Autonomous Thermalling as a Partially Observable Markov Decision Process

Iain Guilliard1
Australian National University
Canberra, Australia
iain.guilliard@anu.edu.au

Richard J. Rogahn
Microsoft Research
Redmond, WA-98052
rrogahn@microsoft.com

Jim Piavis
Microsoft Research
Redmond, WA-98052
v-jimpi@microsoft.com

Andrey Kolobov
Microsoft Research
Redmond, WA-98052
akolobov@microsoft.com

Abstract—Small uninhabited aerial vehicles (sUAVs) commonly rely on active propulsion to stay airborne, which limits flight time and range. To address this, autonomous soaring seeks to utilize free atmospheric energy in the form of updrafts (thermals). However, their irregular nature at low altitudes makes them hard to exploit for existing methods. We model autonomous thermalling as a POMDP and present a receding-horizon controller based on it. We implement it as part of ArduPlane, a popular open-source autopilot, and compare it to an existing alternative in a series of live flight tests involving two sUAVs thermalling simultaneously, with our POMDP-based controller showing a significant advantage.

I. INTRODUCTION

Small uninhabited aerial vehicles (sUAVs) commonly rely on active propulsion to stay airborne. They use motors either directly to generate lift, as in copter-type sUAVs, or to propel the aircraft forward and thereby help produce lift with airflow over the drone’s wings. Unfortunately, motors’ power demand significantly limits sUAVs’ time in the air and range.

In the meantime, the atmosphere has abundant energy sources that go unused by most aircraft. Non-uniform heating and cooling of the Earth’s surface creates thermals — areas of rising air that vary from several meters to several hundreds of meters in diameter (see Figure 1). Near the center of a thermal air travels upwards at several meters per second. Indeed, thermals are used by human sailplane pilots and many bird species to gain hundreds of meters of altitude [3]. A simulation-based theoretical study estimated that under the exceptionally favorable thermalling conditions of Nevada, USA and in the absence of altitude restrictions, an aircraft’s 2-hour endurance could potentially be extended to 14 hours by exploiting these atmospheric updrafts [5, 4].

Researchers have proposed several approaches to enable autonomous thermalling for fixed-wing sUAVs [41, 42, 6, 16, 24, 34, 29]. Generally, they rely on various parameterized thermal models characterizing vertical air velocity distribution within a thermal. In order to successfully gain altitude in a thermal, a sUAV’s autopilot has to discover it, determine its parameters such as shape and lift distribution inside it, construct a trajectory that would exploit this lift, and exit at the right time. In this paper, we focus on autonomously identifying thermal parameters and using them to gain altitude. These processes are interdependent: thermal identification influences

1The author did most of the work for this paper while at Microsoft Research.
its beliefs. This effectively allows the autopilot to build a simulator for a specific thermal in real time “on the fly”, and trade off information gathering to refine it versus exploiting the already available knowledge to gain height. We propose a fast approximate algorithm tailored to this scenario that runs in real time on Pixhawk, a common autopilot hardware that has a 32-bit ARM processor with only 168MHz clock speed and 256KB RAM. On sUAVs with a more powerful companion computer such as Raspberry Pi 3 onboard, our approach allows for thermalling policy generation with a full-fledged POMDP solver. Alternatively, our setup can be viewed and solved as a model-based Bayesian reinforcement learning problem [21].

For evaluation, we added the proposed algorithm to ArduPlane [1], an open-source drone autopilot, and conducted a live comparison against ArduSoar, ArduPlane’s existing soaring controller. This experiment comprised 14 missions, in which two RC sailplanes running each of the two thermalling algorithms onboard flew simultaneously in weak turbulent windy thermals at altitudes below 200 meters. Our controller significantly outperformed ArduSoar in flight duration in 11 windy thermals at altitudes below 200 meters. Our controller Thermals within a thermal. Much of autonomous thermalling literature, to noisy state observations (as in this paper’s setting), it is in a partially observable MDP (POMDP) setting [8]. A POMDP is a tuple \( \langle S, A, T, R, \pi_0 \rangle \) where \( S \) is the set of states, \( A \) is the set of actions available to the agent, \( T : S \times A \times S \rightarrow [0, 1] \) is a transition function specifying the probability that executing action \( a \) in state \( s \) will change the state to \( s' \), \( R : S \times A \times S \rightarrow \mathbb{R} \) is a reward function specifying the agent’s reward for such a transition, and \( \pi_0 \) is the start state. An MDP agent is assumed to know the current state exactly. An optimal MDP solution is a mapping \( \pi : S \rightarrow A \) called policy that dominates all other policies under the expected reward value starting at \( s_0 \):

\[
V^\pi(s_0) = E_{T^\pi} \left[ \sum_{i=0}^{\infty} \gamma^i R(S_i, A_i^\pi, S_{i+1}) | S_0 = s_0 \right] 
\]

Here, \( (x^0, y^0) \) is the position of the thermal center at a given altitude, \( W_0 \) is vertical air velocity, in m/s, at the center, and \( R_0 \) can be interpreted as the thermal’s radius (Figure 1, in red). Note that a thermal’s lift doesn’t disappear entirely more than \( R_0 \) meters from its center. In spite of its simplicity, we use this model in our controller for its low computational cost.

II. BACKGROUND
Thermals and sailplanes. Thermals are rising plumes of air that originate above areas of the ground that give up previously accumulated heat during certain parts of the day (Figure 1). They tend to occur in at least partly sunny weather several hours after sunrise above darker-colored terrain such as fields or roads and above buildings, but occasionally also appear at altitudes up to 300m.

Fig. 2: A full-size DG1000 sailplane (left, photo credit: Paul Hailiday) and Radian Pro remote-controllable model sailplane that serves as our sUAV platform (right).

Thermal updrafts are used by birds [3] and by human pilots flying sailplanes. Sailplanes (Figure 2, left), colloquially also called gliders, are a type of fixed-winged aircraft optimized for unpowered flight, although some do have a limited-run motor. To test our algorithms, we use a Radian Pro sailplane sUAV (Figure 2, right) controllable by a human from the ground.

Thermalling strategies depend on the distribution of lift within a thermal. Much of autonomous thermalling literature, as well as human pilots’ intuition, relies on the bell-shaped model of lift in the horizontal cross-section of a thermal at a given altitude [42]. It assumes thermals to be approximately round, with vertical air velocity \( w(x, y) \) being largest near the center and monotonically getting smaller towards the fringes:

\[
w(x, y) = W_0 e^{-\frac{(x-x^0)^2 + (y-y^0)^2}{2R_0^2}} \tag{1}
\]

Here, \( (x^0, y^0) \) is the position of the thermal center at a given altitude, \( W_0 \) is vertical air velocity, in m/s, at the center, and \( R_0 \) can be interpreted as the thermal’s radius (Figure 1, in red). Note that a thermal’s lift doesn’t disappear entirely more than \( R_0 \) meters from its center. In spite of its simplicity, we use this model in our controller for its low computational cost.

MDPs and POMDPs. Settings where an agent has to optimize its course of action are modeled as Markov Decision Processes (MDP). An MDP is a tuple \( \langle S, A, T, R, \pi_0 \rangle \) where \( S \) is the set of states, \( A \) is the set of actions available to the agent, \( T : S \times A \times S \rightarrow [0, 1] \) is a transition function specifying the probability that executing action \( a \) in state \( s \) will change the state to \( s' \), \( R : S \times A \times S \rightarrow \mathbb{R} \) is a reward function specifying the agent’s reward for such a transition, and \( \pi_0 \) is the start state. An MDP agent is assumed to know the current state exactly. An optimal MDP solution is a mapping \( \pi : S \rightarrow A \) called policy that dominates all other policies under the expected reward value starting at \( s_0 \):

\[
V^\pi(s_0) = E_{T^\pi} \left[ \sum_{i=0}^{\infty} \gamma^i R(S_i, A_i^\pi, S_{i+1}) | S_0 = s_0 \right] 
\]

Here, \( S_i \) and \( A_i^\pi \) are random variables for the agent’s state \( i \) steps into the future and the action chosen by \( \pi \) in that state, under trajectory distribution \( T^\pi \) induced by \( \pi \) from \( s_0 \).

If the agent doesn’t have full state knowledge but has access to noisy state observations (as in this paper’s setting), it is in a partially observable MDP (POMDP) setting [8]. A POMDP is a tuple \( \langle S, A, T, O, Z, \pi_0 \rangle \), where \( S, A, T \), and \( R \) are as in the MDP definition, \( O \) is the observation space of possible clues about the true state, and \( Z : A \times S \times O \rightarrow [0, 1] \) describes the probabilities of these observations for different states \( s' \) where the agent may end up after executing action \( a \). Since a POMDP agent doesn’t know the world state, it maintains a belief — a state probability distribution given the observations received so far. Initially, this distribution is a just a prior \( b_0 : S \rightarrow [0, 1] \); the agent updates it using the Bayes rule. POMDP policies are belief-based and have the form \( \pi : B \rightarrow A \), where \( B \) is the belief space. The optimization criterion amounts to finding a policy that maximizes

\[
V^\pi(b_0) = \sum_s b_0(s) V^\pi(s) \tag{3}
\]

where \( V^\pi(s) \) comes from Equation 2. For a known initial belief state \( b_0 \), general POMDPs can be solved with different degrees of optimality using methods from the point-based family [33] or variants of the POMCP algorithm [36].

Extended Kalman filter. Since a POMDP agent’s action choice depends on its belief, efficient belief updates are
Finally, we correct our intermediate belief for observations: 
\[ \tilde{\mu}' = \tilde{\mu}' + K(\bar{a} - h(a, \tilde{\mu}')) \] 
\[ \Sigma' = (I - KH)\Sigma' \] 
(8)
horizontal displacement. This holds for the model in Equation 1 because that model disregards turbulence.

A note on reference frames. Wind complicates some computations related to sUAV’s real-world state, because the locations of important observations such as lift strength are given in Earth’s reference frame, in terms of GPS coordinates, whereas the sUAV drifts with the wind. The autopilot maintains a wind vector estimate and uses it to translate GPS locations to the air mass’s reference frame [24, 39]. To make wind correction unnecessary, our POMDP model reasons entirely in the reference frame of the air mass. Due to Assumption 2 above, in this frame sUAV’s air velocity fully accounts for the thermal’s displacement w.r.t. the sUAV. When solving the POMDP involves simulating observations, their locations are generated directly in this reference frame too.

POMDP Formulation

The components of the thermalling POMDP are as follows:

State space $S$ consists of vectors $(s^n, s^th)$ describing the joint state of the sUAV $(s^n)$ and the thermal $(s^th)$. In particular, $s^n = (\vec{p}^n, v, \psi, \phi, h)$ and $s^th = (\vec{p}^h, W_0, R_0)$, where $\vec{p}^n$ gives the 2-D location, in meters, of the sUAV w.r.t. an arbitrary but fixed origin of the air mass coordinate system, $v$ is sUAV’s airspeed in m/s, $\psi$ is its heading – the angle between north and the direction in which it is flying, $\phi$ is its bank angle, $\phi$ is its rate of roll, and $h$ is its altitude w.r.t. the mean sea level (MSL). The thermal state vector consists of $\vec{p}^h = (x^th, y^th)$ – the 2-D position of the thermal model center relative to the sUAV position, $W_0$ – the vertical lift in m/s at the thermal center and $R_0$ – the thermal radius (Figure 1).

Wind vector $\vec{w}$ and pitch angle are notably missing from the state space definition due to Assumptions 2 and 3.

Action space $A$ is a set of arc-like sUAV trajectory segments originating at the sUAV’s current position, parameterized by duration $T_A$ common to all actions and indexed by a set of bank angles $\{\phi_1, \ldots, \phi_n\}$. Each trajectory corresponds to a coordinated turn at bank angle $\phi_i$ for $T_A$ seconds. It is not exactly an arc, because attaining bank angle $\phi_i$ takes time dependent on sUAV’s roll rate $\dot{\phi}$ and current bank angle $\phi$.

Transition function $T$ gives the probability of the joint system state $s'$ after executing action $a$ in state $(s^n, s^th)$. Under Assumptions 2 and 4, $T$ is described by the sUAV’s dynamics and Gaussian process noise:

$$T(s, a, s') = \mathcal{N}(s'|f_{T_A}(s, a), Q) \tag{9}$$

Here, $f_{T_A}$ captures the sUAV’s dynamics, $Q$ is the process noise covariance matrix, and $T$ satisfies a crucial property: $T$ does not modify thermal parameters $W_0$, $R_0$, and thermal center $\vec{p}^h$ w.r.t. sUAV’s position $\vec{p}^n$ in $s$. Intuitively, thermal doesn’t change just because the sUAV moves. The change in the thermal center position is purely due to sUAV’s motion and the fact that the thermal’s position is relative to sUAV’s.

Reward function $\mathcal{R}$ for states $s$ and $s'$ is $(h_s - h_{s'})$, the resulting change in altitude. This definition relies on Assumption 3. Without it, thermalling controller could gain altitude by manipulating pitch, not by exploiting thermal lift.

Observation set $O$ consists of the possible sets of sensor readings of airspeed sensor, GPS, and barometer readings that the sUAV might receive while executing an action $a$.

Observation function $Z$ assigning probabilities $Z(u, s', o)$ to various observation sets $o$ is governed by action $a$’s trajectory and Gaussian process noise covariance matrix $R$. Each of sUAV’s sensors operates at some frequency $\xi$. During $a$’s execution, this sensor generates a sequence of datapoints of length $\xi T_A$, each sampled from a 0-mean Gaussian noise around the corresponding intermediate point on $a$’s trajectory. The probability of a set of sensor data sequences produced during $a$’s execution is the product of corresponding Gaussians.

Initial belief $b_0$ is given by a product of two Gaussians:

$$b_0(s) = b_0(s^n)b_0(s^th) = \mathcal{N}(s^n|s^n_0, \Sigma^n_0)\mathcal{N}(s^th|s^th_0, \Sigma^th_0),$$

where $s^n_0$ has $\vec{p}^n_0 = (0, 0)$ for mathematical simplicity, because sUAV’s position is w.r.t. the air mass and for POMDP’s purposes isn’t tied to any absolute location, and $s^th_0$’s other components are sUAV’s current airspeed, roll, etc. $s^th_0$ is initialized with estimates of thermals generally encountered in the area of operations. $\Sigma^n_0$ and $\Sigma^th_0$ are diagonal matrices.

Keeping the belief components for sUAV and thermal state separate is convenient for computational reasons. The sUAV state observations are heavily filtered sensor signals, so in practice sUAV’s state can be treated as known. At the same time, thermal state belief, especially the initial belief $b^th_0$, has a lot of uncertainty and will benefit from a full belief update.

Belief updates are a central factor determining the computational cost of solving a POMDP. Our primary motivation for defining the initial belief $b_0$ to be Gaussian, in spite of possible inaccuracies, is that we can repeatedly use the EKF defined in Equations 4-8 to update $b_0$ with new observations and get a Gaussian posterior $b'$ at every step:

$$b' = \text{EKF}_{\text{update}}(b_0, a, o) \tag{10}$$

The EKF_update routine implicitly uses transition and observation functions $T$ and $Z$ from the POMDP’s definition.

IV. THE ALGORITHM

Although using EKF-based belief updates reduces the computational cost of solving a POMDP, solving an EKF-based POMDP near-optimally on common low-power flight controllers such as APM and Pixhawk is still much too expensive. The algorithm we present, POMDSoar (Algorithm 1), makes several fairly drastic approximations in order to be solve the POMDP model. While they undoubtedly lead to sacrifices in solution quality, POMDSoar still retains a controller’s ability to explore in a (myopically) guided way, which, we claim, gives it advantage in messy thermals at low altitudes. This section presents a high-level description of POMDSoar, while further details about its tradeoffs and implementation are contained in the Appendix [23].

To further reduce belief update cost, POMDSoar always uses the belief mean as the estimate of the sUAV state; see, e.g., line 12 of Algorithm 1. For the thermal part of the state,
Algorithm 1: POMDSoar

1: Input: POMDP $\langle S, A, T, R, O, Z, b_0, N \rangle$ — number of thermal state belief samples, $T_A^{\text{exploit}}$ — planning horizon for exploitation mode, $T_A^{\text{explore}}$ — planning horizon for exploration mode, $\Delta t$ — action trajectory time resolution, THERMAL_CONFIDENCE_THRES — thermal state confidence threshold
2: $b_0 = N(s_0^b, \Sigma_0^b)N(s_0^l, \Sigma_0^l)$ — current belief
3: if $tr(\Sigma_0^b) < \text{THERMAL\_CONFIDENCE\_THRES}$ then
4:     Go to EXPLOIT
5: else
6:     Go to EXPLORE
7: end

EXPLORE begin
8: foreach $a \in A$ do
9:     Traj$_a$ $\leftarrow$ SimActionTraj($s_0^b, \phi_a, \Delta t, T_A^{\text{explore}}$)
10:     foreach $i = 1 \ldots N$ do
11:         $s_i^th$ $\leftarrow$ SampleThermalState($b_0$)
12:         foreach $\bar{\tau} = 1 \ldots \text{Traj}$_a$.Length do
13:             $\bar{a}_{a,i,t} \leftarrow \text{SimObservation}(\text{Traj}_a[t], s_i^th)$
14:             $b_{a,i,t} \leftarrow \text{EKF\_update}(b_{a,i,t-1}, a_{a,t-1}, \bar{a}_{a,i,t})$
15:         end
16:         $\Sigma_{a,i}^th \leftarrow$ covariance of $b_{a,i,\text{Traj}$_a$.Length}$
17:     end
18:     Uncertainty$_a \leftarrow \sum_{i=1}^{N} tr(\Sigma_{a,i}^th)$
19: end
20: $\alpha^* \leftarrow \arg\min_{a \in A} \text{Uncertainty}_a$
21: Return $\alpha^*$

EXPLOIT begin
22: foreach $a \in A$ do
23:     Traj$_a$ $\leftarrow$ SimActionTraj($s_0^b, \phi_a, \Delta t, T_A^{\text{exploit}}$)
24:     foreach $i = 1 \ldots N$ do
25:         $s_i^th$ $\leftarrow$ SampleThermalState($b_0$)
26:         foreach $\bar{\tau} = 1 \ldots \text{Traj}$_a$.Length do
27:             $w_{a,i,t} \leftarrow \text{SimLift}(\text{Traj}_a[t], s_i^th)$
28:         end
29:         AltGain$_{a,i} \leftarrow \sum_{i=1}^{\text{Traj}$_a$.Length} w_{a,i,t} \Delta t$
30:     end
31:     ExpAltGain$_a \leftarrow \sum_{i=1}^{N} \text{AltGain}$_{a,i}$
32: end
33: $\alpha^* \leftarrow \arg\max_{a \in A} \text{ExpAltGain}_a$
34: Return $\alpha^*$

however, it uses the full belief over that part. For a Gaussian, it is natural to take covariance trace as a measure of uncertainty, and this is what POMDSoar does (lines 4, 21).

Conceptually, POMDSoar is split into two parts: exploration and exploitation. Whenever it is invoked, it first analyzes the covariance of the controller’s current belief about the thermal state (line 4). If its trace is below a given threshold — an input parameter — POMDSoar chooses an action aimed at exploiting the current belief. Otherwise, it tries to reduce belief uncertainty by doing exploration. In effect, POMDSoar switches between two approximations of the POMDP reward function $R$, an exploration- and a lift-oriented one.

In either mode, POMDSoar performs similar calculations. Its action set is a set of arcs parametrized by discrete roll angles, e.g., -45, -30, -15, 0, 15, 30, and 45 degrees. For each of them it computes the sequence of points that would result from executing a coordinated turn at that roll angle for $T_A^{\text{explore}}$ or $T_A^{\text{exploit}}$ seconds (lines 12, 29). See the Implementation Details section for more information about this computation.

Then, POMDSoar samples $N$ thermal states from the current belief and for each of these “imaginary therals” simulates the reward of executing each of the above action trajectories in the presence of this thermal. In explore mode, action reward is the reduction in belief uncertainty, so at each of the trajectory points POMDSoar generates a hypothetical observation and uses it in a sequence of imaginary belief updates (line 17). The trace of the last resulting belief measures thermal belief uncertainty after the action’s hypothetical execution. For execution in the real world, POMDSoar chooses the action that minimizes this uncertainty.

In the exploit mode, at each generated point of each action trajectory, POMDSoar measures hypothetical lift (line 33) instead of generating observations, and then integrates the lift along the trajectory to estimate altitude gain (line 35). The action with the maximum altitude gain estimate “wins”.

POMDSoar’s performance in practice critically relies on the match between the actual turn trajectory (a sequence of 2-D locations) the sUAV will follow for a commanded bank angle and the turn trajectory predicted by our controller for this bank angle and used by POMDSoar during planning. Our controller’s trajectory prediction model is in the Appendix [23]. It uses several airframe type-specific parameters that had to be fitted to data gathered by entering the sUAV into turns at various bank angles. The model achieves a difference of < 1m between predicted and actual trajectories. Figure 3 shows the match between the model-predicted and actual bank and aileron deflection angle evolutions for several roll commands.

Our POMDSoar implementation is available at https://github.com/Microsoft/Frigatebird. It is in C++ and is based on ArduPlane 3.8.2, an open-source autopilot for fixed-wing drones [1] that has Tabor et al. [39]’s ArduSoar controller built in. We reused the parts of it that were conceptually common between ArduSoar and POMDSoar, including the thermal tracking EKF and thermal entry/exit logic. ArduSoar and POMDSoar implementations differ only in how they use the EKF to choose the sUAV’s next action. Running POMDSoar onboard a sUAV requires picking input parameters (Algorithm 1, line 1) that result in reasonable-quality solutions within the short (< 1s) time between controller invocations on the flight controller hardware. All parameters values from our experiments are available in a .param file together with POMDSoar’s implementation.
V. RELATED WORK

While our approach can be regarded as solving an POMDP, it is also an instance of Bayesian reinforcement learning (BRL) [18, 21]. In BRL, the agent starts with a prior distribution over model dynamics $T$ and $R$, and maintains a posterior over them as it gets observations. Viewing $T$ and $R$ as part of the augmented state space $S' = S \times T \times R$ reveals that BRL is equivalent to a special POMDP called Bayes-adaptive MDP (BAMDP) [15], which POMDSoar ends up solving. Nonetheless, due to the computational constraints of our platform, POMDSoar is very different from existing POMDP and BRL solvers such as POMCP [36] and BAMCP [22]. However, BRL-induced POMDPs with belief tracking using EKF were studied by Slade et al. [37], and the idea of separating uncertainty reduction from knowledge exploitation for approximate BRL is similar to Dearden et al. [13].

There is vast literature on automatically exploiting thermals [41, 42, 6, 16, 24, 34, 39], orographic lift [27, 19], wind gusts [28], wind fields [32], and wind gradients [31, 9], as well as on planning flight paths to extend fixed-wing sUAV endurance and range [17, 32, 10, 11, 29, 14]. Soaring patterns have also been studied for birds [3]. Since our paper focuses exclusively on thermalling, here we survey papers from this subarea.

In an early autonomous thermalling work, Wharington et al [41, 42] used RL [38] in simulation to learn a thermalling strategy under several simplifying assumptions. Reddy et al. [34] also used RL but removed some of Wharington and Herszberg [42]’s simplifications. In particular, they built a far more detailed thermal model that reflects updrafts’ messy, turbulent nature. However, neither of these techniques have been deployed on real sUAVs so far, and how this could be done is conceptually non-obvious. These works’ version of RL would require executing many trials in the real world in order to learn a thermalling strategy for a given situation, including the ability to restart a trajectory at will — a luxury unavailable to real-world sUAVs. Given the wide variability in atmospheric conditions, it is also unclear whether this method can be successfully used to learn a policy offline for subsequent deployment onboard a sUAV. Our POMDP model circumvents these challenges facing classical RL in the autonomous soaring scenario, since the former allows sampling trajectories during flight on the flight controller’s hardware running a data-driven simulator built on the go. However, incorporating Reddy et al. [34]’s elaborate thermal model into a POMDP-based controller could yield even more potent thermalling strategies.

Allen and Lin [6] were the first to conduct an extensive live evaluation of an automatic thermalling technique. They employed a form of Reichmann rules [6, 35] — a set of heuristics employed for thermal centering by human pilots — to collect data for learning thermal location and radius in flight. Edwards [16] mitigated this approach’s potential issue with biased thermal parameter estimates. In a live flight test conducted primarily at altitudes over 700m, the resulting method along with inter-thermal speed optimization kept a sailplane airborne for 5.3 hours, the record for automatic soaring so far, with thermal centering running on a laptop on the ground. It is not clear whether it is possible to implement it as part of a fully autonomous autopilot on such a computationally constrained device as a Pixhawk, and how such an implementation, if realistic, would perform in less regular low-altitude thermalling conditions. Another controller based on Reichmann rules was introduced by Andersson et al. [7]. Daugherty and Langelaan [12] improved on it by using a Savitzky-Golai filter to estimate total specific energy and its derivatives, which reduced the estimation lag compared to Andersson et al. [7] and allowed latching onto smaller thermals in simulation. Andersson et al. [7]’s controller itself, like those of Edwards [16] and Allen and Lin [6], was evaluated on a real sUAV and did well at altitudes over 400m AGL, but relative live performance of all these controllers is unknown.

Last but not least, our live study compares POMDSoar against ArduSoar [39], the thermalling controller of a popular open-source autopilot for fixed-wing sUAVs called ArduPlane [1]. ArduSoar’s implementation of thermal-tracking EKF and thermal entry/exit logic is shared with POMDSoar. When ArduSoar detects a thermal, it starts circling at a fixed radius around the EKF’s thermal position estimate mean. ArduSoar’s thermal tracking was inspired by Hazard [24]’s work, which used a UKF instead of EKF for this purpose and evaluated a number of fixed thermalling trajectories in simulation.

Thus, several aspects distinguish our work from prior art. On the theoretic level, we frame thermalling as a POMDP, which allows principled analysis and practical onboard solutions. We also conduct the first live side-by-side evaluation of two thermalling controllers. Last but not least, this evaluation is done in an easily replicable experimental setup.

VI. EMPIRICAL EVALUATION

Our empirical evaluation aimed at comparing POMDSoar’s performance as part of an end-to-end sUAV autopilot to a publicly available thermalling controller for sUAVs, ArduSoar, in challenging real-life thermalling conditions. We did this via side-by-side live testing of the two soaring controllers. The choice of ArduSoar as a baseline is motivated by several considerations. First, despite its simplicity, it performs well at higher altitudes. Second, it incorporates central ideas from several other strong works, including Allen and Lin [6], Edwards [16], and Hazard [24]. Third, the availability of source code, documentation, parameters, and hardware for it eliminates from our experiments potential performance differences due to hardware capabilities, implementation tricks, or mistuning.

A field study of a thermalling controller is complicated by
exogenous factors such as weather, so before presenting the results we elaborate on the evaluation methodology.

**Equipment.** We used two identical off-the-shelf Radian Pro remote-controllable sailplanes as sUAV airframes. They are made of styrofoam, have a 2m wingspan, 1.15m length, and carry a motor that can run for a limited duration. To enable them to fly autonomously, we reproduced the setup from Tabor et al. [39], installing on each a 3DR Pixhawk flight controller (32-bit 168MHz ARM processor, 256KB RAM, 2MB flash memory), a GPS, and other peripherals, as shown in Figure 4. A human operator could take over control at will using an X9D+ remote controller. All onboard electronics and the motor was powered by a single 3-cell 1300 mAh LiPo battery.

![Radian Pro sUAV's electronic equipment](image)

**Software.** Both Radian Pros ran the ArduPlane autopilot modified to include POMDSoar, as described in the Implementation Details section of the Appendix [23]. During each flight, POMDSoar was enabled on one Radian Pro, and ArduSoar was enabled on the other. We used Mission Planner v1.3.49 [2] as ground control station (GCS) software to monitor flight telemetry in real time. ArduPlane was tuned separately on each airframe using a standard procedure [1]. The parameters not affected by tuning were copied from Tabor et al. [39]'s, setup, including the target airspeed of 9 m/s. The airspeed sensors were recalibrated before every flight.

**Location.** Flights took place at two test sites denoted Valley and Field located in the Northwestern USA around 47.6°N, 122°W approximately 16km apart, each ≈ 700m in diameter. Figure 5 shows their layout, waypoint tracks, and geofences.

**Flight conditions.** All flights for this experiment took place between October 2017 and January 2018. Regional weather during this season is generally poor for thermalling, with low daily temperatures and their amplitudes (a major factor for thermal existence and strength), nearly constant cloud cover, and frequent gusty winds and rain. All flights were carried out in dry but at best partly cloudy weather, at temperatures ≤ 14°C and daily temperature amplitudes ≤ 9°C. All took place in predominant winds between 2 and 9 m/s; in ≈ 25% of the missions, the wind was around 7 m/s – 78% of our sUAVs’ 9 m/s airspeed during thermalling and off-motor glides.

**Flight constraints.** Due to regulations, all flights were geofenced as shown in Figure 5 and restricted to low altitudes: 180m AGL at the Valley and 160m AGL at the Field site. To avoid collision with ground obstacles, the minimum autonomous flight altitude was 30m and 50m AGL, respectively.

**Mission profile.** Each mission consisted in two Radian Pros, one using POMDSoar and another using ArduSoar as the thermalling controller, taking off within seconds of each other and repeatedly flying the same set of waypoints at a given test site counterclockwise from the Home location as long as their batteries lasted, deviating from this path only during thermal encounters. Figure 6a explains the mission pattern.

Each flight was fully autonomous until either the sUAV’s battery voltage, as reported in the telemetry, fell below 3.3V/cell for 10s during motor-off glide, or for 10s was insufficient to continue a motor-on climb. At that point, the mission was considered over. A human operator would put the sUAV into the Fly-By-Wire-A mode and land it.

**Eliminating random factors.** Live evaluation of a component as part of an end-to-end system is always affected by factors exogenous to the component. The Appendix [23] details the measures we took to eliminate many of these factors, including the presence of non-thermal lift, potential systematic bias due to minor airframe and battery differences, and high outcome variance due to chance of finding thermals.

**Performance measure.** We compare POMDSoar and ArduSoar in terms of the relative increase in flight duration they provide. Namely, for a flight at test site \( S \) where controller \( C^{th} \) ran on airframe \( A \) and used battery \( B \), we compute

\[
RelTimeGain_{A,B,S}(C^{th}) = \frac{FlightTime_{A,B,S}(C^{th})}{BaselineFlightTime_{A,B,S}} \tag{11}
\]

The \( BaselineFlightTime_{A,B,S} \) values are averages over durations of a series of separate flights in calm conditions with thermalling controllers turned off (see the Appendix [23]).

**Results**

Our comparison is based on 14 two-sUAV thermalling flights performed in the above setup. Figure 7 shows the results. Importantly, they are primarily indicative of controllers’ advantages over each other, rather than each controller’s absolute potential to extend flight time, due to altitude and geofence restrictions that often forced sUAVs out of thermals, and the deliberate lack of a thermal finding strategy.

Qualitatively, as Figure 7 indicates, POMDSoar outperformed ArduSoar in 11 out of 14 flights, and did as well in 2 of the remaining 3. Moreover, the flight time gains POMDSoar provides are appreciably larger compared to ArduSoar’s. The data in the “Effect of baseline correction” subsection of the Appendix [23] adds more nuance. It shows that without baseline correction (Equation 11), the results look somewhat different, even though POMDSoar still wins in 11 flights.

These results agree with our hypothesis that in turbulent low-altitude thermals, POMDP-driven exploration and action selection are critical for taking advantage of whatever lift there is. Likely due to active exploration, POMDSoar’s trajectories can be much “messier” than ArduSoar’s; see Figure 5.
The Field test site and POMDSoar’s typical flight path at it.

The Valley test site and ArduSoar’s typical flight path at it.

Fig. 5: The Field and Valley test site layout and typical mission flight paths due to POMDSoar and ArduSoar. When not thermalling (AUTO mode), the sUAVs follow a fixed sequence of waypoints, resulting in a pentagon-shaped path at the Field site and a triangle-shaped one at the Valley. During thermalling, the sUAVs are allowed to deviate from this path anywhere within the geofenced region shown in grey. In the thermalling mode (FBWB for POMDSoar, LOITER for ArduSoar, green sections of the paths), POMDSoar does a lot of exploration, yielding irregularly-shaped meandering trajectories. ArduSoar, due to a more rigid thermalling policy, yields spiral-shaped paths that result in little exploration. Both controllers are forced to give up thermalling and switch to AUTO mode if the sUAV breaches the geofence.

Altitude plot for a typical baseline measurement flight.

Fig. 6: Altitude vs. time plots for thermalling and baseline flights illustrating the mission profile. After being hand-tossed into the air in AUTO mode, each Radian automatically started its motor and climbed to the altitude given by SOAR_ALT_CUTOFF. At that altitude, it would shut down the motor and glide down, still in AUTO mode, until it either reached SOAR_ALT_MIN or detected a thermal. In the latter case, it would abandon waypoint following and enter a thermalling mode (FBWB for POMDSoar, LOITER for ArduSoar) until it gained altitude up to SOAR_ALT_MAX, descended to SOAR_ALT_MIN if the thermalling attempt was unsuccessful, or hit the geofence. In all these cases, the AUTO mode would engage automatically, forcing the sUAV to give up thermalling, guiding it to the next waypoint, and turning on the motor if necessary to climb from SOAR_ALT_MIN to SOAR_ALT_CUTOFF. For Field and Valley test sites, SOAR_ALT_MIN is 50m and 30m, SOAR_ALT_CUTOFF is 110m and 130m, and SOAR_ALT_MAX is 160m and 180m, respectively.

Fig. 7: Relative gains in flight time over the baseline (Equation 11) for POMDSoar and ArduSoar across 14 flights at the Valley (V) and Field (F) test sites. POMDSoar outperformed ArduSoar overall, yielding higher gains in 11 flights, losing in 1, and ending 2 in a draw.

VII. CONCLUSION

This paper has presented a POMDP-based approach to autonomous thermalling. Viewing this scenario as a POMDP has allowed us to analyze it in a principled way, to identify the assumptions that make our approach feasible, and to show that this approach naturally makes deliberate environment exploration part of the thermalling strategy. This part is missing from prior work but is very important for successful thermalling in turbulent low-altitude conditions, as our experiment have demonstrated. We have presented a light-weight thermalling algorithm, POMDSoar, deployed it onboard a sUAV equipped with a computationally constrained Pixhawk flight controller, and conducted an extensive field study of its behavior. Our experimental setup is easily replicable and minimizes the effect of external factors on soaring controller evaluation. The study has shown that in challenging low-altitude thermals, POMDSoar outperforms the soaring controller included in Arduplane, a popular open-source sUAV autopilot. Despite encouraging exploration, due to Pixhawk’s computational constraints POMDSoar makes many approximations. However, companion computers on larger sUAVs may be able to run a full-fledged POMDP solver in real time. Design and evaluation of a controller based on solving the thermalling POMDP near-optimally, e.g., using an approach similar to Slade et al. [37]'s, is a direction for future work.

Acknowledgements. We would like to thank Eric Horvitz, Chris Lovett, Shital Shah, Debadeepta Dey, Ashish Kapoor, Nicholas Lawrance, and Jen Jen Chung for thought-provoking discussions relevant to this work.

REFERENCES


