Deciding to Land a UAV Safely in Real Time

Jonathan Sprinkle, J. Mikael Eklund and S. Shankar Sastry

***Abstract*— The difficulty of autonomous free-flight of a fixed-wing UAV is trivial when compared to that of takeoff and landing. There is an even more marked difference when deciding whether or not a UAV can capture or recapture a certain trajectory, since the answer depends on the operating ranges of the aircraft. A common example of requiring this calculation, from a military perspective, is the determination of whether or not an aircraft can capture a landing trajectory (i.e., glideslope) from a certain initial state (velocity, position, etc.). As state dimensions increase, the time to calculate the decision grows exponentially. This paper describes how we can make this decision at flight time, and guarantee that the decision will give a safe answer before the state changes enough to invalidate the decision. We also describe how the computations should be formulated, and how the partitioning of the state-space can be done to reduce the computation time required. Flight testing was performed with our design, and results are given.**

I. Introduction

The history of aviation has seen a gradual increase in the automation provided to fly an aircraft. The prevalence of computerized autopilots and other such technologies in commercial airliners, as well as recent emergence of fly-by-wire aircraft, seems to suggest that much of the automation has emerged in the past few decades with the advent of the digital era.

However, it will surprise many readers to learn that the first automatic fixed-wing aircraft landing actually took place on August 23, 1937. This pre-World War II proof of concept was accomplished mainly because of the ingenuity and hard work of several driven military personnel, as well as a competent electronics engineer. Most importantly—as far as this paper is concerned—it was accomplished on a clear runway where it was the only expected landing.

A. Motivation

This historical example shows that the technology to land *one* fixed-wing aircraft (plane) without pilot interaction is simply an implementation matter. However, when managing dozens of planes (or more) with differing flight characteristics, landing times, fuel constraints, etc., the clear path to implementation is not evident. Automated versions of flight controllers for this task are still not ready for wide deployment.

Between the large problem of managing the arrival and landing of large quantities of planes (we will call this $\mathcal{P}_{many}$), and the small problem of landing one plane ($\mathcal{P}_{one}$), there is the problem of landing *the next* plane ($\mathcal{P}_{next}$). The $\mathcal{P}_{next}$ problem stems from the requirement that while plane p_i is landing, plane p_{i+1} is already on its glideslope[1]. Assuming that there is enough spacing between the planes (i.e., $\mathcal{P}_{many}$ has been solved), then each plane's $\mathcal{P}_{one}$ solution is sufficient to land that plane. However, $\mathcal{P}_{next}$ 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.

So, what does p_{i+1} do now? Ground controllers (solving the $\mathcal{P}_{many}$ problem) will usually instruct p_{i+1} to *vector-off*, i.e., fly back around to the end of the queue of landing planes (or fly a holding pattern) and try to land again. However, 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 glideslope. The question then becomes, *can p_{i+1} recapture the glideslope?*[2] If it can, then the time to land the queue of planes does not increase, and the situation is optimal.

This is the motivation for the experiment we explain in this paper. It extends from the need for improvement in the

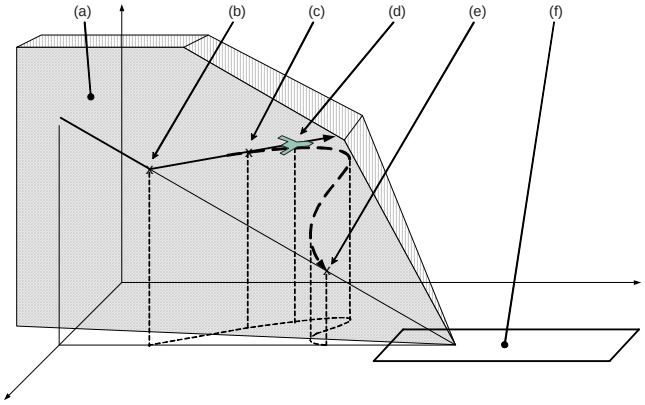


Fig. 1. Demonstration landing scenario. The safe-set of operation (a) exists relative to the desired point of landing on the virtual runway (f). At some point (b) a vector-off maneuver is requested by the ground control to simulate a problem with landing, and the aircraft departs the glideslope in three physical dimensions, as well as velocity, pitch, and roll.. At some point (c) following that the command to land (if possible) is given. At this point the aircraft (d) will continue to vector-off (if landing is unsafe) or will issue commands to recapture the glideslope at some point (e).

This work was supported by the DARPA Software Enabled Control project, under subcontract to Northrop Grumman Corporation, contract number #F33615-99-C-3613.

Dr. Sprinkle, Dr. Eklund, and Prof. Sastry are with the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, CA 94720, USA. e-mail: {sprinkle,eklund,sastry}@EECS.Berkeley.Edu

[1]The glideslope is the angle of descent, velocity, and attitude of the plane as it "glides" in for its landing

[2]It is important to note that the calculation to recapture the glideslope is not the same class of problem as determining whether the current state lies *on* the glideslope, 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.

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 internal feasibility analysis (driven largely by experience and instinct) suggests that it would be too dangerous. Note that the use of the pilot's instinct is more common in a military, rather than commercial, context.

One difficulty with substituting a computer-driven controller for a human controller when solving $\mathcal{P}_{next}$ 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 safe flight 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 boundaries of the aircraft, and should plan to stay within those regions at all times. Since the state of the aircraft is changing at all times, any 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.

B. Solution

In this paper, we detail our solution to this problem, and our description of the computer-driven controller that implements the solution. Further, we justify that this solution is safe for all situations, and that it is feasible to do in real-time (that is, the decisions can be make in flight, on board the plane).

One situation that depends heavily on the pilot's experience is the recapture of the glideslope during a landing sequence where a vector-off command has been issued by ground control. If the state of the runway prevents landing (e.g., debris or stalled aircraft) when an aircraft is on approach, then the ground control will command the approaching aircraft to vector-off. At this point, the aircraft will leave the glideslope and conduct a missed approach procedure. If, however, the runway state improves to allow landing, the ground control can issue a clear/back-to-approach command to the aircraft, alerting it that if it deems a landing is possible, then it is cleared to land.

With the computer-driven controller, at this point one of two mutually exclusive events occurs. If the controller determines that it is feasible to land (i.e., that a set of maneuvers exists that does not exceed the safety constraints of the aircraft, and recaptures the glideslope to allow a landing) then it completes those maneuvers. If the controller does not have a set of maneuvers allowing for a safe landing, then the aircraft will continue the missed approach protocol.

C. Background: Safe Sets

We developed key methodologies and technologies from the theory of hybrid systems and control for the off-line computation of safe-sets using a variant of reachability analysis, followed by online approaches to the computation of feasible control laws for rendering the safe sets invariant, that is staying within the domain of the safe set. Using these technologies, we have the ability to calculate safe sets *offline* and follow controller design characteristics *online* in order to provide more options to the controller while not sacrificing the safety of the aircraft.

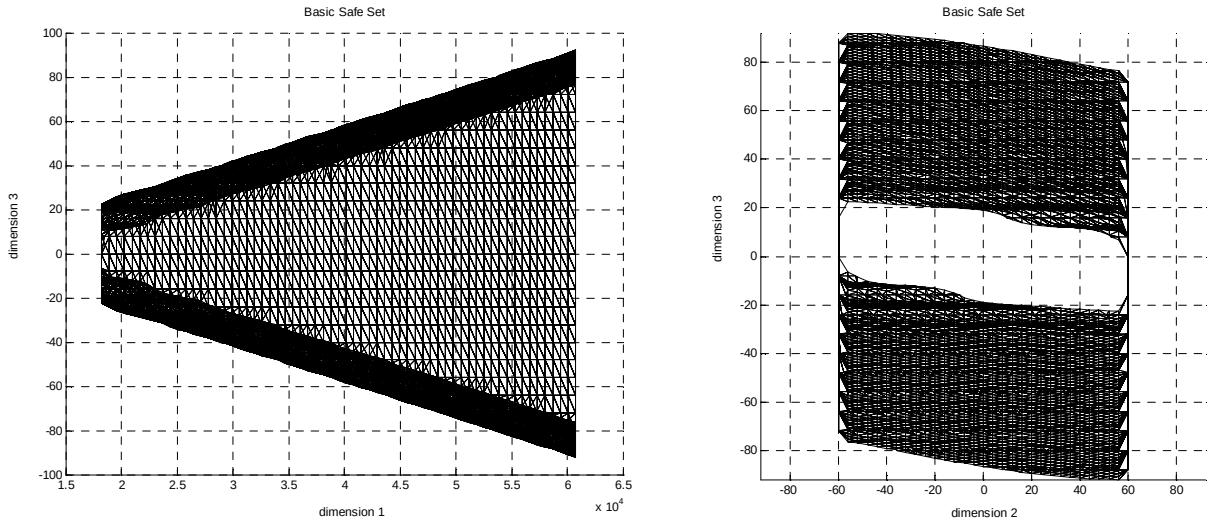
Safe sets are used to determine whether at some point in the future, $t = t_n$, that the system will be in a certain state (e.g., pass through a point, or be operating at a certain velocity). The safe set (see Fig. 2) is the union of all possible states of the system (within some time, or operational, frame) and determination of safety can be gained by checking to see if the desired (or undesired) current or future state lies within this set. The two basic kinds of safe set calculations, forward and backward, are discussed below.

1) Forward Reachable Sets: A forward reachable set is a forward-looking set (from some point in time, $t = t_0$) that involves the calculation or approximation of all future states of the system (up to some time $t = t_N$). Calculation of these safe sets is computationally intensive, and in general is not suitable for online calculation, even in a soft real-time scenario. This is due to the inherently complex nature of exploring *all possible* future states of the system. This scales exponentially with the size of the system state and is therefore impractical to use for any system of size larger than 3 or 4 dimensions, even with offline calculations for evaluation of controller performance (see [1] for more information on the use of forward reachable sets).

2) Backward Reachable Sets: A backward reachable set, conversely, is a backward-looking set (from some point in time, $t = t_N$) that calculates or approximates all *previous* states of the system (until time $t = t_0$). Calculation of these safe sets is also impractical at runtime, for the same reasons as forward reachable sets. However, there is an underlying difference between the backward and forward reachable sets, which makes one more useful than the other for our particular problem. This is described in detail in the next section (see [2] for more information on the use of backward reachable sets).

II. Design

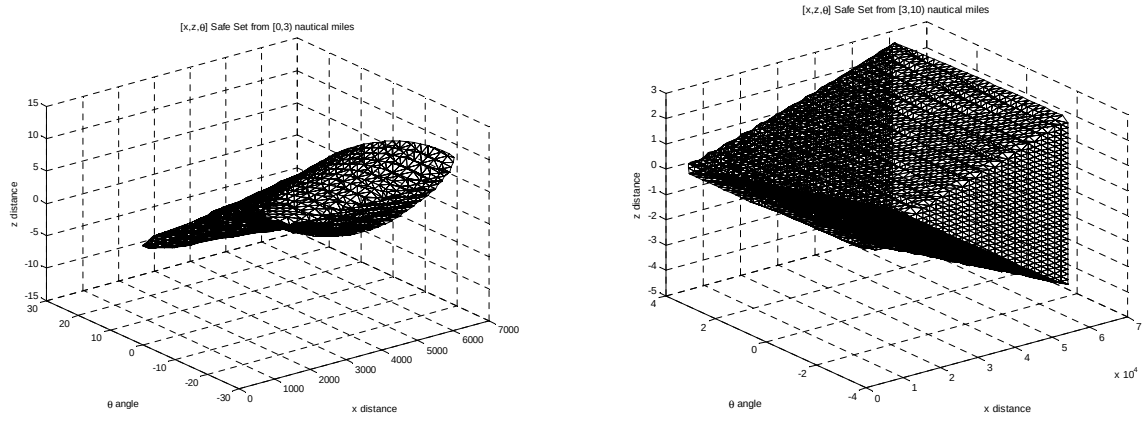
As with any complex problem, the design of the solution can significantly decrease (or increase) the complexity of its implementation. In our case several restrictions on the problem guided our solution significantly.



(a) x, z projection of 3-dimensional safe set.

(b) y, z projection of 3-dimensional safe set.

Fig. 2. An example of a safe set. The "narrow" portion of the cone is the initial set of states, with the flare representing the expanding set of possible states that extend from this initial state set.



(a) The $[0, 3)$ nautical mile portion of the x, z, θ statespace

(b) The $[3, 10)$ nautical mile portion of the x, z, θ statespace

Fig. 3. The two "halves" of the x, z, θ statespace. Because of the relative smoothness of the $[0, 1)$ and $[1, 3)$ statespaces, we were able to optimize the lookup table by keeping these two spaces in the same safe set, resulting in a bipartition rather than a tripartition of the state spaces.

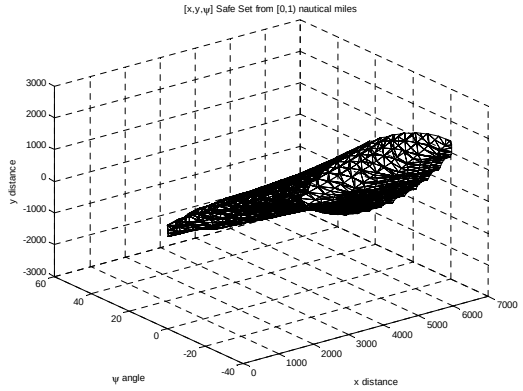
1) The decision whether to recapture the glideslope must be absolutely safe. Therefore, the time required to calculate the decision must not invalidate the safety of that decision.
2) The zone of possible landing states does not change in time or space. Therefore, it may be assumed stationary for the entire experiment.

The use of forward reachable sets to solve this problem is somewhat naïve, since the motion of the plane requires a constant update to the possible set of forward states, and the "ideal" state is completely motionless. This realization makes backward reachable sets an excellent choice. In order to calculate the backward reachable set, we used a Toolbox of Level Set Methods [3], which has been used to solve many reachable set problems [4] (see [5], [6] for more information on level-sets).

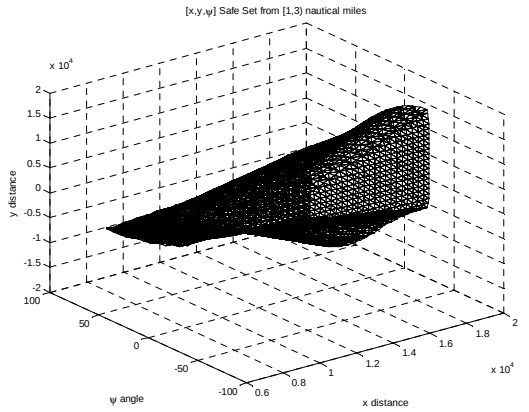
A. Dimensions of the controller

The validation of the safety of the aircraft is bounded by the behavioral limits of the aircraft, as well as the required glideslope and the physical location of the runway. The physical limits of the aircraft are the maximum descent rate ($\dot{z}_{drop}$), minimum velocity (v_{min}), maximum rate of change of pitch ($\dot{\theta}_{max}$), and maximum turn rate ($\dot{\psi}_{max}$). Combined with the three spatial dimensions (x, y, and z), this creates an analysis in six dimensions (since $\dot{z}_{drop}$ is not an input, but a constraint).

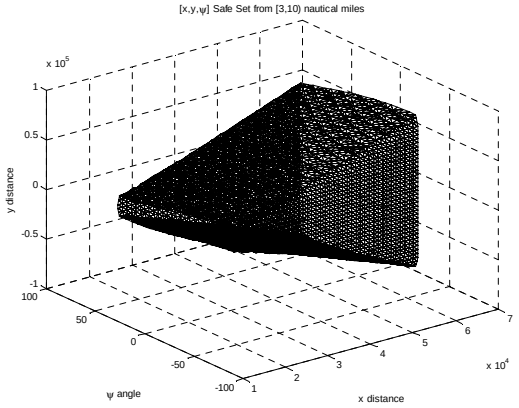
The control input for the aircraft is an open loop autopilot. This controller allows changes to be made to the rate of change of altitude, rate of change of turn, and velocity. The rate of change of altitude and turn, in conjunction with



(a) The $[0, 1)$ nautical mile portion of the x, y, ψ statespace



(b) The $[1, 3)$ nautical mile portion of the x, y, ψ statespace



(c) The $[3, 10)$ nautical mile portion of the x, y, ψ statespace

Fig. 4. The three portions of the x, y, ψ statespace. The proximity partitions "connect" logically where the x values are the same. In this figure, we see that the (a) set visualization is enhanced to a disproportionate level, making it seem unlikely that these partitions fit together. However, when each is scaled at the same level, the pieces fit together.

the current velocity, gives the calculated values for $\dot{\theta}$ and $\dot{\psi}$.

In order to reduce the complexity required, some safe approximations can be made. The first is that the airspeed can remain within an acceptable range during the recapture of the glideslope. This allows the velocity to be treated as constant throughout, reducing the number of dimensions from six to five.

A second approximation, which does not result in any loss of accuracy, is that any dimension of the input control vector can be changed orthogonally to the others. That is, changes to $\dot{\theta}$ will not change $\dot{\psi}$. This guarantee was provided by the open-loop controller. This allows the remaining five dimensions to be separated into two three dimensional problems ($[x, y, \psi], [x, z, \theta]$) with one overlapping dimension (x position in our case). Reducing the dimensions of the problem is important, given the exponential computational costs.

B. Code generation

The methods used to compute backward-reachable sets are in their infancy, and exist as academic programs rather than deployable toolsets. As such, it becomes necessary to convert the results of the calculations into the appropriate technological space (e.g., lookup tables in C++).

Part of the composure of this problem required that the calculations results produced in MATLAB be transformed into C++ for integration into the avionics controller. The development of a code generator to ease this integration provided a dimensionally-independent method to rapidly and automatically produce the required lookup tables which can be seamlessly integrated into a framework that allows queries for safety.

C. Addressing computational complexities

Level-set computational methods are quite complex, and are in general infeasible for large dimensional problems (e.g., 7 dimensions or more). However, the landing scenario can be formulated in 5 dimensions, resulting in a total time calculation of about 30 seconds for 10 nautical miles of approach, once we had done some partitioning of the state space and approach distance.

1) Proximity partitioning: We divided the 10 nautical mile approach into 3 contiguous sections:

$[0,1), [1,3), [3,10]$. Because this 10 nautical mile distance is sufficiently far from the landing zone, there is no need to do calculations on the same scale as, say, 300-ft out. Thus, we were able to run fine-grained calculations (e.g., very small "pixels") for the nearest section, and increase the pixel-size as we got further from the landing zone. Overlap between the sections mitigated safety critical decisions on the cusp of one of the sections.

2) State space partitioning: As previously described, the state space was partitioned into two 3-dimensional spaces (with one overlapping dimension). At runtime, the lookups of the reachable sets scale well, allowing for real-time lookups of less than 10-ms per query (total size of the lookup table is around 7-MB). Because the state space was partitioned, we actually had to do two lookups per check and guarantee that both lookups gave the same answer. If there was any discrepancy in the answer, the overall result was guaranteed to be false (i.e., it is not safe to recapture).

D. Real time decisions

By reformulating the problem as a computation within the *backward* reachable set, then it becomes possible to store the safety calculations as an encoded set of values into which lookups can be done, resulting in drastically reduced complexity in determining safety.

Additionally, by showing that switching between reachable sets can be performed, it becomes possible to store a large set of reachable sets which represent multiple safety or other calculations that can be important in complex situations. Thus, additional constraints (including roaming objectives, such as a moving target) can be instituted into the query engine as additional safe sets.

III. Experiment

The implementation of our solution was tested in our lab, as well as simulated by Boeing using hardware in the loop (HWIL) testing. Finally, the avionics implementation and decision controller was flown in a capstone demonstration on a Boeing T-33 trainer jet in June, 2004, at Edwards Air Force Base. In this section we describe the final capstone demonstration, and the results and analysis we performed with the resulting data.

A. Description

To perform the capstone demonstration we implemented the controller and decision algorithms as a component of the Open Control Platform (OCP). 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. Our T-33 demonstration aircraft was flown under pilot control to a particular waypoint, where the autonomous controller took over and began to land on a virtual runway several thousand feet in the air[3]. A successful landing was achieved by passing through some waypoint (with some bounds for error) at a certain velocity with a certain attitude. Fig. 1 provides a visualization of the overview of the experiment.

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

1) In order 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 glideslope, 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 [7].
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 glideslope.
4) If determined to be safe, the controller will issue commands to recapture the glideslope; if unsafe, the controller stays on its current course of go-around maneuver.

B. Results

While the demonstration itself was scripted, the timeliness of the issuance of commands was left up to the discretion of the experiment controller (in our case, a VIP who was overseeing the capstone demonstration). The experiment was run twice in our case, once resulting in an "unsafe" result, the other resulting in a "safe" landing. In analysis of the results, we determined that the "unsafe" result was actually indicative of an inability to land with the current abilities of the controller, and not a negative response due to an uncertain result.

These results are encouraging, given the relatively small amount of effort required to develop the entire system and implement it into the OCP framework (using academic off-the-shelf tools such as MATLAB, and the Toolbox for Level Set Methods the total development time was less than 4 person-months).

IV. Conclusions

A. Benefits

The proof of concept shown in this paper, including the demonstration that showed feasibility, has many possible uses and advantages over human- and instinct-based landing decisions. The safety of the ground and vehicle is increased, due to the following reasons.

- 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

The design also lends itself to an execution framework that is aircraft-independent.

[3]The virtual runway was used to allow us to focus on the problem of glideslope recapture in reducing the complexity through not requiring control of landing gear and not actually performing the landing.

- 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

In addition, the level of autonomy is increased for UAVs.

- 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

The benefits for rapid online calculations required is also attractive.

- 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 infeasible to do N concurrent trajectory generations at each timestep, whereas N lookups into (an albeit, large) memory-based tables is attractive
- additional computational cycles are available for other portions of the avionics controller

The most important advantage is that safety can be guaranteed. While still in its proof of concept phase, we expect that this technology will be transitioned into UAVs when they are transitioned into a common technology, as opposed to their current state of applied research.

B. Commercial Applicability

Some discussion is merited on the applicability of this technique for commercial aircraft. We do not suggest that such a motivating scenario is sufficient in the commercial sector, since the scenario itself is less frequent than in the military case. However, we do expect that some decisions made in commercial flight control (either on the ground, or in the air) that require heavy calculations may be immediately answered as "no" to err on the side of caution, and in their cases the same technique of offline calculation and online lookup may be applicable to those decisions. Since our experience and motivations are not in the commercial sector, and we have not researched any such problems in that arena, we defer to experts in that field to comment on applicability or future research.

C. Future Work

The future work of this research revolves around increasing the speed at which new safe sets can be created. Although the set generation time is generally speedy, the compilation of the final executable is not. However, this may not be necessary if textual or binary lookup tables can be accessed as files (a feature not available to us in our implementation). Another solution to this problem is to generate already compiled object files from the LSM Toolbox output, rather than C++ files that would need to be compiled.

Finally, a future area of research is the automatic generation of the safe sets from the flight and safety characteristics of a generic aircraft. At this time, a translation of the operating bounds of the aircraft is required, but the possibility exists to automate this for wider deployment of the technology.

ACKNOWLEDGMENTS

Many thanks to Omid Shakernia, Travis Vetter, and Robert Miller of Northrop Grumman Corporation for their help in defining the experiment description. The computational work described in this paper would not have been possible without Ian Mitchell's Toolbox of Level Set Methods [3], as well as his help in expressing the state variables effectively and efficiently. The experiment platform was provided by Boeing Phantom Works, and funded by the DARPA IXO program Software Enabled Control, under the management of Dr. John Bay, to whom we owe the justification for the experiment and research.

REFERENCES

[1] A. Chutinan and B. H. Krogh, "Computational techniques for hybrid system verification," *IEEE Trans. on Automatic Control*, vol. 48, no. 1, pp. 64–75, 2003.

[2] I. Mitchell, "Application of level set methods to control and reachability problems in continuous and hybrid systems," Ph.D. dissertation, Stanford University, Aug. 2002.

[3] ——, "A toolbox of level set methods," Department of Computer Science, University of British Columbia, Tech. Rep., 2004.

[4] C. Tomlin, I. Mitchell, A. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 986–1001, July 2003.

[5] J. A. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[6] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002.

[7] T. J. Koo and S. S. Sastry, "Hybrid control of unmanned aerial vehicles for autonomous landing," in *Proceedings of 2nd AIAA "Unmanned Unlimited"*, ser. Systems, Technologies, and Operations-Aerospace, Land, and Sea Conference. AIAA, Sept. 2003.