon, G. C., Remes, B. D., De Wagter, C., and Mulder, M. (2017). Obstacle avoidance strategy using onboard stereo vision on a flapping wing mav. *IEEE Transactions on Robotics*, 33(4):858–874.

Tordesillas, J., Lopez, B. T., Carter, J., Ware, J., and How, J. P. (2019). Real-time planning with multi-fidelity models for agile flights in unknown environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 725–731. IEEE.

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

Van Breda, R. (2016). Vector field histogram star obstacle avoidance system for multicopters. Master's thesis, Stellenbosch University, Stellenbosch, South Africa.

Zhao, L. (2015). 3D Obstacle Avoidance for Unmanned Autonomous System (UAS). Master's thesis, University of Las Vegas, Las Vegas, USA.

How to cite this article: Bulka, E., & Nahon, M. (2022). Reactive obstacle-avoidance for agile, fixed-wing, unmanned aerial vehicles. *Field Robotics, 2*, 1507–1566.