Innovations Syst Softw Eng (2005)
DOI 10.1007/s11334-005-0017-x

ORIGINAL PAPER

Jonathan Sprinkle · Aaron D. Ames
J. Mikael Eklund · Ian M. Mitchell
S. Shankar Sastry

Online safety calculations for glide-slope recapture

Received: 24 April 2005 / Accepted: 12 June 2005 / Published online: 29 July 2005

Abstract As unmanned aerial vehicles (UAVs) increase in popularity and usage, an appropriate increase in confidence in their behavior is expected. This research addresses a particular portion of the flight of an aircraft (whether autonomous, unmanned, or manned): specifically, the recapture of the glide slope after a wave-off maneuver during landing. While this situation is rare in commercial aircraft, its applicability toward unmanned aircraft has been limited due to the complexity of the calculations of safety of the maneuvers. In this paper, we present several control laws for this glide-slope recapture, and inferences into their convergence to the glide slope, as well as reachability calculations which show their guaranteed safety. We also present a methodology which theoretically allows us to apply these offline-computed safety data to all kinds of unmanned fixed-wing aerial vehicles *while online*, permitting the use of the controllers to reduce wait times during landing. Finally, we detail the live aircraft application demonstration which was done to show feasibility of the controller, and give the results of offline simulations which show the correctness of online decisions at that demonstration.

Keywords Unmanned aerial vehicles (UAVs) · Reachability model analysis · Controller synthesis · Code generation

This work was sponsored by the Large National Science Foundation Grant for Information Technology Research (NSF ITR) "Foundations of Hybrid and Embedded Software Systems", Award #0225610, and by Defense Advanced Research Projects Administration (DARPA) "Software Enabled Control" (SEC) Program, under contract #F33615-98-C-3614.

J. Sprinkle (✉) · A.D. Ames · J.M. Eklund · S. Sastry
Dept. of Electrical Engineering & Computer Sciences
231 Cory Hall
University of California, Berkeley
Berkeley, CA, 94720-1770 USA
Tel.: +1-510-6435798
Fax: +1-510-6432356
E-mail: {sprinkle,adames,eklund,sastry}@EECS.Berkeley.Edu

I.M. Mitchell
Computer Science Department
ICICS/CS 217
University of British Columbia
Vancouver, BC, V6T 1Z4, Canada
Tel.: +1-604-8222317
Fax: +1-604-8225485
E-mail: mitchell@cs.ubc.ca

1 Introduction

Aircraft control is extremely complex. The model of the nonlinear behavior of an aircraft is a subject usually left to senior undergraduate or graduate education—the ability to exercise *control* on this complex model with any degree of certainty is still emerging research for many control applications. When the effects of wind, altitude, humidity, and payload (in terms of mass, inertia, and drag) are added to the model of the system, the ability to consider a complete system model for all kinds of control is nontrivial.

1.1 The software-enabled control program

To decrease the complexity and difficulty of these prospects, aircraft control uses the common software concept of *information hiding* to manage the layers of control. These basic layers of control are shown in Fig. 1, which has been adapted from [9] to include general control.

The difficulty of guaranteed control for some aeronautical maneuvers and applications, especially in aeronautical systems, prompted the development of the Software Enabled Control (SEC) program, in the Defense Advanced Research Projects Administration Information eXploitation Office (DARPA/IXO). The overall objective of SEC was to provide control technologies which would advance the state of the art for both manned and unmanned vehicles (see `http://www.darpa.mil/`).

Here, the inability to develop control laws for some systems using traditional control methods guided the development of alternative methods which were based on software, but which nonetheless required satisfaction of functional,

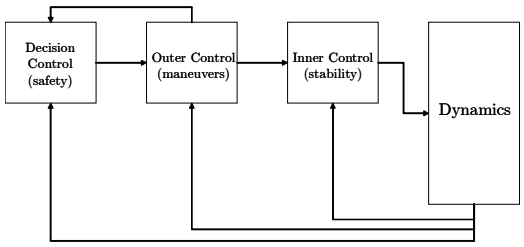


Fig. 1 The basic layers of control for a fixed-wing aircraft (adapted from [9]). Our contribution lies mainly in the decision controller, with inputs to the outer control

performance, and reliability constraints along the following axes:

- **Stability**: *Will the system be stable during execution?*
- **Safety**: *Will the system be safe during execution?*
- **Optimality**: *Is a heuristic acceptable, or is an optimal solution required?*
- **Compositionality**: *Does the coupling of two controllers require additional guarantees?*
- **Robustness**: *Do small perturbations make the system unstable?*
- **Provability**: *Is the software algorithm or decision provably correct?*
- **Decidability**: *Is it possible to give a decision in terms of one of these constraint axes?*
- **Time-criticality**: *Does software execution time affect the behavior of the system?*

It is questions like these that the SEC program was interested in answering.

1.2 Motivation

Many of the tasks which are carried out by humans, such as landing, takeoff, refueling, etc., are generally held in high confidence by crew, ground controllers (and, as appropriate, passengers) because of the rigorous training process through which pilots pass. As pilots make decisions regarding the stability and safety of the aircraft in these more difficult situations (Here we specifically compare the difficulty of landing to that of steady-state flight, as described in [16].) they are influenced by this training and experience.

For unmanned vehicles, this presents something of a problem, since for manned vehicles the situation awareness[1] of the pilot influences decisions that are made. It is nontrivial to separate this "seat-of-the-pants" knowledge from instrumental and other knowledge, and thus provide a high-confidence assessment of what an autonomous or remotely controlled vehicle is doing during risky maneuvers.

This problem becomes even more difficult when guarantee of data is not present for remote control of the aircraft by a pilot on the ground. In this case, time-critical decisions which require a human in the loop regarding the aircraft behavior (e.g., what waypoint to follow next, whether or not it is safe to land) should be made whenever possible by a *decision computer*, which weighs the state of the aircraft against desired states, known control laws, and stability, safety, optimality, compositionality, robustness, provability, time-criticality, and constraints.

[1] Situation awareness is the mental model of the "system" by a pilot, and is the subject of research in ergonomics [1] and aviation psychology, as well as important to the safety of the aircraft [6].

1.3 Overview

This paper presents a general framework and aircraft-specific solution for a particular kind of decision computer: glide-slope recapture. Section 2 defines the general problem, its boundary conditions, and the way in which the decision computer is used to solve the problem. It also provides some background material on the history of this problem, previous work that has been done, and why the issue is becoming more important for the domain of unmanned vehicles.

Section 3 gives several candidate control laws that we apply to maintain and recapture a glide slope. In this section we address the stability, optimality, and provability questions which might be asked of the controller.

Section 4 operates with control laws and aircraft-specific properties to address the safety and robustness of the system, using backward-reachable sets. In this section we give a brief background and introduction to the technology and complexity of calculating these reachable sets, and techniques which can be used to combine safe sets and subtract unsafe sets to yield more refined safety and robustness claims.

Section 5 details how it is possible to take the calculated reachable sets from Sect. 4 and apply those raw data directly to the synthesis of a decision controller, which is a generated executable that can be run on an onboard computer. In this section we give details which the reader may wish examine to employ the technique to avionics testbeds, and we address issues of compositionality and time-criticality which must be answered to maintain the safety and stability claims made in Sects. 3 and 4.

Section 6 describes the SEC Capstone Demonstration, in which we controlled a Boeing T-33 trainer jet using our generated software through its autopilot interface.

Section 7 gives analysis of the flight demonstration, as well as several a posteriori comparisons of the control law which was used during the flight demonstration, to other control laws described in Sect. 3. We also discuss in this section the fundamental differences in the reachable sets we used for our online flight demonstration (and previously presented in [15]) and those which we develop in Sect. 4. In this section we also give thought to the overall contribution of this research, and its possible impact upon policies for automated landing in the future.

Section 8 describes future work we envision for this research in software engineering terms, as well as the electrical engineering and computational issues which remain either unsolved or infeasible, and are in need of future attention. Finally, we give concluding remarks in Sect. 9.

2 Problem description

This section gives some motivation for and provides technical and informal descriptions of the *glide-slope recapture* problem, which is an interesting subproblem of multi-vehicle landing scheduling. This problem is also explained in [15], which presents the applied accomplishment of the authors in that particular problem, and promise for future work. This paper delves deeper into the problem, and provides more context for future solutions, as well as reasonings about the application demonstration which have not previously been discussed. For the sake of continuity of this paper, we repeat here some of the descriptions and story which are given in [15].

2.1 Historical perspective

The first automatic fixed-wing aircraft landing took place using a manned vehicle controlled by an onboard autopilot on 23 August 1937. This pre-World War II proof of concept was accomplished due to the ingenuity and hard work of several driven military personnel, as well as a competent electronics engineer. Also, it used a clear runway where it was the only expected landing. In fact, for years after this event the landing of multiple aircraft was treated as a sequential set of individual landings, where each plane was not allowed to approach the runway until the plane before it had landed and taxied off the runway. It was not until the large demand for food and supplies which motivated the Berlin airlift [3] that this problem had to be solved; without this pipelining (as it is termed in computer hardware and chip design) it would not have been possible to meet food and supply needs.

The experience, protocol, and technologies devised to solve this problem for the Berlin airlift helped to enable commercial airports all around the world handle the large quantity of air traffic which exists today. This general problem of scheduling aircraft-landing times and ordering is managed nonoptimally by air traffic control (ATC). Please refer to Bayen et al. [2] for more insight into the difficulty of, and some heuristical solutions to, this problem. Formally, the problem can be broken down into several subproblems, which are solved by different levels of control. We leave the long-range planning and details up to the interested reader, and instead concentrate on the problem as it affects us.

2.2 Problem formalization

2.2.1 $\mathcal{P}_{\text{many}}$ under normal conditions

Here we will examine the uncomplicated problem, that is, the problem without added complications of faults or disturbances, of landing a large quantity of fixed-wing aircraft on one runway, which we will call the $\mathcal{P}_{\text{many}}$ problem.[2]

[2] This section is taken largely from [15].

The $\mathcal{P}_{\text{many}}$ problem is the problem that ATC solves every day, which is to provide a time-spaced ordering for N aircraft which are on arrival to a runway. Once an aircraft ordering is provided (i.e., once $\mathcal{P}_{\text{many}}$ has been solved) then each aircraft, $p_1 \dots p_N$, obeys the spacing commands of ATC to put themselves on the glide slope, and maintains the velocity, heading, and descent commands which are the solution to the $\mathcal{P}_{\text{one}}$ problem. Figure 2a shows this phenomenon pictorially.

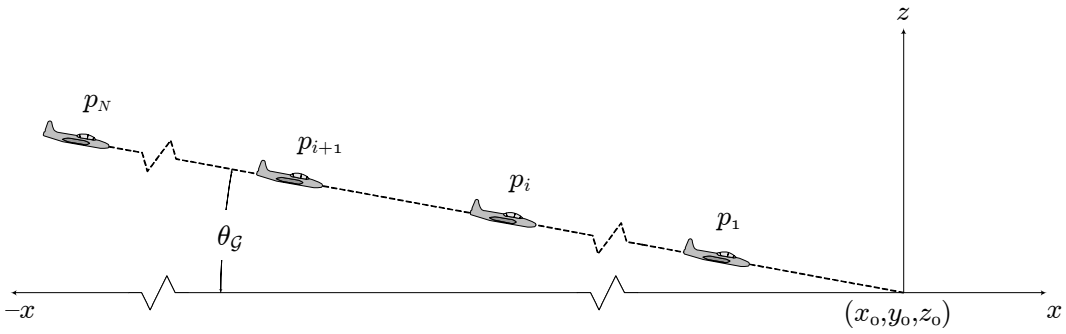
A note on aircraft diagrams. Aeronautical experts will note that the angle of attack, α, and sideslip angle, β, are not shown or mentioned in our body dynamics. Since our onboard systems do not actually give us the angular measurements of the body but simply the direction of translational motion, we absorb these values into the translational motion, and display *all* aircraft translational motion with α and β as effectively 0°. This will result in diagrams such as Fig. 2a, which present a view of the aircraft which is nonrepresentative of how the aircraft would actually appear in reality. This phenomenon does not affect our controller, since our autopilot controller enforces our commanded descent and turn rates and will thus reproduce the necessary α and β to produce lift during landing.

The $\mathcal{P}_{\text{one}}$ problem is that which lands some plane, p_i, as if it were the only plane scheduled to land—that is, any deviation from the control as defined in the $\mathcal{P}_{\text{one}}$ solution requires human intervention. This is what we call the $\mathcal{P}_{\text{next}}$ problem, in this paper.

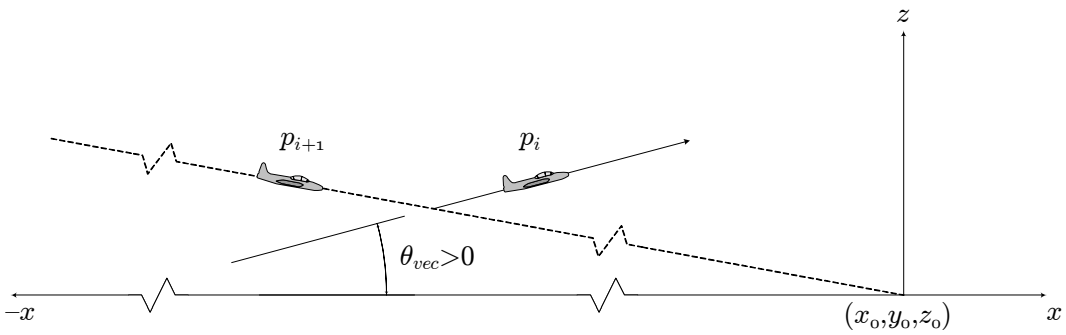
2.2.2 $\mathcal{P}_{\text{next}}$: a fault-susceptible problem

Between the large problem of managing the arrival and landing of large quantities of planes (we will call this $\mathcal{P}_{\text{many}}$), and the small problem of landing one plane ($\mathcal{P}_{\text{one}}$), there is the problem of landing *the next* plane, ($\mathcal{P}_{\text{next}}$). The $\mathcal{P}_{\text{next}}$ problem stems from the requirement that while plane p_i is landing, plane p_{i+1} is already on its glide slope. The glide slope is the position vector, angle of descent, velocity, and attitude of the plane as it "glides" in for its landing to arrive at some predefined point in space. Assuming that there is sufficient spacing between the planes (i.e., $\mathcal{P}_{\text{many}}$ has been solved), then each plane's $\mathcal{P}_{\text{one}}$ solution is sufficient to land that plane. However, $\mathcal{P}_{\text{next}}$ emerges as a problem when some fault occurs during or directly after the landing of p_i, affecting the ability of p_{i+1} to use the runway to land.

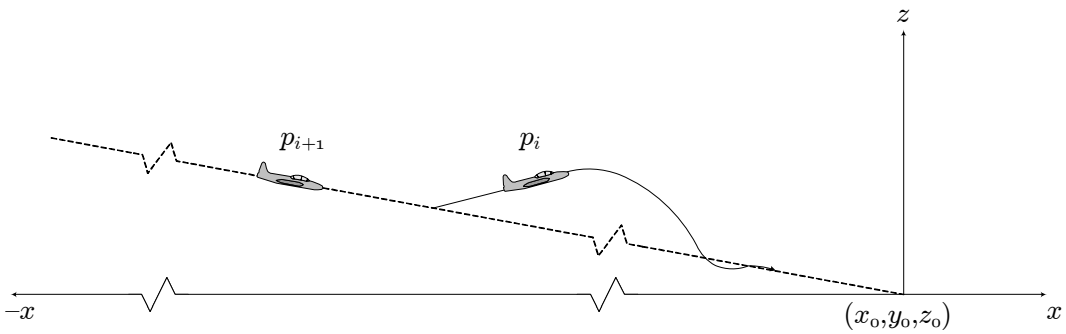
When some fault emerges which puts into question the safe landing of p_{i+1}, ATC (solving the $\mathcal{P}_{\text{many}}$ problem) will usually instruct p_{i+1} to *vector-off* and fly a go-around maneuver. Figure 2b visually depicts this possibility. This means that p_{i+1} will *immediately* depart from the glide slope, and choose a trajectory which will send it to the end of the queue of landing planes (or fly a holding pattern) and try to land again. This requires ATC to resolve the $\mathcal{P}_{\text{many}}$ problem and reinsert p_{i+1} into the queue, perhaps influencing other aircraft already lined up, depending on the fuel or other constraints of the other aircraft.



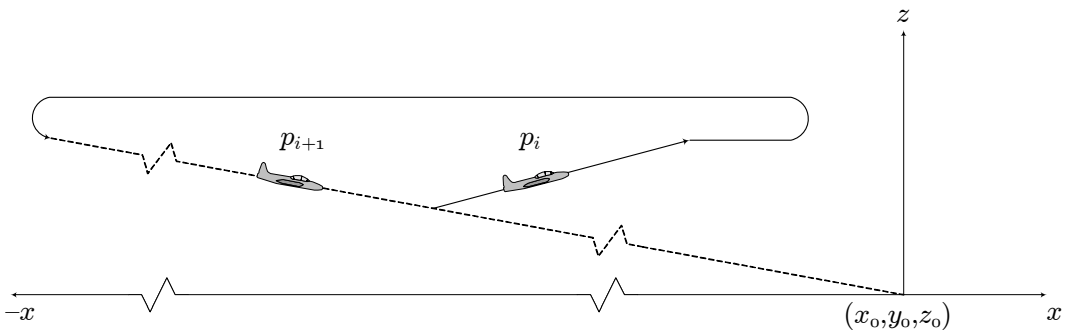
(a)



(b)



(c)



(d)

Fig. 2 Initial departure from the glide slope by p_i (Fig. 2 b), and possible maneuvers if the "Land" command is given by the ground once significant departure from the glide slope has taken place (Fig. 2 c, d). Each of these is shown in the x–z plane for ease of understanding, though planes which vector-off are generally directed out of this plane by some angle ψ_{vec}. **a** A queue of $p_1 \ldots p_N$ planes coming in to land, with glide slope $\theta_{\mathcal{G}}$. **b** Departure of aircraft p_i from the glide slope, as requested by ground control. **c** Recapture of aircraft p_i to the glide slope, using the recapture controller according to the $\mathcal{P}_{\text{next}}$ formulation. **d** Aircraft p_i uses the "go-around" control law, and circles back into the landing set.

2.2.3 $\mathcal{P}_{\text{retry}}$: *glide-slope recapture*

Occasionally, the disruption on the runway will be solved *before* the p_{i+1} plane has turned around, but *after* it has already left the glide slope. The question then becomes, *can* p_{i+1} *recapture the glide slope*? If it can, then the time to land the queue of planes does not increase, and the optimality of the solution will likely not decrease. This is considered to be the best case for ATC, since resolving the $\mathcal{P}_{\text{many}}$ problem is highly unlikely, though not impossible, as we discuss next.

There is some probability that p_{i+1} will land in the same sequence, but not at the same time as it would have previously. This in turn may cascade down the stack of p_N aircraft and result in some aircraft falling below its minimum speed to maintain lift. We posit that, if this is the case, then ATC will not ask p_{i+1} to retry its landing maneuver.

It is important to note that the calculation to recapture the glide slope is not the same class of problem as determining whether the current state lies *on* the glide slope, or *what control* will take the aircraft to the glide slope, which is used to close the loop in a landing controller. Our problem is the real-time calculation to determine whether the landing controller may be safely invoked. The $\mathcal{P}_{\text{retry}}$ solution, then, includes both the control law or laws which will guide p_{i+1} back onto the glide slope, and then resume its $\mathcal{P}_{\text{one}}$ controller, as well as the *calculation* that the invocation of the recapture control law does not violate any stability or safety constraints. Figure 2c visually describes a case where the $\mathcal{P}_{\text{retry}}$ solution is followed.

If the calculation deems that stability or safety constraints will be violated, then p_{i+1} will continue on its plan to rejoin the landing queue, as shown in Fig. 2 d.

2.2.4 *Applicability of a* $\mathcal{P}_{\text{retry}}$ *solution*

The motivation we have to solve this problem extends from the need for improvement in the operability of UAVs during landing operations—especially in a military context, where high-risk maneuvers are more likely to be employed. During maneuvers to land on a carrier (or in any hostile environment) one major input to a human-driven controller (i.e., a pilot) when solving this recapture problem is the experience of that pilot in similar situations—whether encountered in a simulator or in the field. The pilot of an aircraft will decline to land in certain situations if the pilot's situation awareness suggests that it could violate the human or aircraft performance constraints. Note that the use of the pilot's instinct is in a military, rather than commercial, context where caution almost always trumps aircraft or human performance constraints.

One difficulty with substituting a computer-driven controller for a human controller when solving $\mathcal{P}_{\text{next}}$ is that the computer controller does not have the vast experience or dependable natural instinct of a trained human pilot. Attempting to substitute the human with the computer could be disastrous if the computer were not capable of determining whether a planned sequence of maneuvers would exceed the aircraft's safety or stability constraints—possibly resulting in loss of the aircraft and damage on the ground. Any controller that is used during flight, then, must be aware of the safety and stability constraints of the aircraft, and should never violate those constraints. Since the state of the aircraft is changing at all times, and the calculation period of a solution may be outdated by the time a solution is calculated, time-criticality is an issue. The decision cannot be made over a period of seconds; rather, the period should be milliseconds, since the window of opportunity to land safely may close during the time in which the decision is made. This is discussed further in Sect. 5, where we examine the time-criticality constraints of $\mathcal{P}_{\text{retry}}$.

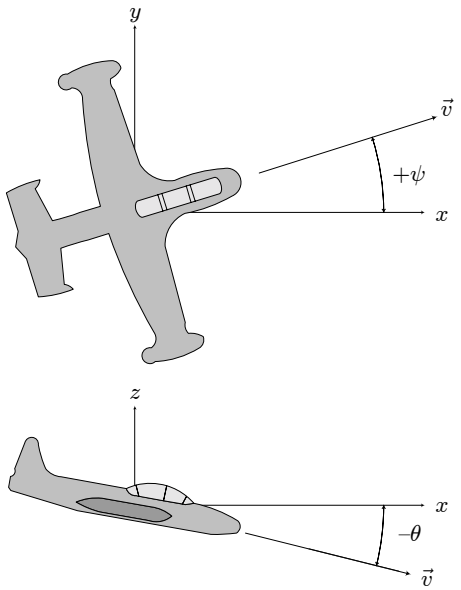


Fig. 3 Definitions of angular measurements in terms of body motion. Note that x represents the longitudinal axis of the runway, y represents the axis for the width of the runway, and z is altitude. These directions are useful for the direction of motion for designing the control laws, but note that during reachability calculations many of these values will be negative due to the calculations backward in time

We now turn our attention to the control laws and state equations used in the $\mathcal{P}_{\text{retry}}$ solution to recapture the glide slope.

3 Control laws

In this section, we construct a pair of control laws that force an aircraft to reach the glide slope. We first consider the case when the aircraft is moving in two-dimensional space, and then extend the control laws to the case when the aircraft is moving in three-dimensional space.

3.1 Two-dimensional control laws

Consider the simplified kinematic model of an airplane moving in two-dimensional space as dictated by the ordinary

differential equation (ODE):

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_3 \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ u \end{pmatrix} = f(x_1, x_3, \theta, u) \tag{1}$$

where the velocity of the aircraft, v, is assumed to be constant. We also assume that the control u is bounded, that is $u \in [-U_{\max}, U_{\max}]$ where $U_{\max}$ is the maximum value that the control input can take. We will also suppose that the angle θ can only take values in a set $[-\theta_{\max}, \theta_{\max}] \subset [-\pi, \pi)$. These values depend on the specific aircraft being considered, e.g., for a passenger airliner, the maximum angle would be much smaller than for a fighter aircraft.

The *glide slope* is the subset of the state space such that, if the trajectories of (1) reach this subset before reaching the landing zone, a good landing is possible. We define the glide slope as follows:

$$\mathcal{G} = \{(x_1, x_3, \theta) \in \mathbb{R}^2 \times [-\pi, \pi) : \; x_3 = \tan(\theta_{\mathcal{G}})x_1 \\ \theta = \theta_{\mathcal{G}}\} \tag{2}$$

where $\theta_{\mathcal{G}}$ is the *angle of approach*; typically, this is taken to be $\theta_{\mathcal{G}} = -3^\circ$.

The goal is to define a feedback control law

$$u = K(x_1, x_3, \theta) \tag{3}$$

such that, if

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_3 \\ \dot{\theta} \end{pmatrix} = f(x_1, x_3, \theta, K(x_1, x_3, \theta)) \tag{4}$$

is the closed-loop system, then

$$\lim_{t\to\infty} (x_1(t), x_3(t), \theta(t)) \in \mathcal{G}\,. \tag{5}$$

We define the feedback controller K to achieve this specification. Reachability analysis will then be used on the closed-loop system to see which solutions reach some neighborhood of the glide slope before contacting the ground. First, we construct a pair of control laws. Each of these will be used later on the more general three-dimensional case. First, suppose that we want our third state variable, θ, to track some desired angle which is given as a function of the state variables: $\theta_d(x_1, x_3, \theta)$. Then we can force θ to converge to this desired-angle function in finite time by defining the bang–bang control law:

$$K(x_1, x_3, \theta) = -U_{\max}\text{sign}(\theta - \theta_d(x_1, x_3, \theta))\,. \tag{6}$$

Therefore, we can construct a family of control laws by defining desired-angle functions θ_d.

Although bang–bang controllers are optimal, they depend on the ability to change the control input in a discontinuous and nonsmooth fashion, which is an unrealistic assumption in an aircraft with momentum and other consideration which drive the feasibility of its control inputs. Ideally, we would want to command values which would not saturate the inner-loop controller, and thus provide the behavior we anticipate in offline verification.

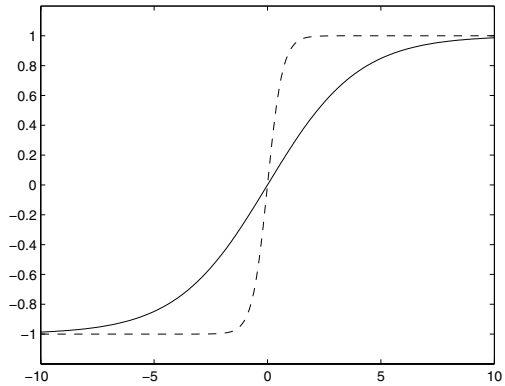


Fig. 4 Smoothed version of the sign function, used to yield a more desirable control input

Since bang–bang control laws are not desirable, our actual control laws will utilize approximations of the sign function. That is, we will use Γ_ϵ to be any function that uniformly converges to the sign as $\epsilon \to 0$. A specific example of this is obtained by using the well-known sigmoid function given by:

$$\sigma_\epsilon(x) = \frac{1}{1 + \mathrm{e}^{-\frac{1}{\epsilon}x}} \tag{7}$$

The approximation of the sign function is then given by

$$\Gamma_\epsilon(x) = 2\sigma_\epsilon(x) - 1 = \frac{2}{1 + \mathrm{e}^{-\frac{1}{\epsilon}x}} - 1 \tag{8}$$

We will typically take a value of ϵ that is small, but not too small, e.g., $\epsilon = 1$. Variations on these values, which show the convergence to the actual sign function, are shown in Fig. 4. In this case, we obtain smooth feedback control laws (assuming θ_d is smooth) by considering the control law:

$$K(x_1, x_3, \theta) = -U_{\max}\Gamma_\epsilon(\theta - \theta_d(x_1, x_2, \theta))\,. \tag{9}$$

By considering a set of desired-angle functions, we obtain a set of feedback control laws.

Thus far, we have not utilized our restrictions on the angle. To do this, we threshold the desired-angle function. More specifically, we consider the saturation function $\text{Sat}^V(x)$ defined by

$$\text{Sat}^V(x) = \begin{cases} x & \text{if } |x| < V, \\ \text{sign}(x)V & \text{otherwise.} \end{cases} \tag{10}$$

We can write this function in terms of sign functions and, utilizing our approximation of the sign function, we obtain a smoothed version of this function, $\text{Sat}^V_\epsilon(x)$, given by

$$\text{Sat}^V_\epsilon(x) = V\sigma_\epsilon(x - V) - V\sigma_\epsilon(-(x + V)) \\ +x\left((1 - \sigma_\epsilon(x - V))(1 - \sigma_\epsilon(-(x + V)))\right)^{(1+\frac{1}{V})^{\sqrt{\epsilon}}} \tag{11}$$

This approximation of the saturation function is somewhat subtle. An especially remarkable feature of this function is the presence of the expression $(1 + \frac{1}{V})^{\sqrt{\epsilon}}$. The purpose of this

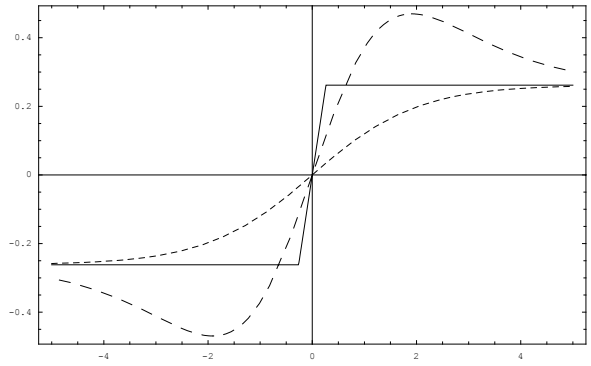


Fig. 5 Candidate smoothed versions of the $[\mathrm{Sat}_\epsilon^V(x)]$ function, used to saturate at the maximum input value. Note the relative overshoot present without the exponential term in (11), compared to the canonical $\mathrm{Sat}^V(x)$ function

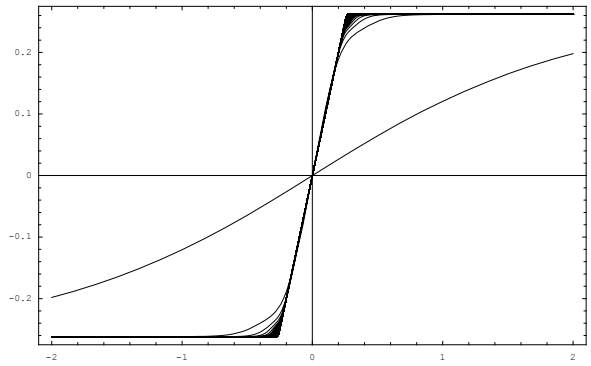


Fig. 6 Many different values of ϵ show that, as ϵ approaches zero, the approximate saturation function $\mathrm{Sat}_\epsilon^V(x)$ converges nicely to the canonical $\mathrm{Sat}^V(x)$ function

expression is two-fold: it prevents the approximation of the saturation function from overshooting (Fig. 5) the saturation value and it forces the approximation to converge in a well-behaved way to that saturation function (Fig. 6) as $\epsilon \to 0$. If this term were absent—as it would be in more obvious approximations of the saturation function—there would be a substantial amount of overshoot in the approximation of this function. This would greatly reduce the believability of control laws that utilize approximate saturation functions.

Utilizing the smooth saturation function, we obtain a smooth feedback control law

$$K(x_1, x_3, \theta) = -U_{\max}\Gamma_\epsilon(\theta - \mathrm{Sat}_\epsilon^{\theta_{\max}}(\theta_d(x_1, x_3, \theta))) \quad (12)$$

which smoothly prevents the desired-angle function from exceeding the maximum angle.

Control law I–2D: First, we define the desired-angle function by

$$\theta_d(x_1, x_3, \theta) = \begin{cases} -\theta_{\max} & \text{if } x_3 > \tan(\theta_\mathcal{G})x_1 \\ \theta_{\max} & \text{if } x_3 < \tan(\theta_\mathcal{G})x_1 \\ \theta_\mathcal{G} & \text{if } x_3 = \tan(\theta_\mathcal{G})x_1 \end{cases} \quad (13)$$

which can be written as a discontinuous function in terms of the sign as follows

$$\theta_d(x_1, x_3, \theta) = -\theta_{\max}\mathrm{sign}(x_3 - \tan(\theta_\mathcal{G})x_1) - \theta_\mathcal{G}(\mathrm{sign}(x_3 - \tan(\theta_\mathcal{G})x_1))^2 + \theta_\mathcal{G} \quad (14)$$

Basically, this desired-angle function says that, if the aircraft is above or below the glide slope, it should turn towards the glide slope as much as possible by applying $-\theta_{\max}$ or $\theta_{\max}$, and if it is on the glide slope it should stay on the glide slope.

It is desirable to have a smooth version of this function, so again taking our approximation for the sign function (8) we obtain a smooth desired-angle function:

$$\theta_d^\epsilon(x_1, x_3, \theta) = -\theta_{\max}\Gamma_\epsilon(x_3 - \tan(\theta_\mathcal{G})x_1) - \theta_\mathcal{G}(\Gamma_\epsilon(x_3 - \tan(\theta_\mathcal{G})x_1))^2 + \theta_\mathcal{G} \quad (15)$$

Using the methods given in (9), we obtain a smooth (in fact, analytic) feedback control law[3] approximating the bang–bang controller given by:

$$K_\mathrm{I}(x_1, x_3, \theta) = -U_{\max}\Gamma_\epsilon(\theta - \theta_d^\epsilon(x_1, x_3, \theta)) \quad (16)$$

where θ_d^ϵ is given in (15). We will later generalize this control law to the a three-dimensional configuration space.

Control law II–2D: We can define a control law by requiring that our desired angle change proportionally to the distance of the aircraft from the glide slope; this is a slight variation on the standard proportional controller as we first obtain the glide slope in space and then we focus on obtaining the glide-slope angle. In other words, our desired-angle function is given by

$$\theta_d(x_1, x_3, \theta) = -(x_3 - \tan(\theta_\mathcal{G})x_1) + \theta_\mathcal{G} \quad (17)$$

From this, and utilizing (12), we obtain the desired feedback control law:

$$K_\mathrm{II}(x_1, x_3, \theta) = -U_{\max}\Gamma_\epsilon(\theta - \mathrm{Sat}_\epsilon^{\theta_{\max}}(\theta_d(x_1, x_3, \theta))) \quad (18)$$

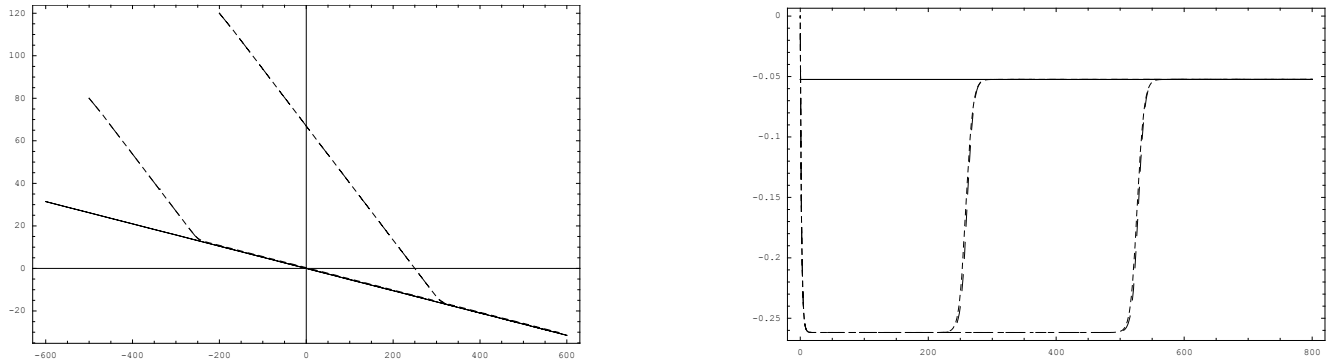
where θ_d is given in (17) and the smooth saturation function is given in (11). As with control law I–2D we will generalize this control law to the three-dimensional case.

Regardless of the controller chosen, in many initial states the glide slope will not be reachable, as illustrated in Fig. 7. In such a case no controller would be capable of reaching the landing target without breaking one or more constraints. The exact determination of these reachability states is the subject of Sect. 4.

3.2 Three-dimensional control laws

We consider a simplified kinematic model of an airplane moving is three-dimensional space. Specifically, this is given by the ODE:

[3] In this case we do not need to use the smooth approximation of the saturation function since, by design, our desired-angle function will not exceed the maximum angle.



(a) (b)

Fig. 7 Two attempts to capture the glide slope from two initial positions: in the trajectory starting further towards the left the glide slope is reachable, in the trajectory towards the right it is not reachable and the aircraft misses the landing target at the origin. **a** Trajectory of recapture attempts: x and z shown. **b** Trajectory of recapture attempts: θ shown

$$\dot{x} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} v\cos(\psi)\cos(\theta) \\ v\sin(\psi)\cos(\theta) \\ v\sin(\theta) \\ u_1 \\ u_2 \end{pmatrix} = f(x, u_1, u_2) \tag{19}$$

This equation dictates the evolution of the aircraft relative to the origin, which is the assumed location of the landing strip. As in the two-dimensional case we will assume that the control inputs are bounded, i.e., $u_1 \in [-U^1_{\max}, U^1_{\max}]$ and $u_2 \in [-U^2_{\max}, U^2_{\max}]$. Again, these values depend on the aircraft being considered. We will also assume that the angles θ and ψ are constrained, that is $\theta \in [-\theta_{\max}, \theta_{\max}] \subset [-\pi/2, \pi/2]$ and $\psi \in [-\psi_{\max}, \psi_{\max}] \subset [-\pi, \pi)$.

The *glide slope* is a subset of the state space such that, if the trajectories of (19) reach this subset before reaching the landing zone, a good landing is possible. We define the glide slope as in the two-dimensional case:

$$\begin{aligned} \mathcal{G} = \{(x_1, x_2, x_3, \theta, \psi) \in \mathbb{R}^3 \times [-\pi, \pi) \times [-\pi/2, \pi/2] : \\ x_2 = \tan(\psi_{\mathcal{G}})x_1 \\ x_3 = \sin(\theta_{\mathcal{G}})\sqrt{x_1^2 + x_2^2 + x_3^2} \\ \theta = \theta_{\mathcal{G}} \\ \varphi = \psi_{\mathcal{G}}\} \end{aligned} \tag{20}$$

where $\theta_{\mathcal{G}}$ is the *glide slope heading*; typically, this is taken to be $\theta_{\mathcal{G}} = -3^\circ$. Here, $\psi_{\mathcal{G}} = 0$ is the *glide slope heading*.

The goal is, as in the two-dimensional case, to define feedback control laws $u_1 = P(x)$, and $u_2 = Q(x)$ such that, if

$$\dot{x} = f(x, P(x), Q(x)) \tag{21}$$

is the closed-loop system, then

$$\lim_{t\to\infty} x(t) \in \mathcal{G}. \tag{22}$$

We extend the two-dimensional controllers to the three-dimensional case to achieve this goal; as for the two-dimensional controllers, the general method for designing controllers that reach the glide slope is to define desired-angle functions $\theta_d(x)$ and $\psi_d(x)$; the goal of the controllers will be to track these desired-angle functions.

Control law I–3D: The first three-dimensional controller will be a direct generalization of the first two-dimensional control law, i.e., it is obtained by first considering a bang–bang control law and then smoothing this control law. Progressing directly to the desired-angle functions, which could be equivalently be stated in terms of conditional statements, we have:

$$\begin{aligned} \theta_d(x) = &-\theta_{\max}\text{sign}(x_3 - \sin(\theta_{\mathcal{G}})\sqrt{x_1^2 + x_2^2 + x_3^2}) \\ &-\theta_{\mathcal{G}}\left(\text{sign}(x_3 - \sin(\theta_{\mathcal{G}})\sqrt{x_1^2 + x_2^2 + x_3^2})\right)^2 \\ &+\theta_{\mathcal{G}} \end{aligned} \tag{23}$$

$$\begin{aligned} \psi_d(x) = &-\psi_{\max}\text{sign}(x_2 - \tan(\psi_{\mathcal{G}})x_1) \\ &-\psi_{\mathcal{G}}(\text{sign}(x_2 - \tan(\psi_{\mathcal{G}})x_1)^2 + \psi_{\mathcal{G}} \end{aligned} \tag{24}$$

This desired-angle function says that, if the aircraft is above or below the glide slope, it should turn towards the glide slope as much as possible by applying $-\theta_{\max}$ or $\theta_{\max}$ and, if it is on the glide slope, it should stay on the glide slope. Similarly, it says that, if the aircraft is to the left or right of the glide slope, then it should apply $-\psi_{\max}$ or $\psi_{\max}$ and, if it is on the glide slope, it should stay on the glide slope.

Smooth versions of these desired-angle functions are obtained by using the approximation of the sign function given in (8). This yields the desired-angle functions

$$\theta_d^\epsilon(x) = -\theta_{\max}\Gamma_\epsilon(x_3 - \sin(\theta_\mathcal{G})\sqrt{x_1^2 + x_2^2 + x_3^2}) - \theta_\mathcal{G}\left(\Gamma_\epsilon(x_3 - \sin(\theta_\mathcal{G})\sqrt{x_1^2 + x_2^2 + x_3^2})\right)^2 + \theta_\mathcal{G} \quad (25)$$

$$\psi_d^\epsilon(x) = -\psi_{\max}\Gamma_\epsilon(x_2 - \tan(\psi_\mathcal{G})x_1) - \psi_\mathcal{G}(\Gamma_\epsilon(x_2 - \tan(\psi_\mathcal{G})x_1)^2 + \psi_\mathcal{G} \quad (26)$$

which are direct generalizations of the desired-angle function given in (15). From (9) it follows that our feedback controllers are given by

$$P_\mathrm{I}(x) = -U^1_{\max}\Gamma_\epsilon(\theta - \theta_d^\epsilon(x)) \quad (27)$$

$$Q_\mathrm{I}(x) = -U^2_{\max}\Gamma_\epsilon(\psi - \psi_d^\epsilon(x)) \quad (28)$$

where θ_d^ϵ and ψ_d^ϵ are the desired-angle functions given in (25) and (26). These control laws are a direct generalization of that given in (16).

Control law II–3D: We can generalize the proportional controller given in (18) to a three-dimensional controller. In other words, our desired-angle functions are given by

$$\theta_d(x) = -\left(x_3 - \sin(\theta_\mathcal{G})\sqrt{x_1^2 + x_2^2 + x_3^2}\right) + \theta_\mathcal{G} \quad (29)$$

$$\psi_d(x) = -(x_2 - \tan(\psi_\mathcal{G})x_1) + \psi_\mathcal{G} \quad (30)$$

From this, and utilizing (12), we obtain the desired feedback control law:

$$P_\mathrm{II}(x) = -U^1_{\max}\Gamma_\epsilon(\theta - \mathrm{Sat}_\epsilon^{\theta_{\max}}(\theta_d(x))) \quad (31)$$

$$Q_\mathrm{II}(x) = -U^2_{\max}\Gamma_\epsilon(\psi - \mathrm{Sat}_\epsilon^{\psi_{\max}}(\psi_d(x))) \quad (32)$$

where θ_d and ψ_d are given as in (29) and (30).

4 Reachability calculations

The finite horizon backwards *reach set* $\mathcal{G}(t)$ is the set of states from which trajectories arise that lead to some target set $\mathcal{G}_0$ after exactly a specified time t. If the system's dynamics include inputs, those inputs can be chosen to drive the trajectories towards or away from the target set; the interpretation of these inputs as either optimal controls or worst-case disturbances depends on the circumstances.

4.1 Encoding the problem as a reachable set

Although we have already defined the glide slope as a mathematical ideal, we also have to define it as a set to which we want to test intersection. In the glide-slope-recapture scenario studied here, we lightly abuse our notation and denote the target set, $\mathcal{G}_0$, as a small set of states at the end of the runway where the aircraft can safely depart from the glide slope and accomplish a soft touchdown with a flare maneuver.

$$\mathcal{G}_0 = \begin{cases} \theta_\mathcal{G} \in [2.85^\circ, 3.15^\circ], \\ \psi_\mathcal{G} \in [-0.2^\circ, +0.2^\circ], \\ x_2 \in [-100, +100]\ \text{ft}, \\ x_3 \in [-15, +15]\ \text{ft}, \\ x_1 = 0. \end{cases} \quad (33)$$

It is appropriate to name this state $\mathcal{G}_0$ since it denotes the desired glide-slope angle, $\theta_\mathcal{G}$, and it represents the desired value at some time, zero.

The control inputs ($\dot\theta$ and $\dot\psi$) are chosen to drive trajectories toward this set of safe states. The union of the backwards reach sets over all time will therefore represent the controllable safe envelope, or the set of states from which the aircraft can safely set up for a soft landing.

As with the generation of Lyapunov functions for stability analysis, the reach set of a nonlinear system is often impossible to determine analytically. Depending on the type of dynamics and the presence of inputs, there are many approaches to approximating these sets; in this paper we adopt an algorithm based on the Hamilton–Jacobi (HJ) partial differential equation (PDE) [12, 17]. We briefly recap some features of that work, and modify some other features here. Most notably, that work included competing input signals, while here all inputs are seeking to drive the system to the target set. Also, the reach set described here is for a fixed point in time t, while the reach set in previous work was a union over all times $s \in [0, t]$ (when necessary, we will refer to such a union as a *reach tube*). These modifications simplify the presentation, and also allow us to apply some tricks to simplify the computational effort.

The target and reach sets are represented by an implicit surface function $\psi_\mathrm{surf}(x, t)$

$$\mathcal{G}_0 = \{x \in \mathbb{X} \mid \psi_\mathrm{surf}(x, 0) \le 0\},$$
$$\mathcal{G}(t) = \{x \in \mathbb{X} \mid \psi_\mathrm{surf}(x, t) \le 0\},$$

where $\mathbb{X}$ is some well-behaved configuration space; in this case $\mathbb{X} = [0, 360^\circ)^2 \times \mathbb{R}^3$ and $x = (\theta\ \psi\ x_1\ x_2\ x_3)^\mathrm{T}$. Construction of the implicit surface function $\psi_\mathrm{surf}(x, 0)$ for $\mathcal{G}_0$ is generally straightforward: there are simple functions for common shapes such as circles, spheres, cylinders, halfspaces and prisms, and the set operations of union, intersection and complement become the pointwise functional operations minimum, maximum and negation. For the target set (33), $\mathcal{G}_0$ is the intersection of eight halfspaces (each of the four intervals has two endpoints), and so $\psi_\mathrm{surf}(x, 0)$ turns out to be the pointwise maximum of eight simple linear functions. The separate treatment of the x_1 constraint in $\mathcal{G}_0$ is described below.

Calculation of the backwards reach set then reduces to solving an initial-value HJ PDE. Let the system dynamics be given by $\dot x = f(x, u)$, where f is bounded and Lipschitz-continuous in x. Choose the input signal u to drive trajectories toward $\mathcal{G}_0$ and assume that it is measurable and bounded at

each point of time $u(t) \in \mathcal{U}$, where $\mathcal{U}$ is compact. Then the implicit surface function $\psi_{\text{surf}}(x, t)$ for the backwards reach set $\mathcal{G}(t)$ is the solution to the initial-value HJ PDE

$$\frac{\partial \psi_{\text{surf}}}{\partial t} - H\left(x, \frac{\partial \psi_{\text{surf}}}{\partial x}\right) = 0, \tag{34}$$

where

$$H(x, p) = \min_{u \in \mathcal{U}} p^{\mathrm{T}} f(x, u) \tag{35}$$

and the initial conditions $\psi_{\text{surf}}(x, 0)$ are defined as above. For more details, see [12]; however, note that the HJ PDE given there is for a reach-tube calculation with competing inputs and is a terminal-value PDE (time proceeds backwards from $t = 0$).

Finding an analytic solution of this nonlinear PDE is not usually possible; in fact, it often does not have a classical differentiable solution at all. Fortunately, a well-defined weak solution exists, and numerical methods have been designed to approximate it. The Toolbox of Level Set Methods is a collection of such algorithms that runs in MATLAB and is available free of charge on the web [8]. For more information on level sets, please see the excellent [13].

4.2 Computational complexity

The biggest problem with such techniques is that they fall prey to Bellman's curse of dimensionality. In the case of the toolbox, the state space is divided into a Cartesian grid, so the computation and memory costs grow exponentially with dimension. In practice, systems of dimension 1–3 can be studied interactively on a desktop, systems of dimension 4–5 require long periods on a well-equipped machine, and systems of dimension 6+ are impractical.

Consequently, when studying the reach set of systems we work hard to reduce the dimension of their state space. In the case of the glide-slope-recapture problem, it turns out that we can capture the important features of the full five-dimensional system by solving two PDEs in only two spatial dimensions. The two tricks for dimensional reduction outlined below may prove useful in other reach-set calculations.

4.2.1 Distance and time

The first trick makes use of the fact that we are not actually interested in the reach set at any particular time, since we do not usually know the exact time until touchdown. A more useful measure is the set of safe states (the reach set) at a particular distance from the end of the runway (a state which is assumed known). While this information is available from the reach tube, it seems like a waste to calculate with a temporal variable which will then be discarded. Fortunately, for this particular system the dynamics of x_1 are monotonic in time, because $\theta, \psi \in (-90^\circ, +90^\circ)$ and thus

$$\dot{x}_1 = f_{x_1}(x, u) = f_{x_1}(x) = v \cos\psi \cos\theta > 0.$$

Furthermore, the dynamics of all variables are independent of x_1. Therefore, we can substitute x_1 for our time variable in the PDE and reduce the spatial dimension of the problem by one [7]. Let $\tilde{x} = (\theta\ \psi\ x_2\ x_3)^{\mathrm{T}}$ be the reduced state dimension and $\dot{\tilde{x}} = \tilde{f}(\tilde{x}, u)$ be the reduced dynamics (simply remove the x_1 component). Solve (34) with

$$H(\tilde{x}, p) = \min_{u \in \mathcal{U}} \frac{p^{\mathrm{T}} \tilde{f}(\tilde{x}, u)}{f_{x_1}(\tilde{x})} \tag{36}$$

instead of (35). The result $\psi_{\text{surf}}(\tilde{x}, t)$ is an implicit surface representation of the reach set at $x_1 = t$. Simple implementation of this change of variables is also the reason why the target set's dependence on x_1 in (33) is a point rather than a range; in this case, the target set is still represented by only the initial conditions $\psi_{\text{surf}}(\tilde{x}, 0)$ of the PDE. There are ways to implement a range, but they are more complicated.

While a four-dimensional system's reach set can be computed, it still takes several days on a fast machine.

4.2.2 Dynamical decoupling

The second trick for dimensional reduction is to split the system into two separate projections. Noting that the θ–x_3 dynamics are independent of the ψ–x_2 dynamics, we simply compute two separate reach sets. One uses the θ–x_3 target set and dynamics, while the other uses the ψ–x_2 target set and dynamics. A safe landing is possible from a state only if it lies inside both reach sets.

Astute readers may object that these projections are coupled through f_{x_1} as it appears in (36), since the x_1 dynamics depend on both θ and ψ. However, in (36) f_{x_1} is a common divisor for all the other dynamics. In the θ–x_3 projection, for example, plugging any value of ψ into f_{x_1} would have the same effect on the evolution of the reach set in both the θ and x_3 dimensions. Therefore, we simply ignore the missing angular dimension when calculating f_{x_1} in these projections, since it is effectively a common scaling factor. Calculations with the full four-dimensional dynamics support this simplification.

We are investigating ways of working in lower-dimensional subspaces even if the dynamics of the projections are inseparably coupled. We have examined this problem in previous work [11], but those techniques are not directly applicable—they find overapproximations of the reach set, but for envelope-protection problems like this one safety demands underapproximations.

Code to approximate the reach set of the glide-slope-recapture problem using the toolbox is available as a separate download from [8]. Some additional details of that code are worth mentioning here.

The treatment of inputs in (35) and (36) both generate reach sets based on optimal, measurable input signals. In practice, these are bang–bang controllers and are not usually suitable for implementation on real vehicles. Given any appropriate feedback control law $u = K(x)$, such as those designed in Sect. 3, we can compute the reach set of the system under that feedback control by using the dynamics $\dot{x} = f(x, K(x))$ in (35). From the viewpoint of the minimization in that equation, these dynamics are now input-free

(u does not appear). Assuming that the feedback control law satisfies the same constraints as the optimal controller, the resulting reach set will be smaller; however, it may be more practical to implement. The released code allows for computation of both optimal and feedback controlled reach sets.

4.2.3 Grid scaling

Like most methods to approximate the HJ PDE, the toolbox works on a regular Cartesian grid of the state space. The grid cell size gives a rough idea of the resolution of the reach set approximation; features of the reach set that are smaller than a few grid cell widths will not be resolved. Unfortunately, the target set is very small compared with the size of the eventual reach set; for example, in the x_2 dimension the target set at $x_1 = 0$ is roughly a factor of 1000 smaller than the size of the reach set at $x_1 = 10NM$ [AQ1]. Resolution of both sets would therefore demand a grid with several thousand nodes in each dimension, and such grids are very expensive to work with.

To resolve this range of scales despite the restriction of a uniform grid, we perform a nonlinear change of variables for some dimensions. The uniform grid is built in $\bar{x}$ space, where $kx = \sinh \bar{x}$ and k is a scale factor that controls the degree of nonlinearity. The hyperbolic sine function is chosen because it is monotonic and has a bounded derivative over any bounded domain. An example built grid is shown in Fig. 8. By increasing k, we can increase the resolution near the origin of the state space, at the cost of reducing resolution at the edges of the grid. For the problem described in this paper, such a reduction is acceptable since it occurs in states far from touchdown. The transformed dynamics are given by

$$\dot{\bar{x}} = \frac{\mathrm{d}\bar{x}}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{k\dot{x}}{\sqrt{1+k^2x^2}}.$$

Such nonlinear scaling allows reasonable approximation of the reach set with only a hundred nodes per dimension, instead of several thousand. If n is the number of nodes per dimension and d is the dimension, the cost of approximating the solution of (34) to some fixed t using the toolbox is $\mathcal{O}(n^{d+1})$, so the savings due to scaling is enormous. The released code includes subfunctions to perform the nonlinear transformation as well as linear transformations that are used to handle k and to improve the relative scaling of the different dimensions even if nonlinear scaling is not used.

4.3 Maintaining constraints

In addition to constraints on their temporal derivatives, the angular states ψ and θ are also constrained in their values through (Sect. 3). The reach set must remain a subset of these constraints at all times if it is to denote the set of states from which it is safe to land. In the implementation these constraints are represented by implicit surface functions, and are then imposed on the reach-set evolution through a process called masking. Further discussion of masking and other implementation details for calculating reach sets using the toolbox can be found in the toolbox documentation.

5 Online decision control synthesis

In Sect. 4 we described how it was that we were able to produce the backward-reachable sets for both the dynamical system (i.e., what was dynamically possible to happen) and the controller (i.e., what would happen under certain initial conditions). In this section we discuss exactly how we can take the raw data produced by these reachability calculations and form from it a *decision controller*, which yields a decision for the control of the aircraft under certain conditions.

5.1 Requirements

To operate the decision controller on the aircraft, several requirements must be met, which are somewhat, but not entirely, independent of the requirements of the code. These revolve mostly around the robustness of the code.

Functional requirements:

- the answer must not be a false positive, under any circumstances;
- false negatives should be minimized (or at least, be acceptably low).

Performance requirements:

- the answer must come *in time*;
- the possibility of preemption should be minimized.

Reliability requirements:

- an answer should always be returned.

5.2 Software engineering

To enforce the requirements we examined several options for the final software solution. We crafted the software in such a way that it addressed the concerns discussed in the Introduction: specifically, stability, safety, optimality, etc.

5.2.1 False positives and false negatives

The two functional constraints (no false positives, and acceptably low false negatives) emerge from the definition of the safe set, as discussed in Sect. 4. The safe set defines a surface object, inside of which everything is controllably safe, and outside of which everything is uncontrollably unsafe. The boundary of the set is somewhat tricky, since being on the boundary implies safety assuming that you utilize exactly the control law.

However, we are doing a numerical computation of the level set, which means that there are ϵ differences between

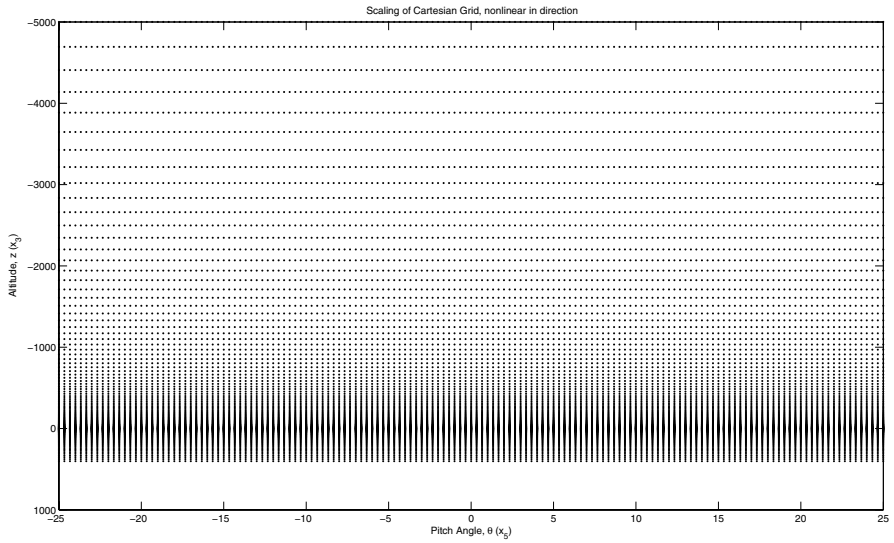


Fig. 8 Nonlinear grid scaling in the altitude dimension. This is useful since such a small target set requires additional sample points around any "interesting" behavior, but would prove too dimensionally difficult to replicate that frequency over large values of the variable

what the actual and computed set are. To account for this, we define an ϵ-boundary around the zero values in which we assume that we are *outside* of the set. The size of this ϵ-boundary is defined by the acceptably low nature of the computation, and is aided in definition by the method of computation of the reach set (e.g., 'low' versus 'veryHigh') as well as the definition of the Cartesian grid (a higher N will result in a more accurate reach set).

The ability to place a guarantee on the number of false negatives is discussed in Sect. 8.

5.2.2 Compositionality

Using the hierarchical controller inherent in the aircraft design (discussed in Sect. 6) we must consider the effects of switching control strategies. Specifically, we must consider the effects of switching to decrease rather than increase in altitude.

Solving these kinds of switching problems for systems in general falls into the domain of *hybrid systems*, which is an emerging and rich area of research. For our particular application, we must concern ourselves with the physics of commanding the aircraft to change its momentum to descend rather than ascend.

Since we are not controlling from the inner loop (i.e., the stability controller), any commands we give to the autopilot will not make the aircraft unsafe in flight (i.e., we will not cause a stall, or a structural malfunction). What safety concerns remain, then, are best described in the following thought: how could we crash into the ground? There are only a few possibilities for this (assuming that we will maintain lift and no structural damage):

1. overshooting the glide slope;
2. not capturing the glide slope;
3. capturing the glide slope in space, but not velocity or angle.

To overcome #1 we designed the control law discussed in Sect. 3 to follow trajectories with an acceptable overshoot. Then, upon performing the reachability analysis, we used the control law, rather than the raw performance possibilities of the aircraft, to create the set. This means that the reach set is the full set of possible *controller* initial conditions that will result in control directly to the target set.

To overcome #2, which is not quite capturing the glide slope in space (and thus running past the runway when attempting to land), we place realistic bounds on the control input values. This again stems from the careful definition of a controller with smooth inputs, which are thus more likely to reflect actual inputs at flight time. By obeying these input constraints to the controller, we approximate the affect of momentum during flight, which produces a lag in response to inputs based on current velocity and angular acceleration.

To overcome #3, we simply recall that the angles of approach and pitch are both calculated in the reachable sets, and are part of the target set. This means that if we reach the target set, we will be in the correct attitude.

5.2.3 Time-criticality

The decision made by the decision controller must be made in accordance with the flow of time. For our particular application, there were discrete time steps between refresh of the input values for the autopilot. Therefore, upon request of a decision for $\mathcal{P}_{\text{retry}}$, a decision prior to advancement to the next time step is the upper bound on the optimal time.

However, there is no guarantee that such a decision can be made prior to that time if dependence upon hardware or lengthy calculations are required. This is one reason why we chose backward-reachable sets, because, if the set is known from an offline calculation, then it is possible for us to examine that set while online and check our place in it. Thus, our computation time interval is on the order of milliseconds, if we can have the file loaded in memory. For a general calculation interval, this phenomenon is shown in Fig. 5. Note that, for our application, the calculation interval and the decision interval overlap and thus the decision for control is concurrent with the calculation of safety.

5.3 Code generation

Given that we need to have an online controller which is compiled from all of the offline-computed reach sets, it is somewhat intuitive that we should create this online executable through *generative techniques*. That is, we defined a transformation function which takes as a parameter a safe set, and produces as its output a C++ class which will perform evaluative measurements based on the safety, reliability, and time-criticality requirements outlined in the earlier portions of this section. By joining together these classes, we are able to produce a global view of the state of the aircraft, and from there pronounce safety under certain conditions.

5.3.1 Implementation details

We created our code generator using the C++ standard template library (STL) to provide maximum support for compilation across platforms. In fact, we were able to test the algorithms using Windows XP and Server 2003 as the operating system, and further performed tests and at flight time ran on RedHat Linux 9.

The code which was generated also took advantage of templatized classes, and the object-oriented features of the C++ language. Many of these implementation details are uninteresting as research, but we present the main *generated* classes below.

5.3.2 The `SafeSet` class

The `SafeSet` class is the base class which is called to check the safety of some current state, `x`, against some final location, `runway`, with compile-time values for the glide slope, angle of approach, and ϵ-error bounds (i.e., how close to trust the level set). The basic definition of this class is shown here:

```
class SafeSet
{
public:
SafeSet( );
virtual ~SafeSet();
SafetyResult CheckSafety
( Common::iX x,
Common::iX runway );
protected:
SafetyResult IsItSafe
( Common::iX x );
/* member data of safe sets,
defined as
their own classes at
generation time */
3_psi_400_1nm           *d_psi_400_1nm;
3_psi_400_3nm           *d_psi_400_3nm;
3_psi_400_10nm      *d_psi_400_10nm;
3_theta_400_1nm     *d_theta_400_1nm;
3_theta_400_10nm    *d_theta_400_10nm;
};
```

This particular code snippet is taken from the two orthogonal reach sets which share a common distance in x from the runway, and wish to look up different values to ensure that the state is safe in both sets. This is discussed in great detail in [15].

5.3.3 The reach-set classes

Each of the classes `3_psi_400_1nm` is a reach-set class—a self-contained class capable of looking up the value of a reach set from the state location at some point(s). They are name-mangled, as such:

`[`θ_G`]_[projection]_[velocity]_[maxXdistance],`

for each reach set, to make the generated code easier to understand. These classes contain member functions to do the lookup, which use contained classes of kind `Range` to interpolate and understand the safe sets as state information.

5.3.4 The `Range` class

The `Range` class is used to do a reverse lookup of the indices of a particular point in space and angle in the reach set. Since the reach set is stored as a multidimensional matrix, it is necessary to find from the (x, z, θ) values the value of the reach set which corresponds to those values. This is done through an interpolation scheme through the reach-set class.

```
class Range_3_psi_400_10nm_0 :
public Range
{
public:
Range_ngc3_3_psi_400_10nm_0() :
Range([minX],[maxX],[samples]) { }
~Range_ngc3_3_psi_400_10nm_0() { }
};
```

During this interpolation, the reach set is evaluated *forward* in time by one time step, in accordance with the time-criticality constraints. This parameter is set depending on the particular configuration of the outer- and inner-loop controllers, and is configurable at code generation time.

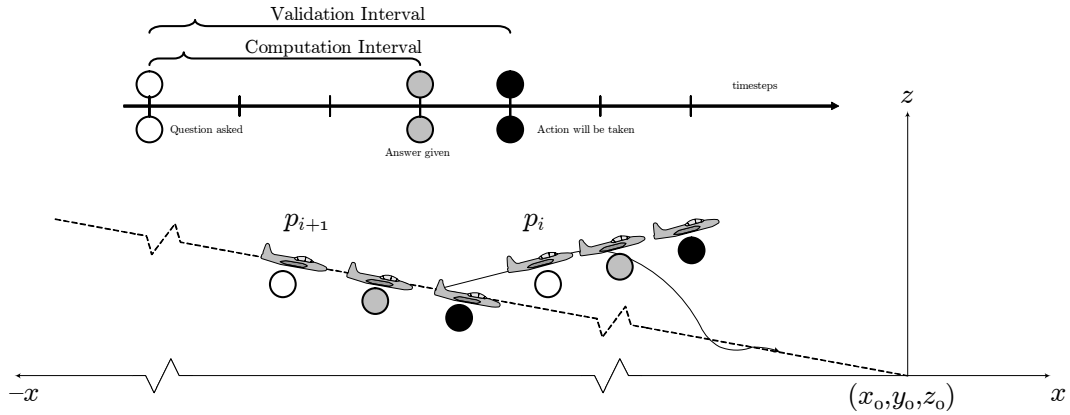


Fig. 9 The computation interval should influence the state data used for the calculation (derived from validation interval)

5.3.5 The `SafetyResult` class

Once a decision has been made, it is relayed back to the decision computer through the `SafetyResult` class. This class stores a large deal of information for the requesting program, including the exact value of the state at the time it was asked, whether the reach set at that point is safe, unsafe, or on the boundary, as well as internal debugging functions which guarantees that the safe set has been properly located (i.e., that out of "safe", "unsafe", and "boundary" exactly one is true).

The `SafetyResult` answer is compiled based on the topology of the deployed system. For example, in the code snippets above, the decision for which `Range` to check is based on the value of x, as divided into three portions for one portion of the safe set, and two portions for another. For the safe sets, which are calculated using the nonlinear scaling factor, this need for splitting reach sets into separate pieces is somewhat mitigated, though it may still be required for large reachability-analysis problems.

5.3.6 Variables and parameters

In general, we defined the following values as configurable at the code generator's compile time:

- `epsilon`: the proximity to the reach set which results in a decision of unsafe regardless of the sign of the value.
- `validationTime`: the additional time to advance the state of the aircraft, which allows the time-criticality constraint of safety to be addressed.

6 Flight demonstration

To provide elementary proof of concept of the Software Enabled Control project, a capstone demonstration was organized which allowed institutional participants to implement their control laws in software and also test their software on a real jet aircraft either alone, in cooperation with, or (in one case) against human pilots (see [14] depending on the application). In this section, we describe how we transformed our problem so that it could be demonstrated with the given equipment and safety concerns.

6.1 Capstone demonstration overview

The SEC program consisted of two capstone demonstration experiments, one of which was a fixed-wing UAV flight-test experiment that took place over three weeks at Edwards Air Force Base (AFB) using a pilot's aircraft modified for autonomous flight. The capstone demonstration development took place over a 13-month period in which many of concepts developed in the SEC program were prepared for this experiment. A second capstone demonstration experiment was also run to highlight SEC concepts on rotary-wing UAVs [10].

6.1.1 Equipment

A two-seater jet trainer T-33 owned by the Boeing Aircraft Company (originally manufactured by the Lockheed Martin Company) was modified by Boeing for use in the SEC capstone demonstration live flight testing in June 2004. This UAV surrogate aircraft included a third-party autopilot system which did not include airspeed control of the aircraft. The UAV had on board a safety pilot who could take control of the aircraft in the event of controller malfunction or poor decision making, as well as for controlling the airspeed of the aircraft based on indicator alerts and a displayed target airspeed given to the pilot to increase/decrease thrust. The trajectory of the UAV was controlled by a Linux OS laptop computer running Boeingć6s Open Control Platform (OCP) that was interfaced to the avionics and autopilot of the aircraft. The OCP system, including the safe-landing software, sent control commands to the avionics pallet, which transformed them into autopilot maneuver commands. The state of the UAV was available via this avionics interface.

The avionics and autopilot of the Boeing-owned T-33 had limited state data and output control available, which required

a simplified model to be used. For example, angle of attack (α angle) and side-slip (β angle) were not available and the autopilot was only capable of receiving rate of climb and rate of turn commands, not pitch, roll or yaw (θ, φ, ψ) commands.

6.1.2 Software component

To enable parallel development, reliable testing, rapid integration, and an operating-system-independent interface, a software in-the-loop simulation (SILS) platform was provided for the SEC capstone demonstration by Boeing. This OCP-based software platform used a generic (blackbox) aircraft simulator called DemoSim and Java-based UAV experiment controller GUI. The final versions of OCP and the experiment controller used were identical to those that would be used in the final flight-test experiments. The DemoSim model included the avionics and autopilot interfaces identical to the T-33 UAV surrogate such that the state information and autopilot commands would be identical from the perspective of the OCP-based applications.

The landing controller and decision algorithms were implemented as a single component in the Open Control Platform (OCP). The OCP is a CORBA-based real-time distributed control platform.

The reachable safe sets were computed offline using the level-sets toolbox and stored for online decision making by the controller. We chose C++ as our implementation language due to speed advantages, as well as the ability to have arbitrarily large arrays optimized for performance at compile time.

The controller and decision algorithms were tested in the SILS environment and provided to Boeing for integration and testing in their Hardware In-the-Loop Simulator (HILS). The HILS system included an interface to the same avionics system used on the UAV and a proprietary aircraft simulation system. Along with software, the experiment test plans were provided with which Boeing developers tested and verified all the software for the various experiments, including the landing controller.

6.2 Experiment

The capstone demonstration flight test took place at Edwards AFB in the Mojave Desert in June 2004. The T-33 demonstration aircraft was flown under pilot control to the flight test area, where the autonomous controller took over and began to land on a virtual runway several thousand feet in the air.[4] A successful landing was achieved by passing through a virtual landing spot on the runway (with some bounds for error) at a certain velocity with a certain attitude.

The demonstration of concept was then carried out in the following set of steps:

1. To simulate an accident on the runway, the ground control issues a vector-off command during descent onto the runway, at which time the aircraft departs the glide slope, changing its state vector in every dimension. The aircraft then proceeds from this time as if it were going to perform a go-around maneuver, as described in [9].
2. After some time the ground control issues a revised command to land if possible.
3. The decision algorithm examines the current state of the aircraft and determines whether it is safe to recapture the glide slope.
4. If determined to be safe, the controller will issue commands to recapture the glide slope; if unsafe, the controller stays on its current course of go-around maneuver.

6.3 Data gathered

During the flight test, the landing controller experiment was run on two separate flight tests for a total of four landing experiments. During these experiments the algorithm demonstrated its effectiveness and robustness in windy conditions and with user challenges such as VIPs commanding the wave-off and recapture signals from the experiment controller. The controller was exercised in these limited opportunities in conditions such that both "safe" and "unsafe" decisions were successfully made.

7 Analysis

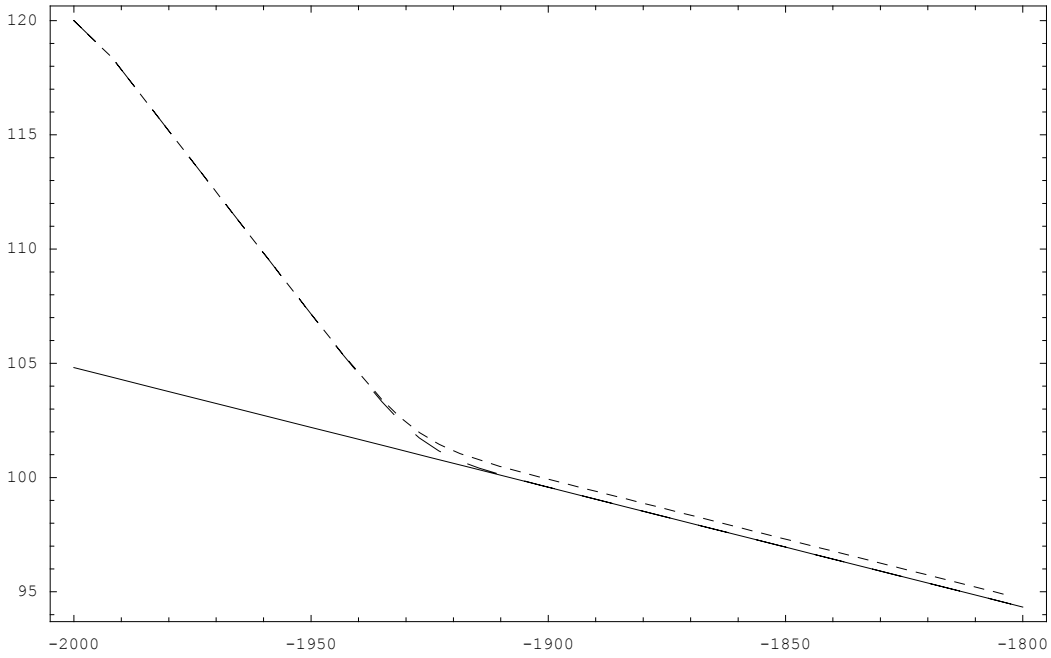
7.1 Control law comparisons

Here we provide some analysis for the two-dimensional (x_1, x_3, θ) glide-slope-recapture controllers. In Fig. 10 we see that the proportional control laws with saturation (II–2D) has a steady-state error, as one might expect from such a proportional controller. This is not apparent in the other control law. This steady-state error could be compensated for by using adding an integral control function, for instance, but this would increase the complexity of this control law in comparison to the other.

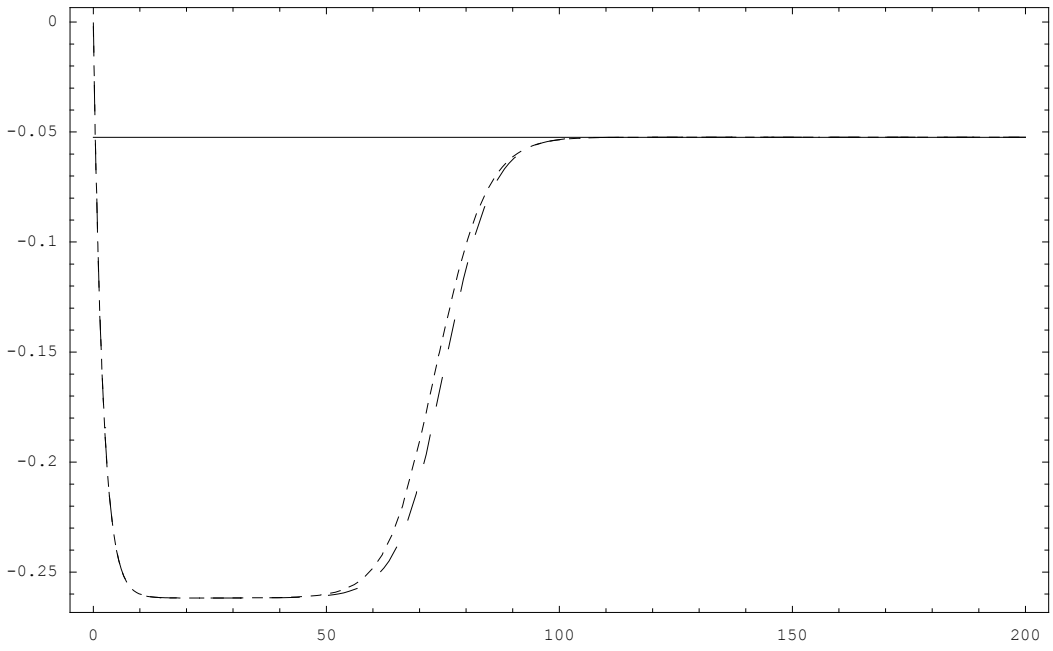
The initial response of the control laws and the initial convergence to the glide slope are very similar for both of the controller as they both effectively use the maximum theta value during this time.

Neither control law exhibits overshoot in the results, however, for both of these control laws this is a matter of tuning the control law parameters. In the case of control law I–2D this is a simpler process in that the single parameter has a direct and predictable effect on the response. For the proportional controller II–2D this tuning is more difficult as the response is very sensitive to the controller parameters.

[4] The virtual runway was used for safety and to allow us to focus on the problem of glide-slope recapture in reducing the complexity through not requiring control of landing gear and not actually performing the landing.



(a)



(b)

Fig. 10 Plots of the two control laws for two-dimensional recapture: I–2D and II–2D. **a** Position plots of the two control laws. The plane starts with initial conditions off the glide slope, and then recaptures according to the two different control laws. The trajectory with steady-state error is the saturated value controller (II–2D). **b** Pitch plots of the two control laws. The trajectory error as shown above is obvious when considering the change in angles with respect to position in x_1

7.2 Decision controller analysis

The ability to generate smooth controllers like those developed in Sect. 3 means that the reachable sets are dependent on the controller, rather than the so-called noninertial dynamics of the aircraft.

In [15] and during the capstone demonstration, a reach set based on these noninertial dynamics was used. That is, when performing the reachability calculations the bang–bang style control was assumed to be viable, meaning that instantaneous changes in angle were possible. In fact, this is not the case, so we decided to compare a simple controller to a smoothed controller (such as those developed in Sect. 3) to check for variations in the reach set.

In fact, variations were noticeable, as shown in Fig. 11. However, these variations in reality reflect the relative smoothness of the control laws applied. For the bang–bang control, which assumes a perfect sign function, sharp edges are evident in the reach set (Fig. 11 b). However, where those edges are sharp in that set, they are smoothed in accordance with the appropriate Γ_ϵ smoothing.

7.3 Broader impact

The experimental success as part of the SEC capstone demonstration drove the research in the direction of application, and we were able to show feasibility of the execution of a safety decision computer. In general, we believe that decision control, as a safety buffer between requests and actions, is a useful abstraction for unmanned vehicle control, and is also extensible to manned vehicles.

7.3.1 Increased safety

- reduced stress and decision load for the pilot
- aircraft training less of a factor than before (due to set calculations for individual aircraft dynamics)
- hyper-accurate safe-set calculations possible, reducing the risk to the pilot

7.3.2 An aircraft-independent execution framework

- it allows for pilots to be trained on one aircraft, but familiar with the procedure on all aircraft
- the computational intensity would be the same for all aircraft (given that the lookup tables are the same size)
- integration strategies can be more uniform, given the common execution framework, across aircraft

7.3.3 Increased level of autonomy

- multiple versions of safe sets increase the effectiveness of the autonomy of the aircraft (in the long term)
- no violation of the operational parameters of the aircraft
- multiple safe sets that can be interchanged to allow modified risk acceptability due to times of war, emergency, or hazardous conditions

7.3.4 Rapid online calculations

- the ability to generate lookup tables that may be queried in hard real time
- lookup/calculation only required at decision points (rather than continuously generating trajectories to satisfy possible safety decisions)
- for multiple definitions of safety it becomes unfeasible to do N concurrent trajectory generations at each time step, whereas N lookups into (an albeit large) memory-based tables is attractive
- additional computational cycles are available for other portions of the inner-loop controller

One final note on analysis, is that we actually do not do a reachability calculation into the *negative* region of pitch (see the figure relating to the altitude recapture). This is a valid assumption since we assume we will be capturing from *above* the glide slope, meaning that if glide slope is 3°, then we will perhaps be using pitch angles between 15° and 0° (to account for overshoot), but we do not anticipate capturing with a large angle of pitch, say -10°. However, this is an area for future work.

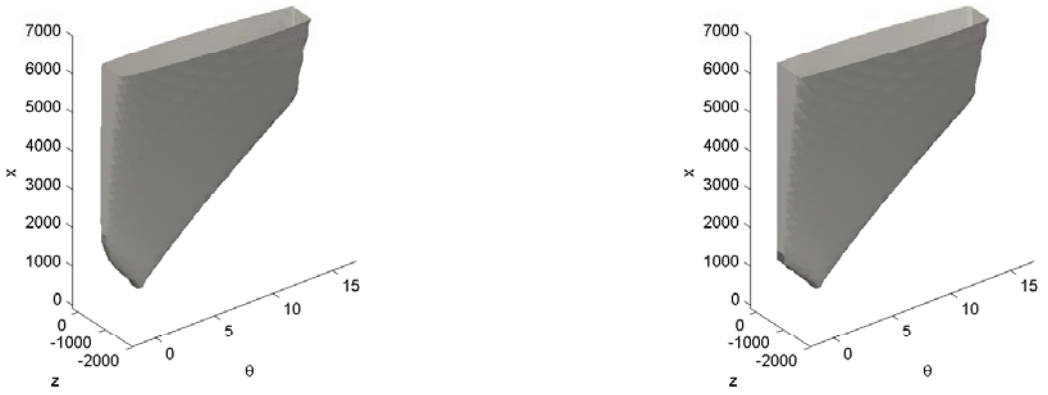
8 Future work

8.1 Reachable-set calculations

The future work of this research revolves around techniques through which new safe sets can be created. We made strides in calculation speed/accuracy by presenting a variable grid size, which provided the ability to have more data points where the dynamics were more complex. An interesting future application of this is a discovery method (not unlike Runge–Kutta) where the emerging shape of the reach set determines the necessary grid points. Although this is nontrivial to implement, it is quite promising as a technology to reduce the domain knowledge required to transform a controller or system into the proper format for reachability analysis.

Although our own expertise biased our choice of reach-set algorithms, to our knowledge there are no other algorithms that could approximate this set for this system's nonlinear dynamics with inputs and state constraints. The flexibility of the HJ formulation allows us to work around the restrictions of a regular computational grid, and permits approximation with either optimal inputs determined during the calculation or predetermined nonlinear feedback controllers. Although expensive to precompute, the resulting implicit surface function is very cheap to consult at runtime for decisions about safely recapturing the glide slope.

We can identify three significant shortcomings of the current approach. The first is the size of the reach set's implicit surface representation, which may be an issue in UAVs with limited memory. This problem may be addressed by various implicit surface-storage compression techniques, at the cost of slightly more runtime computation. The second is the fact that current HJ approximations give no guarantees about the



(a) (b)

Fig. 11 Two reach sets, one with a smoothed control input (Fig. 11 a), and one using maximal values (i.e., bang–bang control)(Fig. 11 b). **a** A reach set which uses a smooth control law. Note that the corners are more rounded, showing that instantaneous value changes in the input are restricted. **b** A reach set which uses bang–bang control. Note the sharpness of the corner of the reachable set, showing that instantaneous value changes are possible

sign of approximation errors. Confidence in the reach set for this problem demands that the result be a guaranteed underapproximation. Related algorithms based on viable sets [4] can make this guarantee, but their discrete set representation requires more nodes for similar accuracy; we are investigating ways of adopting the guarantee without losing the accuracy of implicit surface functions. The third flaw is that high-fidelity system models require more dimensions. We hope that our work on coupled projections may permit us to move in this direction.

A final note on reachable sets is that the process of correctly devising and producing these sets is more of an art than a science. In short, it is nontrivial to choose grid spacings, initial condition ranges, and encode the Hamiltonian multiplication factors when just beginning to use such a toolbox. This creates the need for more advanced tools to perform verification of known good *forward* simulations and perhaps overlay them against the reach sets, to give increased confidence that the sets are progressing backwards in time correctly. It is important to understand that the accuracy of the reach set does not affect the accuracy of the code generator which produces the decision controller: in this case, we contend that what we have provided is the decision *compiler*, while the *code* must be written by the designer.

8.2 Commercial applicability

Some discussion is merited on the applicability of this technique for commercial aircraft. While we concentrated and were driven by an aeronautics application which is quite frankly more common in a military domain, the technology which was developed to solve this problem is applicable in the commercial sector for any decision which might at this time be infeasible due to the complexity of the calculations.

Since our experience and motivations are not in the commercial sector, and we have not researched any such problems in that arena, we hope that experts in the commercial sector will recognize this as an enabling technology to be applied to those existing problems.

8.3 Distributed vehicle formation

Another possible application of this decision control technology would be to provide distributed vehicles with common grounds for optimally configuring themselves into a formation based on knowledge of their global internal state.

An example of this might be a network of vehicles where individual nodes decide their future locations—and the control laws that would take them there—based on the cost of the entire network configuration. For a priori decided algorithms the vehicles could make this decision with no communication, and in a relative rapid manner, simply by following rules set up by a template (see [5] for an application to submerged vehicles where cost of communication is extremely high).

In the vein of reduced communication bandwidth, space-deployed systems such as those to Mars could benefit from a decision controller system which could be easily replicated here on earth, to allow for high-confidence prediction of behaviors, and prevention of risky behaviors.

9 Conclusions

In this paper we have discussed relatively advanced control laws, code generation, aircraft dynamics, safety guarantees,

and discrete switches between controllers. We wish to mention that part of the attraction of this problem of decidability and control lies with the complexity of the domain, coupled with the inherent inflexibility of high-dimensional systems. It is our belief that this sort of problem is indicative of the leading edge of software innovation and best practices in engineering, which we see as an increased need for software experts and complex-systems experts to work together to achieve high-confidence systems.

This is applicable not only to traditional fields such as aeronautics, but also to emerging engineering tasks such as the applied field of critical infrastructure maintenance and understanding, as well as abstract topics such as networks of heterogeneous systems. We believe that to have increased confidence in software which controls our environs and makes safety decisions as well as cost-conscious decisions (such as energy pricing, for example) we will need to continue to address in general the specific problems which we point out in this paper.

Acknowledgements The authors would like to thank Brian Mendel, Jared Rossen, and Jim Paunicka of Boeing Phantom Works, for the integration of the final capstone demonstration of the SEC project. In addition, thanks to Omid Shakernia, Travis Vetter, and Robert Miller of Northrop Grumman who aided greatly in the formulation of the problem. Finally, we acknowledge T. John Koo of Vanderbilt University who provided the impetus for this research through his initial work on the project, given in [9].

References

1. Adams MJ, Tenney YJ, Pew RW (1995) Situation awareness and the cognitive management of complex systems. Hum Factors 37(1):85–104
2. Bayen A, Tomlin C, Ye Y, Zheng J (2004) An approximation algorithm for scheduling aircraft with holding time. In: Proceedings of the 43rd IEEE conference on decision and control, vol 3, pp. 2760–2767
3. Callander BD (1998) The evolution of air mobility. J Air Force Assoc 81(2)
4. Cardaliaguet P, Quincampoix M, Saint-Pierre P (1999) Set-valued numerical analysis for optimal control and differential games. In: Bardi M, Raghavan TES, Parthasarathy T (eds.) Stochastic and differential games: theory and numerical methods, annals of international society of dynamic games, vol 4, Birkhäuser, pp. 177–247
5. Eklund JM, Sprinkle J, Sastry SS (2005) Template based planning and distributed control for networks of unmanned underwater vehicles. In: 44th IEEE conference on decision and control and European control conference ECC 2005 (CDC-ECC'05), (submitted for publication)
6. Endsley MR, Strauch B (1997) Automation and situation awareness: The accident at Cali, Columbia. In: Jensen RS, Rakovan L (eds) Proceedings of the ninth international symposium on aviation psychology, pp. 877–881
7. Greenstreet MR (1996) Verifying safety properties of differential equations. In: Proceedings of the 1996 conference on computer aided verification, New Brunswick, NJ, pp. 277–287
8. Mitchell I (2005) URL http://www.cs.ubc.ca/ mitchell/ToolboxLS
9. Koo TJ, Sastry SS (2003) Hybrid control of unmanned aerial vehicles for autonomous landing. In: Proceedings of 2nd AIAA "Unmanned Unlimited", AIAA, systems, technologies, and operations-aerospace, land, and sea conference
10. Meingast M, Geyer C, Sastry SS (2004) Vision based terrain recovery for landing unmanned aerial vehicles. In: 43rd IEEE conference on decision and control, vol. 2, pp. 1670–1675
11. Mitchell I, Tomlin CJ (2003) Overapproximating reachable sets by Hamilton-Jacobi projections. J Sci Comput 19(1–3):323–346
12. Mitchell I, Bayen A, Tomlin CJ (2005) A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games, IEEE Trans Automat Cont (to appear)
13. Osher S, Fedkiw R (2002) Level set methods and dynamic implicit surfaces. Springer, Berlin Heidelberg New York
14. Sprinkle J, Eklund JM, Kim HJ, Sastry SS (2004) Encoding aerial pursuit/evasion games with fixed wing aircraft into a nonlinear model predictive tracking controller. In: Proceedings of the 43rd IEEE conference on decision and control, vol 3, pp. 2609–2614
15. Sprinkle J, Eklund JM, Sastry SS (2005) Deciding to land a UAV safely in real time. In: Proceedings of American Control Conference (ACC) 2005 (In Publication)
16. Stevens BL, Lewis FL (2003) Aircraft control and simulation, 2nd edn. Wiley-IEEE, ISBN 0471371459
17. Tomlin C, Mitchell I, Bayen A, Oishi M (2003) Computational techniques for the verification of hybrid systems. Proc IEEE 91(7):986–1001