MULTI-VEHICLE COOPERATIVE CONTROL FLIGHT TEST

Anawat Pongpunwattana, Richard Wise, Rolf Rysdyk
University of Washington, Seattle, WA
Anthony J. Kang, The Boeing Company, St. Louis, MO

Abstract

During the spring and summer of 2005, the Autonomous Flight Systems Laboratory at the University of Washington[1] and Boeing St. Louis developed and flight tested control software for the Boeing Multi-Vehicle UAV Testbed. The goal of this collaboration was to demonstrate the ability of a team of UAVs to autonomously allocate a set of targets among the team and fly to the targets while avoiding designated areas. A distributed architecture was used maximizing the benefits of increased autonomy in the areas of task allocation and path planning. The real-time decision making component consists of the Evolution-based Cooperative Planning System (ECoPS) developed under DARPA funding. The approach was demonstrated in simulations and in a flight test at Fort Leonard Wood, Missouri.

Introduction

Unmanned Aerial Vehicles (UAVs) are valuable in many civil and military applications. Currently operational UAVs require a skilled team of technicians and engineers to closely monitor and control the system during the operation. To increase autonomy, the UAVs must have the ability to navigate safely in a hostile environment and make decisions to perform the assigned tasks effectively. To demonstrate such capability, a research team at the Boeing Company in St. Louis, Missouri, developed the Multi-Vehicle UAV Testbed. This testbed provides a modular system for prototyping technologies for cooperative planning and control for a team of UAVs.

Together with the Boeing St. Louis team, the Autonomous Flight Systems Laboratory at the University of Washington (UW) developed control software for the testbed. The objective was to develop efficient real-time control and communication software and to demonstrate its performance in flight. The end product is a distributed control software with efficient algorithms for computation of cooperative trajectories for a team of UAVs to accomplish a common goal set. The algorithms must: 1) be suitable for real-time implementation onboard the UAVs, 2) adapt to changes in the environment, 3) avoid collisions, 4) generate feasible trajectories using uncertain information of the environment, and 5) deal with moving and unexpected "pop-up" targets. The Evolution-based Cooperative Planning System (ECoPS) [1] is used as the real-time decision making component of the software. ECoPS is a distributed planning system for team coordination and path planning. The planning algorithms are based on techniques from evolutionary computation and market strategies.

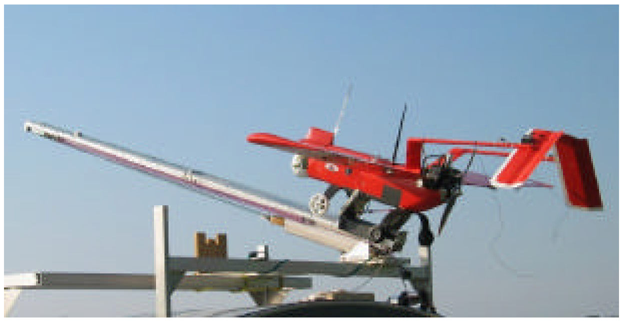

Figure 1. BAT unmanned autonomous vehicle prepared for launch

Multi-Vehicle Testbed

The Boeing Multi-Vehicle UAV testbed consists of a fleet of UAVs and a communication and control infrastructure. The aircraft are BAT UAVs manufactured by the MLB Company, Figure 1. The six-foot wingspan UAV is equipped with an autopilot system which accepts navigation commands through radio communication. The aircraft have a gross weight of 15 lbs which includes a 4 lbs payload and have an endurance of 2

[1]www.aa.washington.edu/research/afsl

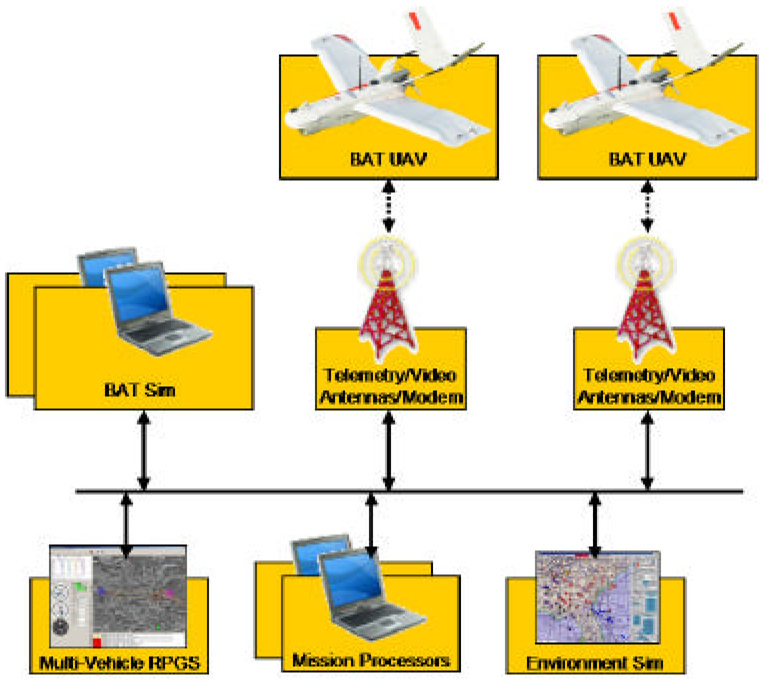

Figure 2. Architecture of the Multi-Vehicle UAV Testbed

to 2.5 hours flying at 25 to 50 mph. The aircraft payload consists of a geo-stabilized 3-axis nose-mounted camera providing real-time video feed to a ground station.

The testbed has a modular design for controlling both real and simulated UAVs simultaneously. Figure illustrates the architecture of the testbed and its components. The testbed consists of five main components: a simulated or real UAV, a multi-vehicle ground control station, an environment simulation, mission processor hardware, and a gateway controller. The simulated UAV is used for initial lab testing and participating along side real UAVs for large multi-vehicle experiments. The Rapid Prototyping Groundstation (RPGStn) is a ground control station that allows the control of up to four real or simulated UAVs. The environment simulation is a Boeing-developed software package that is used to create a multitude of interactive targets and threats. It can also simulate on-board sensors to stimulate the rest of the testbed. The mission processor hardware is a platform for hosting experimental vehicle controller software. It provides access to telemetry downlink and control uplink. It also provides a method for multiple vehicles to interact with each other. The final component is the gateway controller. This is an application often hosted on the mission processor hardware that allows one or more controllers access to the real and simulated UAVs. This application allows the experimental controllers, the ground control station and back-up safety systems to all have access to the UAVs.

The ground-based computing and communication infrastructure is based on the Open Control Platform (OCP). OCP is used as a software tool for developing control algorithms. It provides integration with design tools like MATLAB/SIMULINK in common operating systems (OS) and enables distributed simulation of multiple unmanned vehicles. OCP was developed by the Boeing Company as part of the Defense Advanced Research Projects Agency's “Software Enabled Control” research program. A major component of OCP is the object-oriented middleware called the Real-Time Common Object Request Broker Architecture (RT CORBA), which allows OCP to be used for developing distributed control and communication applications. It provides computing services and an application program interface (API) with graphical tools for automatic code generation facilitating the development of real-time distributed software. Figure 3 shows the hierarchical components of OCP.

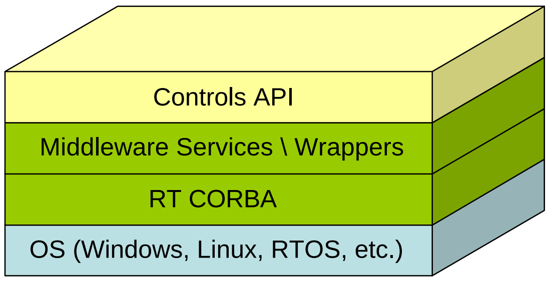
Controls API
Middleware Services \ Wrappers
RT CORBA
OS (Windows, Linux, RTOS, etc.)

Figure 3. Open Control Platform's components

Control System

Each vehicle is controlled by a mission processor/gateway controller which shares information with the ground station and other vehicle controllers via network communication. The vehicle team is provided a set of tasks at the beginning of a mission. Each vehicle's controller makes coordinated decisions in choosing its tasks and computing paths that support the completion of its tasks. The computed paths are sent to the UAV in the form of waypoints and to the ground station for visualization and monitoring. The controller autonomously adapts its task plan and paths to changes in the environment.

Controller Behaviors

The UW controller software is a multi-threaded application created using OCP and consists of two sets of behaviors:

1. Data-driven Behaviors:
   - decodes and processes incoming inter-vehicle communication data.
   - decodes and processes incoming data received from the RPGStn.
   - formats incoming sensor data and updates target information.
   - sets initial flight plan and Home-Waypoints.
   - updates vehicle position and ground speed.
   - updates vehicle altitude and air speed.
2. Timer Behaviors:
   - 1 Hz:
     - controls the execution of the on-line planning process
     - controls the creation and transmission of waypoints to the vehicle and the RPGStn.
   - 5 Hz:
     - controls the off-line planning process.
   - 10 Hz:
     - processes inter-vehicle communication messages for team coordination.

Waypoint Generation

The waypoint generation and transmission is controlled within the controller's 1 Hz timer behavior which also governs the planning process. Planning occurs in two phases, off-line and on-line. Off-line planning executes prior to the start of the autonomous mission. On-line planning starts upon reception of an activation command from the RPGStn. Once activated, a certain length of the path is committed to as the *committed trajectory*. The on-line planning process continues to adapt the path beyond the committed trajectory. Segments of the evolving path are added to the committed trajectory continuously during the flight.

A counter within the 1 Hz timer behavior controls creation and transmission of waypoints as follows:

- counter < 40
  - The current position of the vehicle relative to the position of the next waypoint is compared.
  - If the vehicle is near the next waypoint, its waypoint data is updated.
  - New waypoint data is transmitted to the vehicle.
  - The counter is incremented.

- counter >= 40
  - The evolving path is added to the section of committed trajectory to construct a *total trajectory.*
  - The total trajectory is discretized by normalization based on the size of the operational field area and a standard number of waypoints.
  - Trajectory discretization is done evenly in time.
  - Waypoint data is calculated at appropriate times.
  - Waypoint data is converted to the format used by the vehicle and the RPGStn.
  - Waypoint data is transmitted to the vehicle and the RPGStn.
  - The counter is reset to zero.

Due to bandwidth limitation, a flight plan sent to the vehicle in each update is defined by only four waypoints sequentially listed as: 1) Home-Waypoint 2) Current-Waypoint 3) Next-Waypoint and 4) Home-Waypoint. Home-Waypoint is located at the ground station. Current-Waypoint is the first waypoint ahead of the vehicle at time of transmission. Home-Waypoint is added to the list at both the beginning and the end for fail-safe reasons.

Planning System

Most of the computation performed by each controller pertains to planning. The function of the planning system is to schedule and allocate tasks to those vehicles that can achieve them optimally. A *team objective function* is formulated based on mission objectives and corresponding cost assessment, e.g. in terms of threat exposure, vehicle payload, endurance, and range. The planning system is based on a distributed market-protocol to optimize the team objective function, Figure 4. This scheme is motivated by the optimization process in market economies and team decision theory [2]. Under the *market protocol*, the vehicles choose their own tasks and plan their routes to benefit the team. Completing tasks will bring reward to the team, which is weighed against the cost associated with the task and path execution.

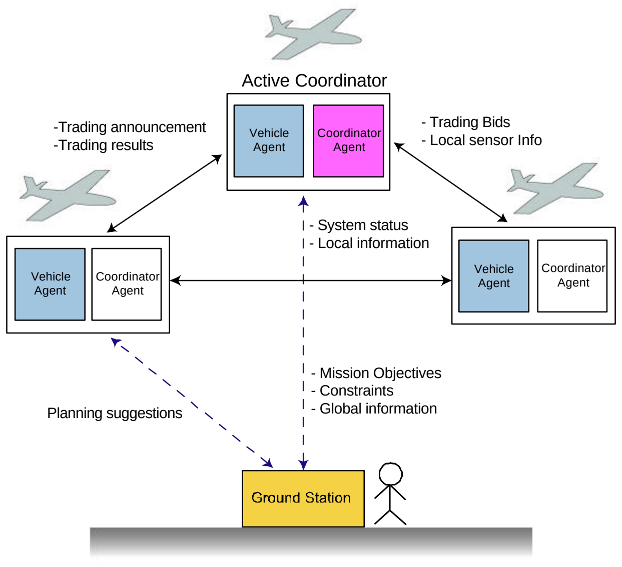

Figure 4. Distributed market-protocol based planning system.

The market-protocol based system considers the vehicles as participants in a market that trades in tasks. In this system, vehicles buy or sell tasks through a coordinator agent that enable organized and efficient trading and facilitate human operator interaction with the system. Coordinator agents are implemented on each vehicle, but only one is active at any given time. The computational burden to run a coordinator agent is small relative to that for making trading decisions. A significant advantage of coordinator agents is the continuous, albeit low-bandwidth, monitoring of team members and task allocation. If a vehicle is unable to communicate, the active coordinator agent will put its tasks up for auction. Conversely, if the active coordinator agent is damaged, a simple election mechanism designates a new coordinator agent from among the communicating team members.

Vehicle Agents are key components of ECoPS containing a *Planner* and a *Communicator* which interact with the active coordinator agent. The Planner serves as the brain of the vehicle agent, making trading decisions to obtain a set of tasks and computing the best task sequence and corresponding paths. An Evolutionary Computation (EC) based technique is used to solve the optimization problems.

Market-Based Algorithm for Task Allocation

This section describes the task trading mechanism in ECoPS. The details of each step are described in the next section and decision making algorithms can be found in [1]. Figure 5 illustrates the concept of the task trading. Consider a system with N_V vehicles and N_T tasks at trading round n. Each vehicle i owns a set of tasks $\mathbf{T}^i$. The task allocation $\mathbf{A}$ can be expressed as

$$? \, (n) = \left\{ \mathbf{T}^1(n), \mathbf{T}^2(n), \ldots, \mathbf{T}^{N_V}(n) \right\} \tag{1}$$

The set of all tasks, $\mathbf{T}$, is given by

$$\mathbf{T}(n) = \bigcup_{i=1}^{N_V} \mathbf{T}^i(n) \tag{2}$$

where $\mathbf{T}^i(n) \subset \mathbf{T}(n)$. The task allocation optimization problem can be defined by

$$\max_{\mathbf{A}} J(\mathbf{A}) \tag{3}$$

where J is the team objective function. Vehicle i submits a set of tasks $S^i \subset \mathbf{T}^i$ in a sell-bid at a price $p(S^i)$. The set of all tasks submitted for sale is expressed as

$$\Psi(n) = \bigcup_{i=1}^{N_V} S^i(n) = \left\{ S^1(n), S^2(n), \ldots, S^{N_V}(n) \right\} \tag{4}$$

The set of all possible tasks vehicle i can buy at trading round n is given by

$$\begin{aligned} \Phi^i(n) &= \bigcup_{j=1,\, j\neq i}^{N_V} S^j(n) \\ &= \left\{ S^1(n), \ldots, S^{i-1}(n), S^{i+1}(n), \ldots, S^{N_V}(n) \right\} \end{aligned} \tag{5}$$

Vehicle i bids on a set of task $B^i \in \Phi^i$ at a price $p(B^i)$. At the end of trading round n, the set of tasks owned by vehicle i is

$$\mathbf{T}^i(n+1) = \mathbf{T}^i(n) \cup B^i(n) - S^i(n) \tag{6}$$

and the task allocation can be written as

$$\mathbf{A}(n+1) = \left\{ \mathbf{T}^1(n+1), \ldots, \mathbf{T}^{N_V}(n+1) \right\} \tag{7}$$

The goal of task trading is to find $\mathbf{A}(n)$ such that

$$\lim_{n\to\infty} \mathbf{A}(n) = \mathbf{A}^* \tag{8}$$

where $\mathbf{A}^*$ is the optimal solution of the task allocation problem, Equation (3). This goal can be accomplished if all vehicles make trading decisions such that

$$J(\mathbf{A}(n+1)) > J(\mathbf{A}(n)), \quad \forall n \in \{1, 2, \ldots\} \tag{9}$$

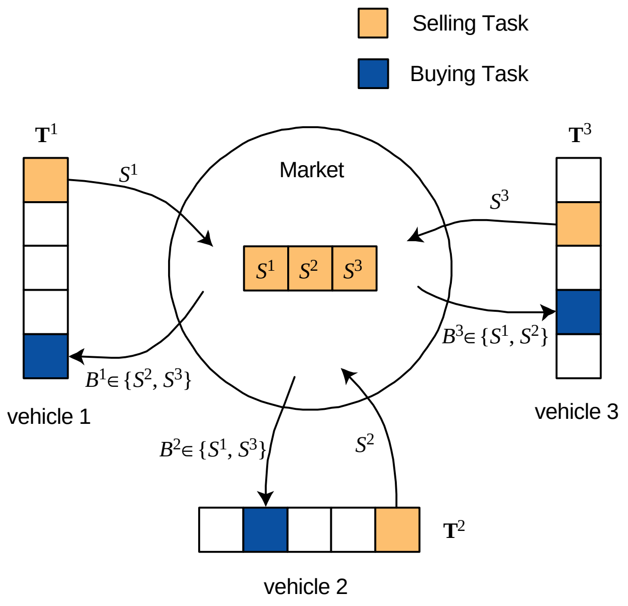

Figure 5. Diagram illustrating the concept of task trading in market-based planning systems.

The most significant part of the mission problem formulation is the definition of the *team objective function.* In ECoPS, this function captures the dynamics and uncertainties associated with the mission. The order of events and actions during the mission affect the function output continuously. For a vehicle with multiple tasks, the cost assessment of the possible task execution sequences will include the level of threat exposure, chances of successful task completion, and time required for task execution. The degree of certainty of information is also accounted for in the objective function, which is thus affected when new information becomes available.

The team objective function J is composed of two main terms and can be written as:

$$J = \sum_{i=1}^{N_T} R_i - \sum_{v=1}^{N_V} C_v \tag{10}$$

The first term represents the expected mission score gained upon task completion. The second term is the expected total cost of task completion, which depends on the probability of survival and the amount of fuel used by all the vehicles. The probability of task completion depends on the probability of survival up to that time. Survivability is computed from the *degree of intersection* of vehicle paths with the distribution of possible obstacle locations. At each time step, the expected mission score is computed by summing the score of expected task completions. Therefore, a simulation is executed for each candidate plan to predict the value of all system states and the objective function. Details of the objective function are provided in [1].

ECoPS Task-Trading Communication

Task trading in ECoPS is done by means of inter-vehicle communication, as described in this section. The task allocation process is initialized as follows:

- The active coordinator agent broadcasts all the unassigned tasks to the team.
- Each vehicle agent submits a bid for a task.
- The coordinator sends bidding results to all vehicle agents.
- Each vehicle agent updates its task plan and
- The active coordinator updates the task allocation table.

These steps repeat until all the tasks are assigned. Then the main task trading starts. Figure 6 shows the task trading sequence in one round of trading which is composed of four main phases: selling, buying, confirming, and transferring:

- The selling phase starts as the coordinator agent broadcast a request for sell bids to all the vehicle agents. The request includes a response time limit. The vehicle agents determine which of their tasks should be put up for sale, and transmit their sell bids to the coordinator agent. A sell bid consists of a *trade item*, in this case a set of tasks and its sell price. The sell price is the minimum bid price of the the buying phase, also referred to as the reserved price.
- The buying phase starts when the time limit of the sell bid request expires, or when the coordinator receives sell bids from all the vehicle agents. The coordinator agent gathers all the trading items and broadcasts a request for buy bids including a time limit. Each of the vehicle agents makes a decision to bid on one of the trading items and submits its buy bid to the coordinator agent.
- The confirming phase is a part of the trading sequence to allow vehicle agents to retract their buy bids before the final trading results are announced. This phase starts when the time limit of the buy bid request expires, or when the coordinator receives buy bids from all the vehicle agents. Given the submitted responses, the coordinator agent determines and broadcasts the initial trading results which indicate whether each bid will be accepted or rejected, again with a time limit for response. Next, each vehicle agent decides to either retract or confirm the bid, and transmits that to the coordinator agent.
- Lastly, the transferring phase starts when the time limit of the confirmation request expires, or when the coordinator agent receives confirmation messages from all the vehicles. The coordinator agent then finalizes the trading results and broadcast them to the vehicle agents. Finally, each vehicle agent updates its task plan accordingly to the received trading results.

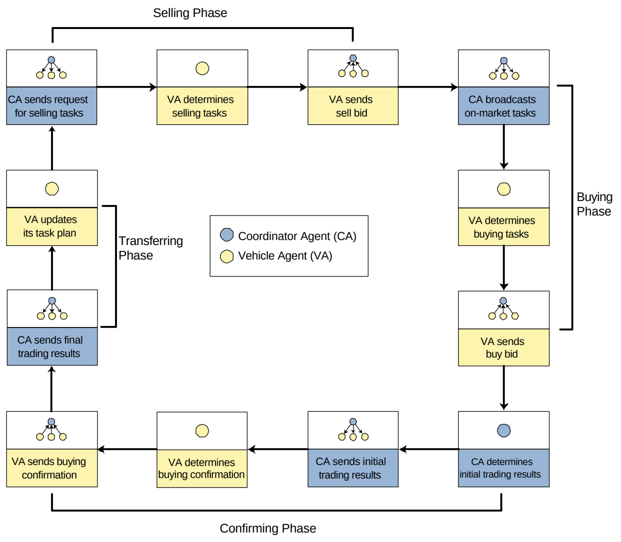
Selling Phase
CA sends request for selling tasks
VA determines selling tasks
VA sends sell bid
CA broadcasts on-market tasks
Buying Phase
VA determines buying tasks
VA sends buy bid
CA determines initial trading results
CA sends initial trading results
VA determines buying confirmation
VA sends buying confirmation
Confirming Phase
CA sends final trading results
VA updates its task plan
Transferring Phase
Coordinator Agent (CA)
Vehicle Agent (VA)

Figure 6. Task trading sequence in one round of trading

ECoPS Path Planning

The path-planning process generates a vehicle motion plan in support of the execution of vehicle tasks. This process is composed of two intrinsically linked elements: task-sequencing and planning of an effective path between task locations. To allow the vehicle to respond effectively to changes in its environment, the path-planning process continuously generates and evaluates feasible paths through a sequence of task location *waypoints* up to its final *goal* location.

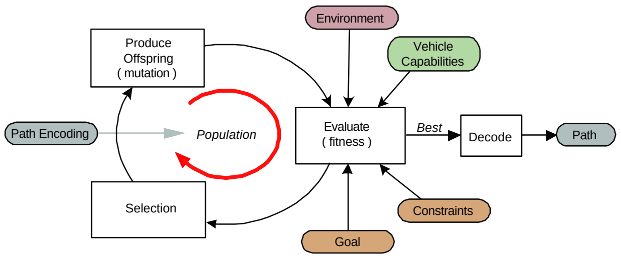

Figure 7. Diagram of EC-based path planning algorithm

ECoPS is based on both Genetic Algorithms (GA) and Evolutionary Programming (EP) [3]. Work by Capozzi [4] suggests that combining features of both paradigms can improve the optimization process. An EC-based path planning algorithm comprises path encoding, fitness evaluation, selection scheme, and mutation mechanisms. Figure 7 illustrates the planning process of the algorithm.

A path is encoded as a sequence of chained parameterized segments, Figure 8. The locations of the vehicle and the goal are shown as a blue triangle and a green circle respectively. In the application to the Boeing Multi-Vehicle UAV testbed, we considered two segment types: straight lines and constant radius curves, Figure 9. The segment parameters are limited to keep motion within the vehicle capabilities. Continuity is enforced between adjoining segments. At all times, each candidate path is extended to the goal location by addition of the necessary number of *go-to-goal* segments.

The fitness function associates a value with each candidate path which captures the dynamics and uncertainties in the system. The fitness value is defined as the inverse of the expected value of the *loss function* given by

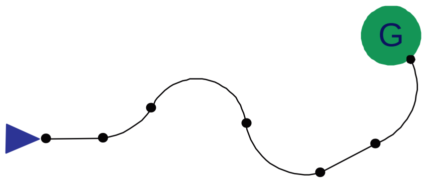

Figure 8. An encoded path composed of a chain of connected segments

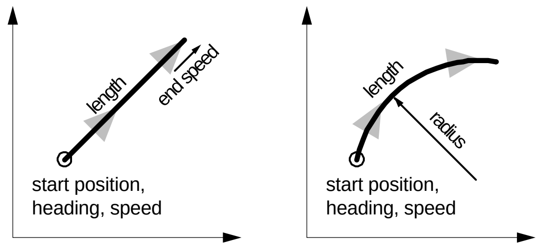
length
end speed
start position, heading, speed
length
radius
start position, heading, speed

Figure 9. Parameterized elemental path segment types

$$L_v = a^L \left(1 - \frac{J_v}{a^F_{sum}}\right) \qquad (11)$$

where J_v is the individual vehicle objective function which can be described similarly to the team objective function given in Equation (10) as

$$J_v = \sum_{i=1}^{N_T} R_i - C_v \qquad (12)$$

a^L is a scaling factor, and a^F_{sum} is defined by

$$a^F_{sum} = \sum_{i=1}^{N_T} a^F_{i,\max} \sum_{j=1}^{N_T} d^v_{ij} \qquad (13)$$

and

$$a^F_{i,\max} = \max_k a^F_i(k), \quad k \in \{0,1,\ldots,N\} \qquad (14)$$

where d^v_{ij} is a task-level decision variable with $d^v_{ij} = 1$ if vehicle *v* plans to execute task *i* at sequence number *j*, otherwise $d^v_{ij} = 0$. $a^F_i(k)$ is a time-dependent score weighting function of task *i* at time step *k*. It is used to define a time window for each task execution. This function adds a high positive value during the time period in which the task should be executed, and small or zero value while the task does not contribute to mission objectives.

If there is no task in its current task plan, the loss function is given by

$$L_v = a^L \left(1 - \frac{J_v}{a_v^V}\right) \quad (15)$$

where a_v^V is the vehicle cost weighting factor.

Given a task plan D_v and path Q_v, the cost function is a linear combination of the probability of losing vehicles during the mission, $\Delta x_v^V(N)$, and the amount of fuel required to travel along the path $F_v(Q_v)$. The cost function $C_v(D_v,Q_v)$ can be expressed as

$$C_v(D_v, Q_v) = a_v^V(\Delta x_v^V(N)) + a^Q F_v(Q_v) \quad (16)$$

with

$$F_v(Q_v) = 1 - Fuel(N) \quad (17)$$

where *Fuel(N)* is the fuel ratio remaining in the vehicle. The weighting factors a_v^V and a^Q are parameters set based on the assessment of the cost of vehicle loss, and fuel consumption.

ECoPS generates new paths by combining (crossover) two paths randomly selected from the current population or by copying a path and altering (mutating) it using one of the following mutation mechanisms chosen at random.

- *Crossover* - takes the starting segments of one path and the ending segments of another and join the two sets.
- *Mutate 1-Point* - randomly changes the parameters of one or more segments, and re-locates the subsequent segments unchanged to enforce the new end-point constraints. The first segment to be mutated and the number of segments to be mutated are selected at random.
- *Mutate 2-Point* - randomly changes the parameters of one or more segments, computes their new end point and reconnects to the start of another segment further along the original path. The segments are chosen at random.
- *Mutate Expand* - adds one or more randomly created segments onto the end of the path. All original parts of the path are left untouched.
- *Mutate Shrink* - removes a random number of segments from the end of the path.

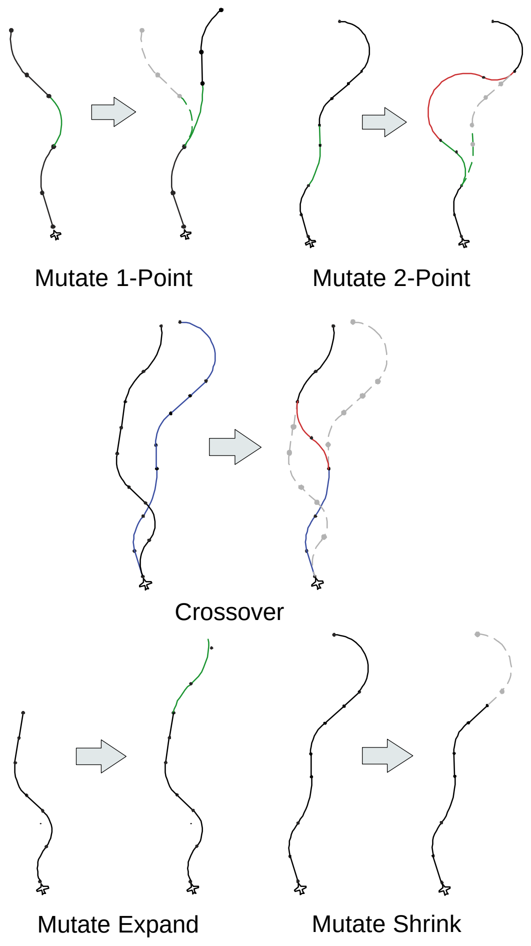

Figure 10. Crossover and mutation mechanisms

In each evolution step, a portion of the *population* of candidate-paths is selected using a q-fold binary tournament selection scheme. The procedures of the selection scheme can be described as follows. Considering a total population of $\mu + l$ paths, for each individual path $i \in \{1, 2, \ldots, m + l\}$:

1. Draw $q \geq 2$ paths randomly from the population (excluding path *i*) with uniform probability $1/(m + l - 1)$. Denote these *competitors* by the indices $\{i_1, i_2, \ldots, i_q\}$.

2. Compare path *i*'s fitness against each of its competitors i_j, $j \in \{1, 2, \ldots, q\}$. Whenever the fitness of path *i* is not worse than that of competitor i_j, path *i* receives a point.

The score of each candidate-path is an integer in the range [0, *q*]. After the scores of all path-candidates are determined, the μ best paths are selected as the *parents* for the next generation.

Simulation and Flight Test Results

Real-time simulation and flight testing demonstrate that ECoPS generates flight plans effectively and quickly adapts its guidance commands to changes in the environment. The test scenario includes two UAVs, three targets and three obstacles. The mission objective is to visit all targets, avoid obstacle areas, and stay within the bounded operating field. The mission is initialized by launching both UAVs and sending them into holding patterns at different altitudes. The autonomous mission is started with a command transmitted by the flight operator.

Snapshots of a simulation run are shown in Figure 11. Obstacle and operating field boundaries are shown as dotted yellow lines. The crosshair-like symbols marked with numbers represent targets. Frame (a) in Figure 11 shows a set of flight plans of the two UAVs generated while orbiting the first waypoint prior to the start of the mission. The blue and green solid lines are the flight plans of the blue and green vehicles respectively. Targets are efficiently assigned to the UAVs, and series of waypoints are generated to guide the UAVs to the target locations without flying through the obstacle areas. Flight plans are updated periodically during the operation to improve the initial plans. Frame (b) shows the flight plans of the UAVs approaching their first targets. The past trajectories of the two vehicles are shown as light blue and light green lines. The result shown in Frame (c) was captured when the two vehicles reached the targets. Frame (d) shows that both vehicles have successfully flown to all targets and are returning to the home location.

The scenario in Figure 12 is similar to the first one, except that the UAVs are not aware of target 3 prior to the mission. The initial flight plans are shown in Frame (a). Once the two vehicle are on their way to the known targets, the information of target 3 is transmitted to the vehicles. Frame (b) shows that the green vehicle effectively adapts its path to the new set of tasks which include the popped-up target 3. In frame (c), the two UAVs have visited targets 1 and 2. The green UAV is heading to the last target and the blue UAV is returning to the home location. Frame (d) shows that the green vehicle has visited target 3 to complete the mission.

Results from the flight test are shown in Figure 13. Because of mechanical difficulties and bad weather during the flight operations, we were able to run only a single vehicle mission. Frame (a) shows the initial flight plan represented by the blue line. Only the first ten waypoints of the flight plan are shown. An improved flight plan generated after the start of the mission is illustrated in frame (b). Frame (c) shows the flight plan and the actual route of the UAV, after it has flown to both targets 2 and 3. The route that is off the flight plan was caused by a dropout of command data packet sent to the UAV, which triggered a return to the home location. The vehicle, however, returned to the commanded flight path after the dropout. The mission was concluded once the vehicle reached target 3.

Conclusion

This paper presents the Boeing Multi-Vehicle UAV Testbed and Evolution-based Co-operative Planning System developed by the University of Washington. We demonstrate that the testbed provides sufficient tools for developing effective real-time planning and cooperation algorithms for autonomous team decision making. We also demonstrate how the planning system is capable of coordinating a team to perform its mission efficiently while adapting continuously to changes in the environment.

Acknowledgments

This work is supported by the Boeing Company in St. Louis, Missouri. The authors would like to thank Michael Abraham and the Boeing Phantom Works research team for supporting this work.

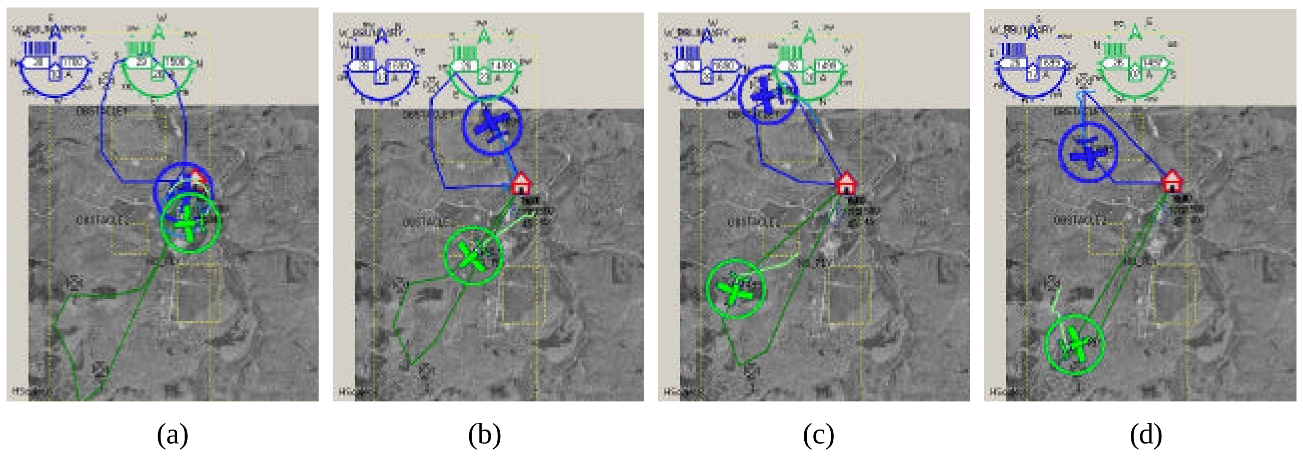

Figure 11. Simulation results of a mission with two UAVs and three targets

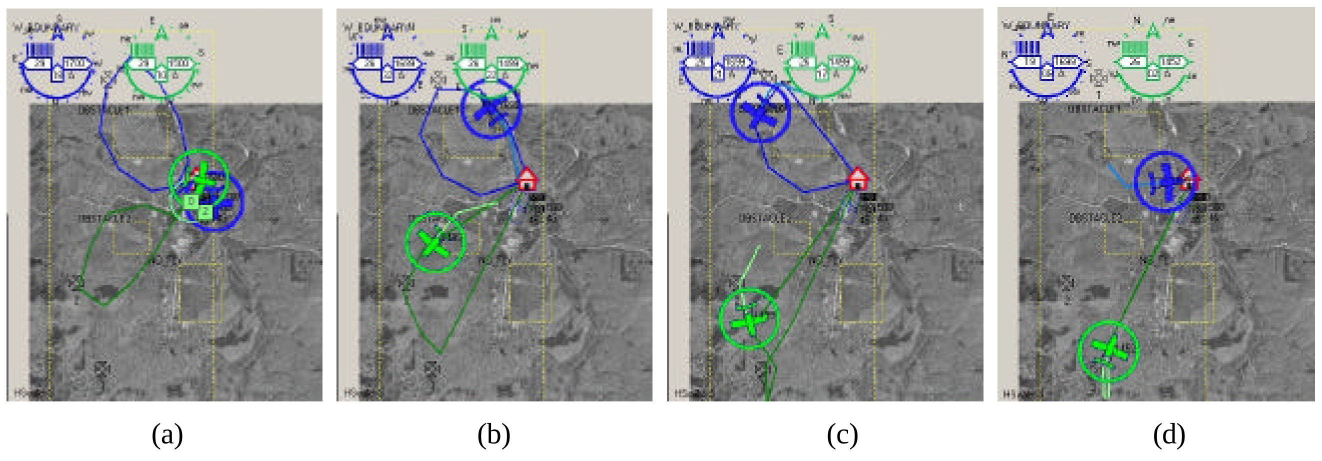

Figure 12. Simulation results of a mission with a popped-up target

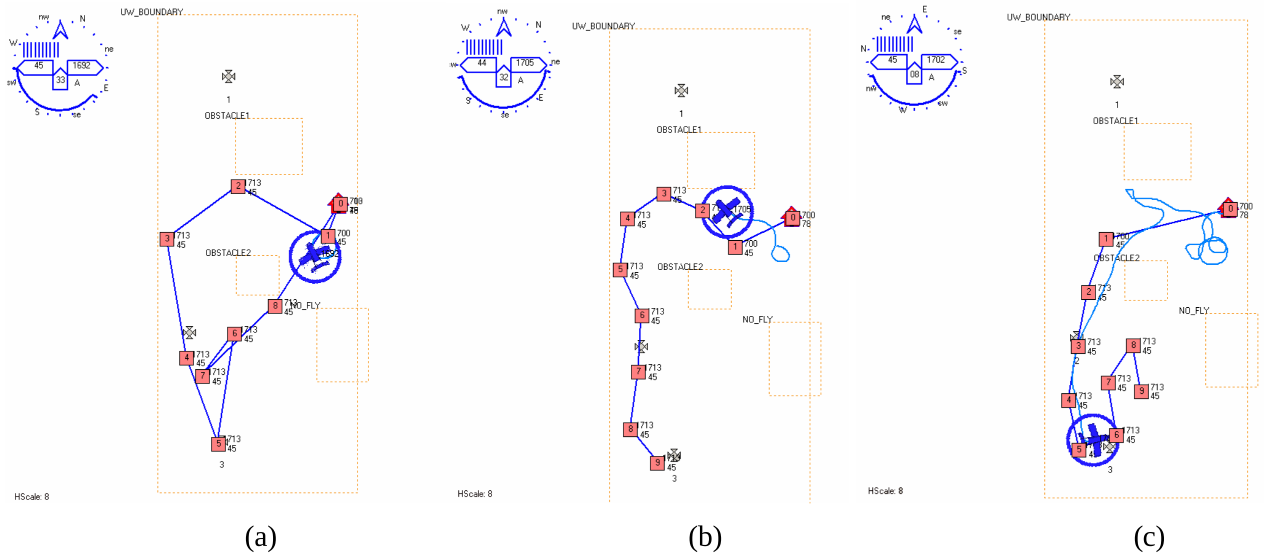

Figure 13. Flight test results of a single vehicle mission

References

[1] Pongpunwattana, A., 2004, "Real-Time Planning for Teams of Autonomous Vehicles in Dynamic Uncertain Environments," Ph.D. Dissertation, University of Washing, Seattle, WA.

[2] Ho, Y. C. and Chu, K. C., 1972, "Team Decision Theory and information Structures in Optimal Control Problems Part I," *IEEE Transactions on Automatic Control*, vol. 17, no. 1, pp. 15-22.

[3] Fogel, D. B., 2000, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd ed., Piscataway, NJ, IEEE Press.

[4] Capozzi, B. J., 2001, "Evolution-based Path Planning and Management for Autonomous Vehicles," Ph.D. Dissertation, University of Washington, Seattle, WA.