UAV as a Reliable Wingman: A Flight Demonstration

S. Waydo, J. Hauser, R. Bailey, E. Klavins, and R. M. Murray

***Abstract*—In this brief, we present the results from a flight experiment demonstrating two significant advances in software enabled control: optimization-based control using real-time trajectory generation and logical programming environments for formal analysis of control software. Our demonstration platform consisted of a human-piloted F-15 jet flying together with an autonomous T-33 jet. We describe the behavior of the system in two scenarios. In the first, nominal state communications were present and the autonomous aircraft maintained formation as the human pilot flew maneuvers. In the second, we imposed the loss of high-rate communications and demonstrated an autonomous safe "lost wingman" procedure to increase separation and reacquire contact. The flight demonstration included both a nominal formation flight component and an execution of the lost wingman scenario.**

***Index Terms*—Aerospace control, command and control systems, cooperative systems, fault tolerance, formal languages, optimal control.**

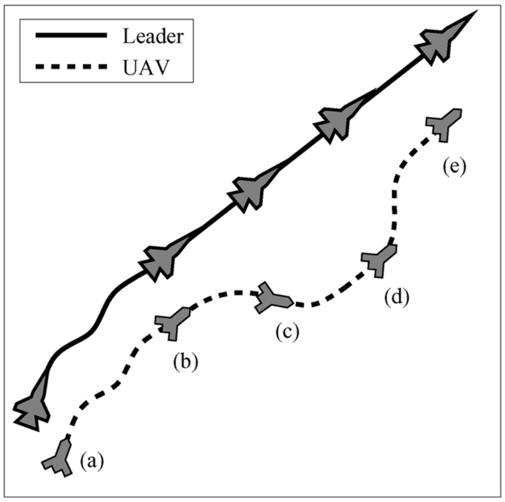


Fig. 1. Lost wingman scenario. At point (a), the aircraft are flying in formation. At (b), the data link is lost and the UAV begins executing the lost wingman maneuver. At (c), the UAV receives a confirmation message from the leader and turns to match heading. At (d), the data link is restored and, at (e), normal operation resumes.

I. Introduction

IN JUNE OF 2004, we demonstrated two new control technologies on board two jet aircraft as part of the DARPA Software Enabled Control (SEC) Program's capstone demonstration: optimization-based control using real-time trajectory generation and logical programming environments for formal analysis of control systems. We tested these technologies by implementing a reliable wingman scenario in which an unmanned air vehicle (UAV) performed formation flying with a manned aircraft. The manned aircraft was an F-15 fighter jet, while the UAV surrogate was a T-33 jet trainer outfitted with X-45 UCAV avionics. During formation flight, the vehicles performed a set of coordinated actions using receding horizon control on the UAV to demonstrate real-time trajectory generation. After demonstrating coordinated formation flight, the UAV simulated loss of the high data rate link between the aircraft and began executing a "lost wingman" protocol designed to (provably) maintain separation between the aircraft using only low-level messages passed along the low data rate link. After safe separation was achieved, the high bandwidth link was restored and the UAV requested a rejoin and executed a safe rejoin maneuver. The demonstration scenario is depicted in Fig. 1.

A nonlinear optimization-based receding horizon controller (RHC) was developed to fly the UAV in formation with the human-piloted aircraft. This controller was implemented using the nonlinear trajectory generation (NTG) software [1]–[3]. We describe the design and implementation of this controller in Section II. A logical programming environment (LPE) framework was used to develop and verify the protocols for executing this scenario. Temporal logic was used to specify and reason about the desired performance and our protocol was implemented using the Computation and Control Language (CCL) [4]. In Section III, we present the LPE framework and its implementation in CCL and, in Section IV, we describe its implementation for this flight demonstration. In Section V, we describe the integration of these technologies into the flight test platform and the results of the demonstration.

The main contribution of this brief is to prove the feasibility of our approach to both nonlinear receding horizon control and formal software modeling and analysis by implementing them on a real flight system operating in an uncertain environment. This flight experiment built on our previous work on applying these tools separately in simulation [5] and on smaller scale experimental testbeds [2], [6], but was the first time they have been joined in a single system. We demonstrated how these two control technologies can work in concert to provide robustness to uncertainty and failures in a dynamic environment and provided a proof-of-concept for how this approach can lead to realizing the goal of human-piloted and autonomous aircraft working together safely.

A. Related Work

Provably safe conflict resolution between aircraft has been extensively studied in the context of air-traffic control [7], [8].

Manuscript received May 1, 2006; revised January 16, 2007. Manuscript received in final form April 4, 2007. Recommended by Guest Editor A. Tsourdos. This work was supported by the DARPA SEC Program, F33615-98-C-3613. The work of S. Waydo was supported by a Fannie and John Hertz Foundation Fellowship.

S. Waydo and R. M. Murray are with the Control and Dynamical Systems Department, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: waydo@cds.caltech.edu).

J. Hauser and R. Bailey are with the Electrical and Computer Engineering Department, University of Colorado, Boulder, CO 80309 USA.

E. Klavins is with the Electrical Engineering Department, University of Washington, Seattle, WA 98195 USA.

Digital Object Identifier 10.1109/TCST.2007.899172

The distinction here is that in addition to verifying the safety of the lost wingman maneuver, we reason about the operation of the underlying decision mechanism (the software).

Several other research teams participated in the SEC demonstration, including at least two applications of receding horizon control. Keviczky *et al.* developed an application programming interface (API) for receding horizon control and demonstrated reference trajectory tracking with the UAV [9]. Schouwenaars *et al.* demonstrated the use of mixed integer linear programming (MILP) implemented in a receding horizon fashion for trajectory generation and task planning [10]. Neither of these demonstrations included a formation flight component and no other demonstration included an LPE component.

II. Receding-Horizon Control With Nonlinear Trajectory Generation

A. *Nonlinear Trajectory Generation Package*

In receding horizon control (RHC), a finite-horizon constrained optimal control problem is solved starting from the current state [11]. The controls of this trajectory are then applied for some portion of the horizon length, after which the optimization is repeated to provide the feedback necessary for stabilization and error correction. Nonlinear Trajectory Generation (NTG) [1], [3], [12] is an RHC software package implementing an efficient computational method combining nonlinear control theory, B-spline basis functions, and nonlinear programming. There are three primary components to NTG. The first is to determine a parameterization such that the equations of motion can be mapped to a lower dimensional space (output space). In particular, we seek a set of *differentially flat* outputs that can be used to convert dynamic constraints to algebraic ones [13]. The second is to parameterize each component of the output in terms of an appropriate B-spline polynomial [14]. Finally, nonlinear programming is used to solve for the coefficients of the B-splines. A more complete description of the NTG framework is described elsewhere [1], [2]. NTG has previously been used to stabilize and perform high-performance flight maneuvers on an indoor vectored-thrust flight experiment [2] and planar hovercraft-like vehicles [6].

B. *Aircraft System Description*

A T-33 jet trainer equipped with a three-axis autopilot was used as a UAV surrogate in the flight demonstration. The T-33 could be directed through low rate (2 Hz) commands to the autopilot specifying the desired altitude and heading (or heading rate). Airspeed commands could also be sent through the same interface, but as the T-33 was not equipped with an actuator to manipulate the throttle these "commands" were only *suggestions* to the pilot as to the desired airspeed. Good judgment on the part of the pilot was essential to the safety and success of the flight demonstration.

With the given interface and flight safety requirements (discussed in Section IV-C), we decided to restrict the desired operations to constant altitude maneuvering with airspeeds within to a reasonable range around a nominal design point. We are thus interested in controlling the aircraft in a 2-D plane (at constant altitude) for the purpose of formation flying including breakup and rejoin maneuvers. Neglecting wind, the kinematics of constant altitude flight are given by

$$\dot{x} = v\cos\chi, \qquad \dot{y} = v\sin\chi \tag{1}$$

where x and y are north and east displacements, v is the velocity, and χ is the heading angle. The velocity coordinate frame is attached to the aircraft center of mass and oriented so that the x-axis lines up with the velocity vector. The acceleration expressed in this frame consists of a longitudinal acceleration $\dot{v}$ and a lateral acceleration $v\dot{\chi}$. For a conventional aircraft, the desired lateral acceleration is obtained by banking the aircraft. Idealizing this situation, we will suppose that the aircraft is flying at constant altitude with zero sideslip angle (coordinated flight) and that the (normally small) angle of attack is equal to zero. The lateral acceleration is then given by

$$v\dot{\chi} = g\tan\varphi$$

where φ is the roll angle and g is the acceleration of gravity. Fixing the velocity, we see that the lateral dynamics of this simplified model includes heading and roll angles χ and φ and turn and roll rates $\dot{\chi}$ and $\dot{\varphi}$. The autopilot provides tracking of the commanded heading χ (or turn rate $\dot{\chi}$) by treating φ or $\dot{\varphi}$ as a pseudo control and using a higher bandwidth regulator to provide close regulation of the roll angle and rate, enforcing limits on the maximum roll rate and roll angle to ensure safe flight operations.

For the purposes of system design and performance evaluation, Boeing provided a simulation environment known as "DemoSim" providing precisely the same interface as used in the flight demonstration. Furthermore, an interface to the DemoSim dynamics was provided for use within the MATLAB/Simulink environment for dynamic analysis and control system design work. Using this environment, we were able to explore the range of maneuvers that would be possible in flight, verifying that the autopilot system (or at least the DemoSim model of it) functioned much as described before. A third order, local linear model describing the mapping from commanded heading to roll, roll rate, and heading was identified from the DemoSim model. This model, with roll rate saturation and time delay included, provided a fast simple model for early testing and evaluation of our control system design. We also identified a second order linear model mapping commanded heading to roll and heading angles for use within the control system. This model was used in developing a simple compensator so that $\ddot{\chi}$ could be used as a pseudo-control.

C. *Localizer Intercept*

A fundamental skill in aircraft navigation is the ability to fly the aircraft to a line in space that goes through a given point with a desired heading χ, a maneuver known as a *localizer intercept*. We use the localizer intercept (and maintenance) as the basis for formation flight where the localizer specification varies as a function of the location and heading of the lead aircraft. For the flight demonstration, the localizer was specified as the line passing through the lead aircraft and oriented with the lead aircraft's heading so that leader–follower behavior would result.

Suppose that we would like to intercept a localizer through the point $(x, y) = (0, 0)$ with heading $\chi = 0$ and suppose that (through independent regulation) the velocity v is constant. The lateral dynamics are then described by

$$\dot{y} = v \sin \chi, \quad \dot{\chi} = (g/v) \tan \varphi, \quad \dot{\varphi} = \dot{\varphi}_{\text{cmd}} \tag{2}$$

where $\dot{\varphi}_{\text{cmd}}$ is thought of as a control (or pseudo-control), possibly subject to actuator dynamics. Note that, with $\dot{\varphi}_{\text{cmd}}$ as the control input, this system is differentially flat. With this setup, the *localizer intercept problem* is to steer the aircraft lateral dynamics to $(y, \chi, \varphi) = (0, 0, 0)$ with local exponential stability subject to constraints on the roll angle and roll rate. We provide a solution to this problem using a receding horizon control strategy.

To this end, suppose that we desire that the aircraft maneuvers with $|\varphi| \leq \varphi_{\max}$ and $|\dot{\varphi}| \leq \dot{\varphi}_{\max}$. Based on experiments with DemoSim, we chose to use $\varphi_{\max} = 30\pi/180$ rad and $\dot{\varphi}_{\max} = 4.5\pi/180$ rad/s. With such limitations, performance of the localizer intercept was best when we restricted the offset heading angle used to approach the localizer, requiring $|\chi| \leq \chi_{\max}$. In practice, we found that $\chi_{\max} = 15\pi/180$ rad worked well, providing a good tradeoff between rapid intercept and requiring excessive speed differential to avoid falling behind the lead aircraft.

As the localizer intercept system (2) is controllable over the intended operating region, local exponential stability may be obtained through the use of receding horizon control with a quadratic cost functional. Exploiting differential flatness, our optimal control problem is the constrained calculus of variations problem

$$\begin{aligned}
&\min \int_0^T \|(y, \dot{y}, \ddot{y})(\tau)\|_Q^2 + \|y^{(3)}(\tau)\|_R^2 d\tau + \|(y, \dot{y}, \ddot{y})(T)\|_{P_1}^2 \\
&|\dot{y}(t)| \quad \leq \quad v \sin \chi_{\max} \\
&|\ddot{y}(t)| \quad \leq \quad g\varphi_{\max} \\
&|y^{(3)}(t)| \quad \leq \quad g\dot{\varphi}_{\max}
\end{aligned} \tag{3}$$

where the constraints provide approximate satisfaction of the physical constraints proposed above. The positive definite symmetric weighting matrices Q and R were chosen to obtain, asymptotically, desirable transient dynamics. That is, we first designed a linear controller that provided reasonable performance with respect to a linear model (plus saturation and time delay) of the system. We then solved the inverse optimal control problem by convex optimization to find Q and R matrices for which this was the optimal controller. By using these Q and R matrices in the RHC formulation, we thus obtained similar behavior away from the constraints, but with a controller that properly respected the constraints as the system approached them. The weight on the terminal state, P_1, was chosen to be consistent with Q and R relative to the model dynamics.

In particular, P_1 is the solution of stationary Riccati equation for the corresponding infinite horizon LQR problem (for the system linearized about straight-and-level flight). A horizon length of $T = 30$ seconds was found to work quite well, providing a sufficient preview to allow a smooth intercept of the localizer while satisfying the constraints. The optimization was recomputed at every 2 Hz computation cycle. The choice of the horizon length and the design of the terminal cost, not to mention the design of the incremental cost, remain somewhat of an art, though theoretical notions that may help guide this process have been developed [15]–[17].

D. Implementation

NTG relies on NPSOL [18], a numerical optimization library that uses sequential quadratic programming to solve nonlinear optimization problems, and PGS, the companion software to the Practical Guide to Splines [14], both of which are implemented in Fortran 77. These utilities were compiled into libraries that were linked to the main executable. The optimization problem (3) was coded in C++ within the main executable and solved by calls to these linked libraries.

III. Formal Methods for Specification and Analysis of Software

In this section, we introduce our approach to using formal methods to specify and verify our reliable wingman algorithms, particularly in the "lost wingman" scenario we implemented for the flight demonstration. The goal of this brief is to be able to specify desired safety and correctness properties of our programs and then write our programs in such a way that we can formally prove they meet the specifications given a model of the world, including aircraft and controller dynamics as well as communications link properties, etc.

A. Computation and Control Language

The computation and control language (CCL) is a programming language designed to make it easy to specify programs together with a model of the environment in a way that makes it possible to prove theorems about them [4], [5]. Inspired by the UNITY formalism [19] and adapted to real-time control systems, CCL is a specification and implementation language for the operation of concurrent systems that interact both through dynamics and logical protocols. The advantages of this approach are as follows.

- The formalism of CCL and the analysis techniques allow us to analyze the dynamical behavior and the logical behavior each in an environment naturally suited to them.
- CCL is both a specification and implementation language, meaning that the protocols we analyze and the code we implement are nearly identical; in some cases they are one and the same.
- Reasoning about CCL specifications is done using standard logical tools amenable to the application of automated theorem proving software.
- We can model and reason about portions of the system using CCL specifications even if they will not be implemented as such.

For this demonstration, we used CCL as both a modeling environment for the complete system and as our implementation language for a portion of the software. We summarize here a few important points about CCL; this material has been described in considerably more detail in previous work [4], [5].

B. Structure and Semantics of a CCL Specification

A CCL specification, or *program*, P consists of two parts I and C. I is a predicate on states called the initial condition. C is a set of *guarded commands* of the form $g : r$, where g is a predicate on states and r is a *relation* on states. In a rule, primed variables (such as x_i') refer to the new state and unprimed variables to the old state. For example

$$x < 0 : x' > y + 1$$

is a guarded command stating: If x is less than zero, then set the new value of x to be greater than the current value of $y+1$. If x is not less than zero, do nothing. Note that guarded commands can be nondeterministic, as is the previous one since it does not specify the exact new value for x, only that the new value must be greater that $y + 1$.

Programs are composed in a straightforward manner: If $P_1 = (I_1, C_1)$ and $P_2 = (I_2, C_2)$, then, $P_1 \circ P_2 = (I_1 \wedge I_2, C_1 \cup C_2)$. This type of composition allows us to modularize our specifications into separate programs such as one for a finite-state machine and one to model the dynamics. In the real system, then, we can implement only the state machine and as long as the real dynamics satisfy the CCL specification any proofs about the original specification will remain valid.

In addition to the initial condition and the collection of guarded commands, we need to specify the *semantics* we will use to interpret the specification. The semantics determine how commands are picked for execution. For the purposes of this brief, where the commands are picked at very close to the same rate, the *EPOCH* semantics will be sufficient [4]. Informally these semantics require that, though commands may be chosen nondeterministically from C, each command must be chosen once before any command is chosen again. Thus, the execution of a system is divided into *epochs* during which each command is executed exactly once. This is an attempt to capture the small-time interleaving that may occur between processors that are essentially synchronized, as well as the nondeterministic ordering of commands that can occur when some are picked by some event-driven process while others are picked by a deterministic process. We generally view epochs as occurring at some fixed frequency, although that has not yet been explicitly modeled.

C. Specification of Experimental System

We now turn to the task of specifying the demonstration system. Here, we provide a few examples of the CCL specifications; the full details and a partial safety proof have been provided elsewhere [20].

1) Dynamics and Semantics: Using techniques described in [20], we specify two programs F_{dyn} and T_{dyn} to keep track of the dynamics of the F-15 and T-33, respectively, using the subscript 1 to denote the F-15 (so for example the position of the F-15 is (x_1, y_1)) and 2 to denote the T-33. These programs are rules that periodically update the aircraft states according to the discrete-time version of (1). We also constrain the execution to obey the EPOCH semantics, with the additional requirement that each epoch begin with the execution of the command describing the dynamics. In the actual system, the execution of each epoch is triggered by an accurate software timer every Δt seconds and each epoch will complete before the next trigger, so this is a realistic model.

2) Controllers: Each aircraft runs a controller that at each update uses its own data plus what is known about the other aircraft to generate controls $\mathbf{u}$. We envision that this is accomplished by some function *control*: $\mathcal{Q} \times \mathcal{Q} \to \mathcal{U}$ (where $\mathcal{Q}$ is the state space of a single vehicle) that we will leave unspecified aside from basic assumptions required for proofs of safety properties [20]. This allows us to use any number of control techniques in the system to be implemented while retaining the correctness of the safety proofs, provided the implemented controller satisfies the safety specification. In particular, we can implement a simple model of the controllers in CCL for analysis and test purposes and a more complex receding-horizon controller for the flight test and retain the validity of our analysis as long as both meet the higher-level specifications. On the T-33 we can implement this as a constraint on the optimization-based controller described in Section II. As with the dynamics, we denote the F-15 controller by F_c and the T-33 controller by T_c.

D. Communication

We suppose there are two communications links between the two aircraft: a "high-speed" data connection for state information and a less frequently used "low-speed" connection. The high-speed link is implemented by the lower-level command and control software, and because the CCL program on the T-33 interacts with this subsystem only to check if new data have arrived we abstract it into a command c_{data} that updates T_s, the next time data will be sent by the F-15, and T, the next time data will be received by the T-33. The send time T_s is incremented by ΔT whenever data are sent, reflecting the fact that the F-15 sends data at a regular rate, while the receive time T can be anything in the interval $(T_s, T_s + \tau_d)$, reflecting an uncertain and nondeterministic time delay of up to τ_d in the system. We keep track of the status of the data link using the boolean variable $\mathrm{data\_on}$.

We model the low-speed communications link using a mailbox with a queue and a nondeterministic time delay of up to τ_c. When a message is sent, a record is added to the end of the queue containing a scheduled arrival time $t_a \in (t, t + \tau_c]$. We write $\mathrm{send}(i, y)$ as shorthand for the process of sending data y to vehicle i. We then have the predicate $\mathrm{in}(i)$ which is *true* if a message has arrived at vehicle i.

As will be seen in the following, the nature of the messages we send is such that we are only interested in the most recent message received. We use $\mathrm{msg}' = \mathrm{recv}(i)$ to denote setting the new value of *msg* to the *data* field of the most recent record in

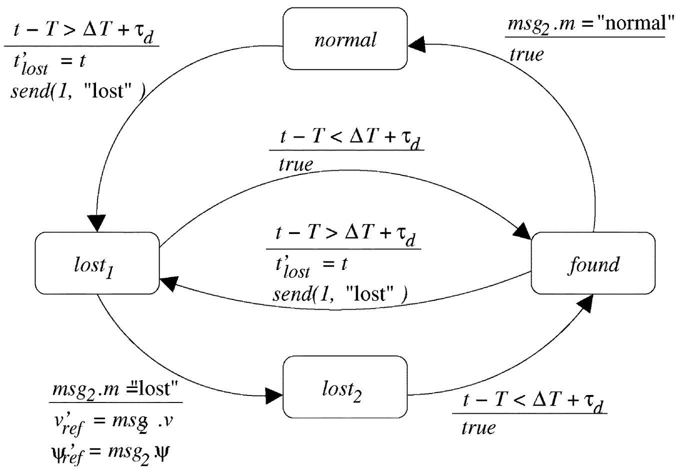


Fig. 2. State machine describing the mode changes and variable updates for the T-33 control system. This machine describes the behavior of the CCL program running on the T-33.

$\text{queue\_}i$ for which $t \geq t_a$ and then deleting from $\text{queue\_}i$ all messages for which $t \geq t_a$.

All of these communications are specified by the program P_{comm}.

Program P_{comm}

$$\begin{array}{ll} \text{Initial} & T_s = t_0 \wedge T \in (T_s, T_s + \Delta T] \\ \text{Commands} & c_{\text{data}} \equiv t > T \wedge \text{data\_on} : \\ & \quad T' \in (T_s + \Delta T, T_s + \Delta T + \tau_d] \\ & \quad \wedge T_s' = T_s + \Delta T \\ & c_{\text{msg},1} \equiv \text{in}(1) : \text{msg}_1' = \text{recv}(1) \\ & c_{\text{msg},2} \equiv \text{in}(2) : \text{msg}_2' = \text{recv}(2) \end{array}$$

IV. Lost Wingman Protocol

The lost wingman protocol consists of two parts: a CCL state machine managing the T-33 and a detailed flight procedure for the pilot of the F-15. The flight procedure can also be written as a state machine, and so, for purposes of analysis, we can model the pilot's behavior using CCL as well.

A. T-33

The state machine running on the T-33 is depicted in Fig. 2. The system has four modes denoted by the variable m_2 in the following programs:

- *normal*, for normal formation flight, abbreviated n;
- lost_1, for the case where the T-33 has ceased receiving state data from the F-15 and has not yet received a confirmation of lost status, abbreviated l_1. In this mode the T-33 executes a turn away from the F-15's last known heading (with right or left determined by the relative positions of the two aircraft);
- lost_2, for the case where lost confirmation has been received from the F-15, abbreviated l_2. In this mode the T-33 matches the speed and heading sent by the F-15 in the confirmation message;
- *found*, for the case where the state data link has been reacquired but normal formation flight has not resumed, abbreviated f. In this mode the T-33 matches heading and speed with the F-15.

This flight demonstration was designed to test what happens when the T-33 enters lost_1 mode from *normal* mode. The portion of the state machine specification involving this portion of operation is

Program T_{sm}

$$\begin{array}{lll} \text{Initial} & m_2 = n & \\ \text{Commands} & c_{\text{lost}} & \equiv m_2 \in \{n, f\} \wedge t - T > \Delta T + \tau_d : \\ & & \quad m_2' = l_1 \wedge t_{\text{lost}}' = t \\ & & \quad \wedge \text{send}(1, \text{"lost"}) \\ & c_{\text{found}} & \equiv m_2 \in \{l_1, l_2\} \wedge t - T < \Delta T + \tau_d : \\ & & \quad m_2' = f \\ & c_{\text{lost2}} & \equiv m_2 = l_1 \wedge \text{msg}_2.m = \text{"lost"} : \\ & & \quad m_2' = l_2 \wedge v_{\text{ref}}' = \text{msg}_2.v \\ & & \quad \wedge \psi_{\text{ref}}' = \text{msg}_2 \cdot \psi \\ & \ldots & \end{array}$$

B. F-15

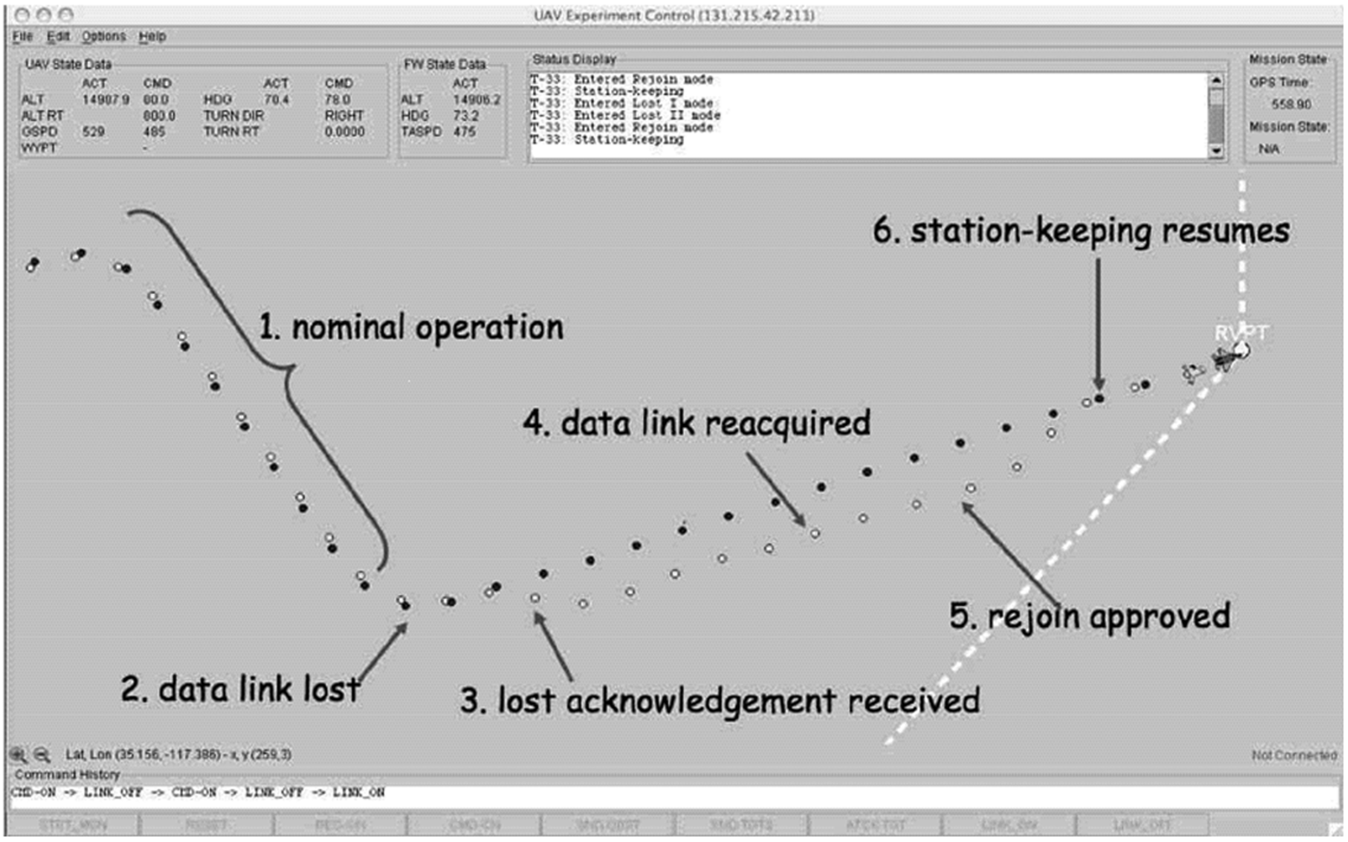
The F-15 state machine consists of just two modes (denoted by m_1 in the specifications): *normal*, for normal formation flight, and *lost*, for the lost wingman scenario. In *normal* mode the pilot is free to fly at will within a predefined performance envelope that ensures safe formation flight is possible. If the pilot receives a "lost" message from the T-33, the procedure is to transition to straight and level flight and transmit to the T-33 the resulting speed and heading (using a button on the test conductor's control interface to do so automatically). Upon receiving and acknowledging a rejoin request and observing that the T-33 has rejoined formation safely, the pilot can resume *normal* operation.

C. Safety

The safety specification we were concerned with in the lost wingman scenario was simple: the aircraft must not collide in the event of a loss of the high data rate link. In [20], we provide a detailed proof that, provided the nominal controller maintains formation within a certain level of accuracy, the T-33 will safely pass through $lost_1$ mode. That is, the T-33 will either make it to $lost_2$ mode, in which it can match speed and heading with the F-15, or to *found* mode in the event that high-rate communication is restored. A highly useful byproduct of constructing this proof is that it yields performance constraints that can be used in the nominal control design. These performance constraints, expressed as bounds on the allowable station-keeping error, also suggest bounds on the allowable uncertainties, but for the purpose of this demonstration all measurements (and shared state data) were assumed to be accurate (a very good assumption for our particular test platform). In principle, it would be straightforward to extend the analysis of [20] to explicitly model uncertainty (or other performance requirements) as well.

D. Implementation

While we may reason about specifications in which the rules are arbitrary relations on states, any code we write will be deterministic, meaning that all rules will be assignments. In this case



	ACT	CMD		ACT	CMD
ALT	14987.9	00.0	HDG	70.4	78.0
ALT RT		000.0	TURN DIR		RIGHT
GSPD	529	485	TURN RT		0.0000
WYPT		-			

	ACT
ALT	14906.2
HDG	73.2
TASPD	475

```
T-33: Entered Rejoin mode
T-33: Station-keeping
T-33: Entered Lost I mode
T-33: Entered Lost II mode
T-33: Entered Rejoin mode
T-33: Station-keeping
```

GPS Time: 558.90
Mission State: N/A

Lat, Lon (35.156, -117.386) - x, y (259,3)

```
CMD-ON -> LINK_OFF -> CMD-ON -> LINK_OFF -> LINK_ON
```

Fig. 3. Screen capture depicting the experiment controller interface on the UAV computer. Results from a portion of a simulation including the lost wingman behavior are shown. The path of the F-15 is marked with dark circles (plotted at 10-s intervals) while the path of the T-33 is marked with lighter circles. Depicted area is about 40-km wide.

there exists a CCL interpreter known as CCLi [4], [21] that can be used to execute CCL code. The implementation code for all of the programs described here is available online [22].

Because we have modularized the different aspects of our CCL specification into different programs, we are free to implement these specifications in whatever manner is most appropriate for each. For this demonstration, the T-33 state machine and low-speed communications protocols were implemented directly in CCL programs updated by the primary executable, while the high-speed communications and receding-horizon control were implemented in the main C++ source code.

For development and testing of our algorithms, we implemented a simulation of the system and our controllers in CCL in addition to the state management code used for the flight test. This allowed us to quickly iterate and test our designs in a simple desktop environment prior to integrating with the complex system required to fly the actual aircraft. When it came time to integrate CCL with the flight system, the CCLi library was linked with the flight code so our CCL protocols could be periodically updated from within the demonstration executable. Implementing the CCL code in this way rather than recoding it in C++ (the language of the flight code) had two significant advantages. First, we could have a very high level of confidence that our flight protocols would execute just as those we analyzed and tested because the code was identical—any mismatch would be due to differences in hardware and operating conditions and would most likely be very small. Second, because the version of CCL we used is interpreted rather than compiled, this strategy allowed us to modify our protocols and retest without having to go through the time consuming process of recompiling the main executable.

The F-15 state machine was implemented by the detailed flight procedures provided to the test crew specifying what actions to take upon receiving messages from the autonomous wingman. Messages from the F-15 to the T-33 were either automatic (in the case of the detailed state information) or sent by the test conductor pressing buttons on a laptop computer interface (described in Section II). For the testing of our algorithms a keyboard controllable model of the F-15 was written in CCL and used to demonstrate the lost wingman scenario.

V. Flight Demonstration

A. Demonstration Overview

The main objectives of the flight experiment were to do the following:

1) demonstrate the use of NTG to perform coordinated formation flight; and
2) demonstrate the use of CCL to manage the lost wingman scenario.

As described, the UAV test platform was a T-33 jet aircraft outfitted with the avionics package of the X-45 UCAV unmanned aircraft. To interface with the test system, we were required

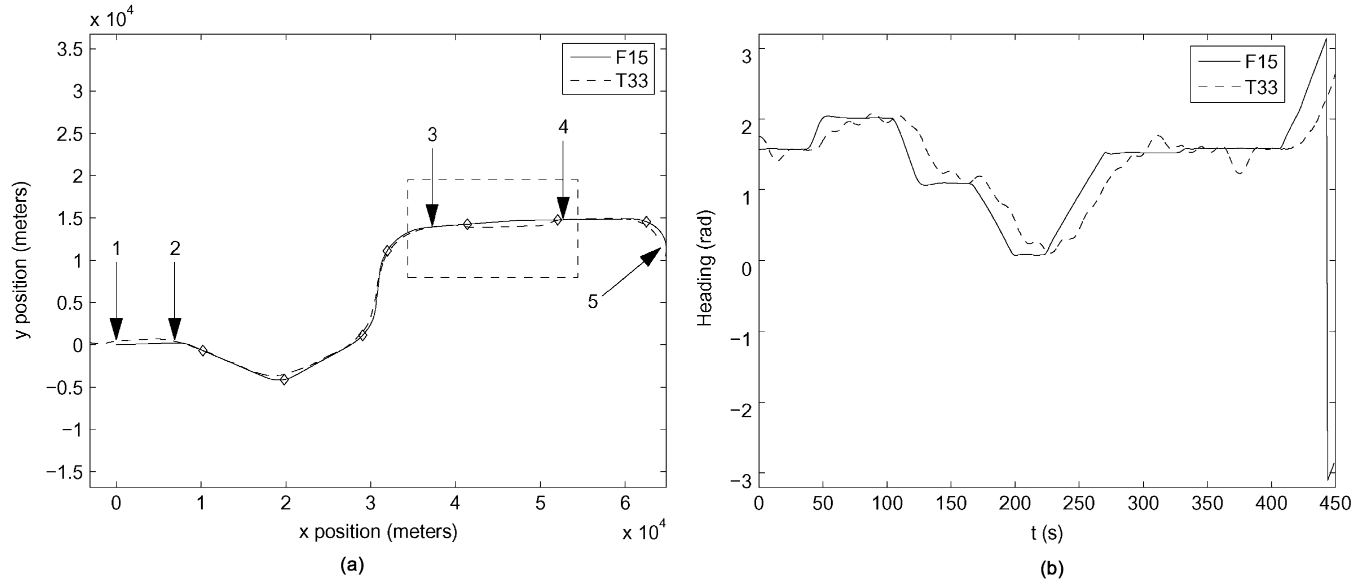


Fig. 4. Flight test results from Flight Test 6 on June 25, 2004. (a) Flight paths. Diamonds mark the position of the F-15 at 60-s intervals. Dashed box is area of interest for Fig. 6. Points of interest: 1) Experiment start; 2) divert maneuvers; 3) lost wingman start; 4) lost wingman complete; 5) experiment finish. (b) Aircraft heading. Time lag between F-15 and T-33 trajectories is the result of both inherent time delay in the flight system (communications delay, etc.) and the large (> 1 mi) separation distance between the aircraft.

to integrate our technologies with the Open Control Platform (OCP) [23], [24], a middleware development platform developed at Boeing as part of the SEC program. The OCP comprised the main structure of the control code, collecting the aircraft's state information and providing hooks through which our software could process this data and generate control inputs. The OCP also handled communications between the two aircraft, automatically providing the transfer of state information needed for formation flight and allowing us to pass other information such as that required to execute the lost wingman scenario.

On both aircraft the test engineer (riding in the aircraft rear seat) interacted with the control software using a Java-based "experiment Controller" interface. This interface is shown with simulation data generated by running our scenario in DemoSim in Fig. 3. On the T-33 side the test engineer had buttons to cut off and restore the high-speed data link to test the lost wingman performance. On the F-15 side, the test engineer had a button to confirm the lost wingman status upon receipt of a lost message from the T-33 and another button to approve a formation rejoin request from the T-33 upon restoration of normal state communications.

Our test scenario proceeded in three stages. First, the F-15 pilot was instructed to fly a nondeterministic path within the test range to test the station-keeping performance of the NTG-based controller. In the second stage, the test conductor eliminated the high-rate data link between the aircraft and the lost wingman behavior was observed. Finally, the high-rate link was restored and the T-33 was allowed to execute a rejoin maneuver and resume station-keeping. At all times the aircraft maintained at least a 1-mi separation for safety reasons; for the purposes of the demonstration a violation of this separation would be considered a "collision." A test pilot on board the T-33 handled flying the aircraft to and from the test area and was available to intervene in the case of problems with the autonomous system.

B. *Flight Test Results*

The flight control experiment was executed seven times over five flights between June 17 and June 25, 2004. We here describe the results from Tests 6 and 7 on June 25, in which the aircraft had to be diverted away from other traffic, resulting in an unplanned set of maneuvers that nicely stressed the formation keeping capabilities of the T-33. Test 6 included an execution of the lost wingman protocol. Figs. 4(a) and 5(a) provide an overview of the flight paths of both aircraft taken over the experiments. Separation between the aircraft was generally maintained at about 1.5 mi, and at no point did the aircraft approach the 1-mi minimum separation limit.

C. *Control Performance*

As can be seen from Figs. 4(a) and 5(a), these flight tests included several maneuvers induced by the avoidance of other traffic in the test airspace. The execution of the lost wingman scenario occurred 4–6 min into Test 6 [in the dashed box in Fig. 4(a)] and was the only occasion in which the T-33 deviated significantly from the flight path of the F-15 (by design of the protocol).

Figs. 4(b) and 5(b) depict the heading of the two aircraft during Tests 6 and 7, respectively. The T-33's heading matched that of the F-15, subject to a significant time delay incurred by the control and communications software. The significant deviations at about 300 and 380 s in Test 6 correspond to the lost wingman divert and rejoin maneuvers. The other small deviations were caused by the F-15 making tighter turns than the autopilot of the T-33 would allow the controller to follow.

For the purpose of safe and intuitive (to the pilot) operation, the autopilot restricts the aggressiveness of the maneuvers that it attempts. For example, experiments with DemoSim indicated that the autopilot would restrict the roll rate to approximately 4

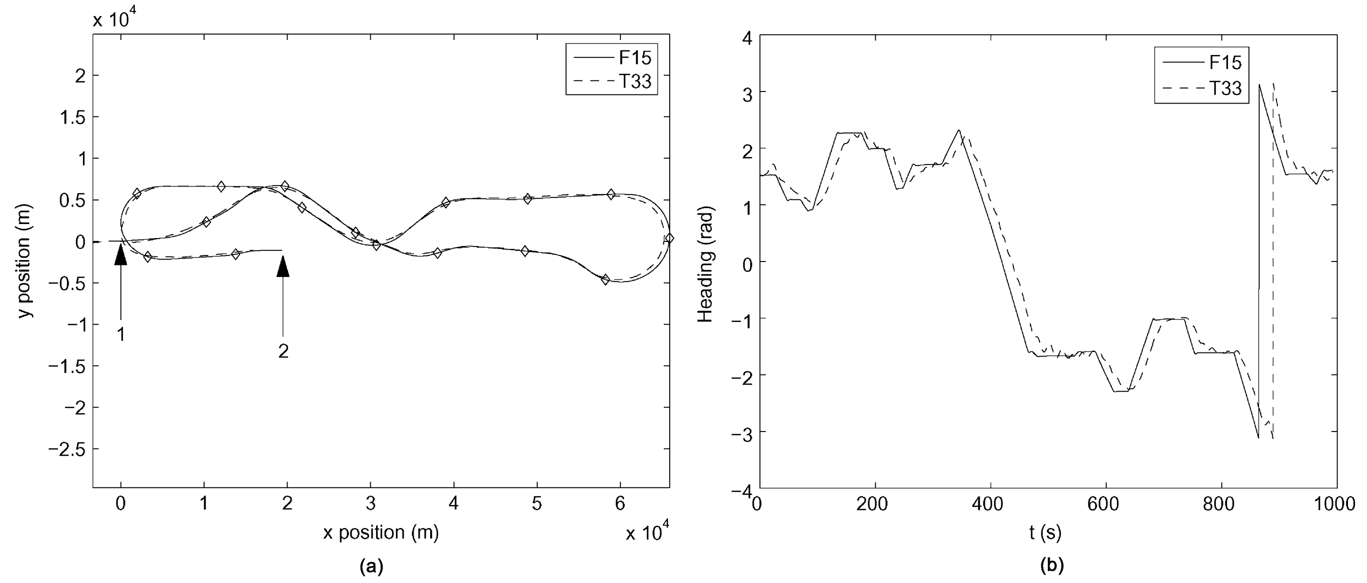


Fig. 5. Flight test results from Flight Test 7 on June 25, 2004. (a) Flight paths. Diamonds mark the position of the F-15 at 60-s intervals. Points of interest: 1) experiment start; 2) experiment finish. (b) Aircraft heading.

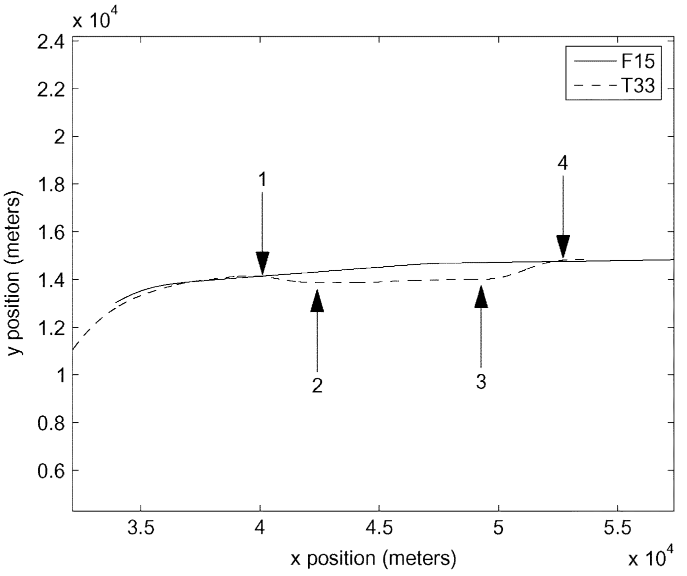


Fig. 6. Flight path of the two vehicles for the lost wingman portion of Flight Test 6 on June 25, 2004. Points of interest: 1) lost maneuver start (Lost 1 mode); 2) lost heading established (Lost 2 mode); 3) rejoin maneuver start (found mode); 4) rejoin maneuver complete (normal mode).

or 5 degrees per second. In flight, we observed typical roll rates of 5 to 6 degrees per second with a maximum of approximately 9 degrees per second.

D. Lost Wingman Performance

Fig. 6 depicts the portion of Test 6 in which the lost wingman protocol was executed. At point (1) the high-rate state communications were cut off and the T-33 automatically transitioned to its lost state upon detecting this condition. From point (1) to point (2), the T-33 executed a tight turn to ensure it would stay clear of the F-15 while broadcasting its lost status via the low speed link. At point (2), the lost confirmation message was received by the T-33, having been sent by the F-15 test engineer pressing a confirmation button on the laptop interface after straight and level flight was achieved. At this time the T-33 turned to match the heading commanded by the F-15. At point (3), the state communications were restored and the T-33 automatically requested a formation rejoin, which was granted by the F-15. At point (4), the T-33 completed its rejoin maneuver and nominal operation resumed. As can be seen from the similarity between this plot and the schematic of Fig. 1 and the simulation results of Fig. 3 the lost wingman procedure executed exactly as designed.

VI. Lessons Learned

Much of the success of this flight demonstration can be attributed to our control design properly respecting the performance constraints on the flight system imposed by having to command the aircraft through the autopilot. The RHC framework allowed us to explicitly capture these constraints in our control design, resulting in achieving the best performance that could be expected given the capabilities of our UAV surrogate. We found that we obtained the best performance when we ensured that the constraints were appropriately conservative, that is, when we made sure our controllers would not attempt to push the flight performance too close to the allowable limits.

As previously described, the choice of parameters for the RHC problem (horizon length, costs, etc.) is still in large part an art requiring significant expertise and extensive simulation. The tactic we used here, namely choosing desired dynamics for the simplified unconstrained system and solving an inverse optimal control problem, was found to be a very effective approach to this problem.

These successes depended heavily on the accuracy of the DemoSim environment provided by Boeing for experiment development. The entire flight test program took place over about two weeks after extensive testing of our control software in simulation. Because of the required testing and the flight demonstration schedule, no revision of our control software after the start of flight tests was feasible. This meant that our control design was entirely based on the system identification we were able to do on the DemoSim model. Much more than in laboratory environments where iteration based on "flight" tests is the norm, a highly accurate simulation environment was critically important. It is within this environment, for example, that the need for conservative constraints on flight performance mentioned before first became clear (by observing poor performance under more relaxed constraints), enabling us to tailor our controllers appropriately to ensure a successful demonstration.

On top of the accuracy of the simulation module, the ability to implement our controllers and interact with DemoSim first in a MATLAB environment and later in a full OCP software environment was crucial. The MATLAB interface to DemoSim enabled the system identification upon which our controller design was based, a process which would have been massively more cumbersome had we needed to operate the full flight software to do experiments for identification. This interface also allowed us to more quickly iterate on our control design without the overhead involved in the flight software implementation. Because the MATLAB simulation was one and the same as the simulation used with the flight software, we were able to implement our final control design in the OCP with high confidence in its success.

ACKNOWLEDGMENT

The authors would like to thank B. Mendel and J. Rossen at the Boeing Company for their extensive help with the integration of our software with the Open Control Platform.

REFERENCES

[1] M. Milam, "Real-time optimal trajectory generation for constrained dynamical systems," Ph.D. dissertation, Control Dynamical Syst. Dept., California Inst. Technol., Pasadena, 2003.

[2] M. Milam, R. Franz, J. Hauser, and R. Murray, "Receding horizon control of a vectored thrust flight experiment," in *Proc. IEE Proc. Control Theory Appl.*, 2005, pp. 340–348.

[3] R. Murray, J. Hauser, A. Jadbabaie, M. Milam, N. Petit, W. Dunbar, and R. Franz, Eds., "ch. Online Control Customization via Optimization-based Control," in *Software-Enabled Control: Information Technology for Dynamical Systems*. New York: Wiley, 2003, pp. 149–174.

[4] E. Klavins, "A language for modeling and programming cooperative control systems," in *Proc. Int. Conf. Robot. Autom.*, 2004, pp. 3403–3410.

[5] E. Klavins and R. M. Murray, "Distributed algorithms for cooperative control," *IEEE Pervasive Comput.*, vol. 3, no. 1, pp. 56–65, 2004.

[6] J. Chauvin, L. Sinegre, and R. Murray, "Nonlinear trajectory generation for the Caltech multi-vehicle wireless testbed," presented at the Eur. Control Conf., Cambridge, U.K., 2003.

[7] C. Tomlin, I. Mitchell, and R. Ghosh, "Safety verification of conflict resolution maneuvers," *IEEE Trans. Intell. Transport. Syst.*, vol. 2, no. 2, pp. 110–120, Feb. 2001.

[8] Z.-H. Mao, E. Feron, and K. Billimoria, "Stability and performance of intersecting aircraft flows under decentralized conflict avoidance rules," *IEEE Trans. Intell. Transp. Syst.*, vol. 2, no. 2, pp. 101–109, Jun. 2001.

[9] T. Keviczky, A. Packard, O. Natale, and G. Balas, "Application programming interface for real-time receding horizon control," in *Proc. Conf. Dec. Control Eur. Control Conf.*, 2005, pp. 1331–1336.

[10] T. Schouwenaars, M. Valenti, E. Feron, and J. How, "Implementation and flight test results of milp-based UAV guidance," in *Proc. IEEE Aerosp. Conf.*, 2005, pp. 1–13.

[11] R. Findeisen and F. Allgö, "An introduction to nonlinear model predictive control," presented at the 21st Benelux Meet. Syst. Control, Veldhoven, The Netherlands, 2002.

[12] California Inst. Technol., Pasadena, "Nonlinear trajectory generation," (2002). [Online]. Available: http://www.cds.caltech.edu/~ntg/

[13] M. Fliess, J. L'evine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *Int. J. Control*, vol. 6, pp. 1327–1360, 1995.

[14] C. D. Boor, Ed., *A Practical Guide to Splines*. New York: Springer, 2001.

[15] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, pp. 789–814, 2000.

[16] A. Jadbabie, J. Yu, and J. Hauser, "Unconstrained receding-horizon control of nonlinear systems," *IEEE Trans. Autom. Control*, vol. 46, no. 5, pp. 776–783, May 2001.

[17] A. Jadbabie and J. Hauser, "On the stability of receding horizon control with a general terminal cost," *IEEE Trans. Autom. Control*, vol. 50, no. 5, pp. 674–678, May 2005.

[18] Stanford Business Software, Inc., Mountain View, CA, "NPSOL," (2006). [Online]. Available: http://www.sbsi-sol-optimize.com/asp/sol_product_npsol.htm

[19] K. Chandy and J. Misra, *Parallel Program Design: A Foundation*. Reading, MA: Addison-Wesley, 1988.

[20] S. Waydo and E. Klavins, "Verification of an autonomous reliable wingman using CCL," California Inst. Technol., Pasadena, Tech. Rep. CaltechCDSTR:2006.001, 2006.

[21] Univ. Washington, Seattle, "The computation and control language (CCL)," (2004). [Online]. Available: http://sveiks.ee.washington.edu/ccl/

[22] California Inst. Technol., Pasadena, "Verification of an autonomous reliable wingman using CCL," (2004). [Online]. Available: http://www.cds.caltech.edu/~waydo/lost_wingman/

[23] J. Paunicka, B. Mendel, and D. Corman, "The OCP - an open middleware solution for embedded systems," in *Proc. Amer. Control Conf.*, 2001, pp. 3445–3450.

[24] "Open Control Platform: A software platform supporting advances in UAV control technology," in *Software-Enabled Control: Information Technology for Dynamical Systems*J. Paunicka, B. Bendel, and D. Corman, Eds. New York: Wiley, 2003, pp. 38–62.