Introduction to the MultiUAV2 Simulation and Its Application to Cooperative Control Research

S. J. Rasmussen *, J. W. Mitchell †, P. R. Chandler ‡, C. J. Schumacher §, A. L. Smith ¶
AFRL/VACA, Wright-Patterson AFB, OH 45433
* Steven Rasmussen is a General Dynamics on-site Contractor.
steven.rasmussen@wpafb.af.mil
† Jason Mitchell is an Aerospace Scientist for Emergent Space Technologies, Greenbelt, MD.
Jason.Mitchell@emergentspace.com
‡ Phillip Chandler is a Senior Controls Engineer.
philip.chandler@wpafb.af.mil
§ Corey Schumacher is a Senior Aerospace Engineer.
corey.schumacher@wpafb.af.mil
¶ Austin Smith was a Student Trainee (Aerospace Engineer) from Purdue University.

Abstract— **This paper describes the MultiUAV2 simulation and how it has been applied to cooperative control of autonomous uninhabited air vehicles (UAV). MultiUAV2 is a simulation based on SIMULINK, MATLAB, and C++ that is capable of simulating multiple UAVs which cooperate to accomplish tactical missions. First there is a discussion of cooperative control of UAVs and then the background of the MultiUAV2 simulation. Next, the simulated mission is explained, including how users can introduce new missions. Next, there are descriptions of the major elements of MultiUAV2, which are: targets/threats, vehicle dynamics, sensors, communications and cooperative assignment algorithms. In the final section, an example of the simulation event flow is presented.**

I. OVERVIEW

Advances in technology have made it possible to field autonomous uninhabited air vehicles (UAVs) that can be deployed in teams to accomplish important missions such as suppression of enemy air defenses; and combat intelligence, surveillance, and reconnaissance. While it is technically possible to field these UAVs, work is needed to develop cooperative control strategies/algorithms to enable them to perform these types of missions. To facilitate cooperative control algorithm development the MultiUAV2 simulation was developed.

MultiUAV2 is a simulation capable of simulating multiple UAVs which cooperate to accomplish tactical missions. It is based on the MATHWORKS' SIMULINK and MATLAB programming environments, see [1], and on compiled C++ functions. MultiUAV2 is organized using SIMULINK blocks, but most of the functionality is written in MATLAB script functions. MultiUAV2 is a non-real-time simulation that contains cooperative control algorithms, six-degree-of-freedom (6DOF) vehicle dynamics models, simple target models, and inter-vehicle communication models. The nominal simulated mission is wide area search and destroy in which UAVs search a predefined area and cooperate to prosecute any targets that are discovered.

MultiUAV2 has been used for many different projects such as developing and testing cooperative control algorithms, implementing time-optimal trajectories, and examining effects of communication difficulties on cooperative control algorithms. It is available to the public and has been provided to many researchers from government, academia, and industry. The latest released version of MultiUAV2 can be downloaded from *http://www.isrparc.org/*, select the link: "*USAF Simulation and Research Program*".

This paper starts with background on cooperative control of UAVs and moves to a background of the MultiUAV2 simulation. After that, there is a discussion of how missions are implemented in MultiUAV2. Next the major elements of MultiUAV2 are explained. These include target/threats, vehicle dynamics, sensors, communications, and cooperative control algorithms. The paper concludes with a sample of the flow of events during the simulation.

II. COOPERATIVE CONTROL BACKGROUND

Major portions of proposed UAV missions can be preplanned, but due to limited information about enemy positions and assets in the battlefield area, the UAVs will have to react to changes in perceived enemy state

during execution of the mission plan. By cooperating with each other, the UAV team will be able to optimize the use of their combined resources to accomplish the goals of their mission. If the UAVs are unable to cooperate with each other in online planning and execution of the mission, then either group autonomy will be traded for high levels of manned intervention or more vehicles/resources will be required to perform the mission. While cooperation of this kind is desirable, it can be very complicated to implement. To perform these missions, acceptable algorithms must be solved with given time constraints and be robust to uncertainties arising from sources such as sensors, communications, and plan execution. Multi*UAV*2 has been used to develop and test many candidate cooperative control algorithms. These algorithms are discussed in §V-E.

III. Multi*UAV*2 Background

Construction of Multi*UAV*2 satisfies the need for a simulation environment that researchers can use to develop, implement and analyze cooperative control algorithms. Since the purpose of Multi*UAV*2 is to make cooperative control research accessible to researchers it was constructed primarily using MATLAB and SIMULINK. Some of the simulation functions are programmed in C++. C++ is used for functions that require faster execution and do not need to be modified by researchers. For example, the cooperative control algorithms are coded in MATLAB scripts and the vehicle dynamics are coded in C++.

The simulation is implemented in a hierarchical manner with inter-vehicle communications explicitly modelled. During construction of Multi*UAV*2, issues concerning memory usage and functional encapsulation were addressed. Multi*UAV*2 includes plotting tools and links to an external program for post-simulation analysis, see [2]. Each of the vehicle simulations include 6DOF dynamics and embedded flight software. The embedded flight software consists of a collection of managers that control situational awareness and responses of the vehicles. Managers included in the simulation are: Tactical Maneuvering, Sensor, Target, Cooperation, Route, and Weapons.

During the simulation, vehicles fly predefined search trajectories until a target is encountered. Each vehicle has a sensor footprint that defines its field of view. Target positions are either set randomly or they can be specified by the user. When a target position is inside of a vehicle's sensor footprint, that vehicle runs a sensor simulation and sends the results to the other vehicles. With actions assigned by the selected cooperative control algorithm, the vehicles prosecute the known targets. For the nominal simulation, the vehicles are destroyed when they attack a target. During prosecution of the targets the vehicles generate their own minimum-time trajectories to accomplish tasks. The simulation takes place in a three dimensional environment, but all of the trajectory planning is for a constant altitude, i.e. two dimensions. Once each vehicle has finished its assigned tasks it returns to its predefined search pattern trajectory. The simulation continues until it is stopped or the preset simulation run time has elapsed.

IV. Simulated Mission

In Multi*UAV*2 the mission is made up of the participants, e.g. UAVs and targets, the mission objectives, e.g. find and destroy the targets, and the tactics, i.e. how the UAVs meet the mission objectives. The nominal mission [1] in Multi*UAV*2 is a wide area search and destroy (WASD) mission. Multi*UAV*2's WASD mission objective is to find an destroy all of the targets. The UAVs search a predefined area for targets. If a suspected target is found the UAVs must classify it to make sure that it is a target. Once a target is classified the vehicles attack it and then verify that it has been killed.

The WASD mission is implemented in Multi*UAV*2 by giving the UAVs the capability to perform tasks, based on the perceived status of the target, such as, search, classify, attack, and verify destroy. To do this the UAVs must keep track of the state of the targets and cooperate to determine which UAV performs which task. Target state functions, see Figure 1, are implemented on each vehicle, for each target, to make it possible for the vehicle to keep track of the state of all of the targets. Based on the target's current state, the vehicle determines which task is necessary to transition the target to the next state. Algorithms to compute trajectories are defined for each task type. UAVs generate trajectories for all of the current tasks and communicate the information required for cooperation to the other UAVs. The UAVs run their cooperation algorithms and implement their assignments.

In order to modify existing or add new missions one can change the participants, mission objective, or tactics. New targets §V-A, and vehicles, see §V-B, can be added to the simulation or existing ones can be modified. The

[1] The "nominal" mission is the default mission supplied with the released version of Multi*UAV*2.

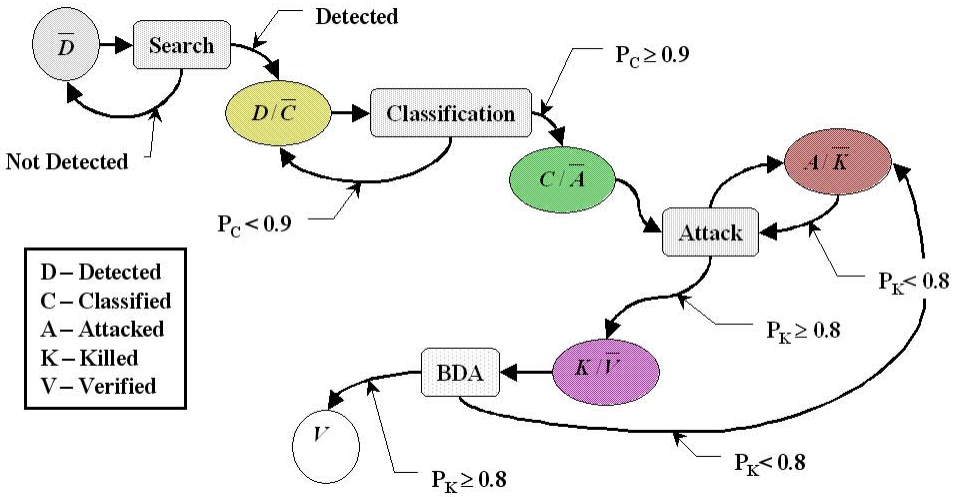

Fig. 1. Target state transition diagram.

mission objectives and tactics are encoded in the target states and the task algorithms. The target states can be modified the cause the UAVs to perform different tasks or tasks in a different order and thus perform different missions. New task algorithms can be added or existing algorithms can be modified to cause the vehicles to perform tasks in a particular manner.

V. Organization of MultiUAV2

MultiUAV2 is organized into two major top-level blocks, Vehicles and Targets, see Figure 2. The other two blocks at the top level, Initialization and DataForPlotting, call functions to initialize the simulation and save simulation data for plotting. Nominally, the top-level blocks contain the sub-blocks and connections required to implement simulation of vehicles and targets, see Figures 3–6. Information flow between the vehicles is facilitated with a *communication simulation*, see §V-D. Information flow between blocks within each vehicle is implemented using Simulink *wires* and, sparingly, global Matlab memory. To facilitate simulation truth data flow between the objects in the simulation a truth message passing mechanism is used, see §V-D.

Nominally there are 8 vehicles and 10 targets implemented in MultiUAV2. By changing global parameters, the number of targets and vehicles used in a simulation can be decreased. The number of vehicles and targets can be increased by adding more blocks and making changes to global parameters. By default, all of the vehicles in this simulation are homogeneous and share

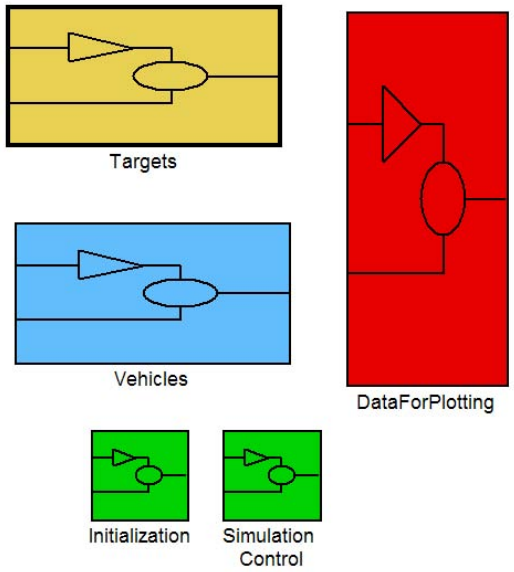

Fig. 2. MultiUAV2 Top level blocks.

the same simulation structure. The same is true for the targets. Therefore, to implement the simulation only one vehicle block and one target block needs to be built and then copies of these blocks can be used to represent the rest of the vehicles and targets. To simplify simulation model modifications, a vehicle and a target block were implemented and then saved in a Simulink block library. This block library was used to create additional instances of the Vehicle and Target blocks. When one uses a block from a block library, a link from

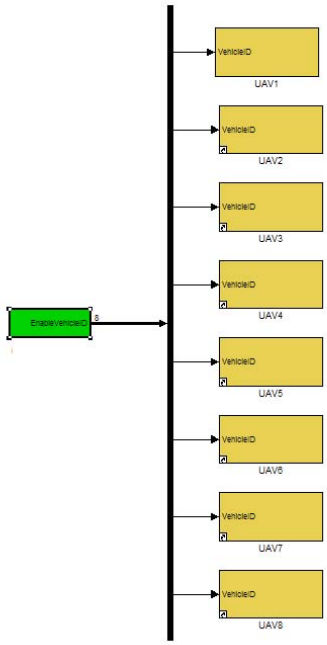

Fig. 3. MultiUAV2 Vehicle blocks.

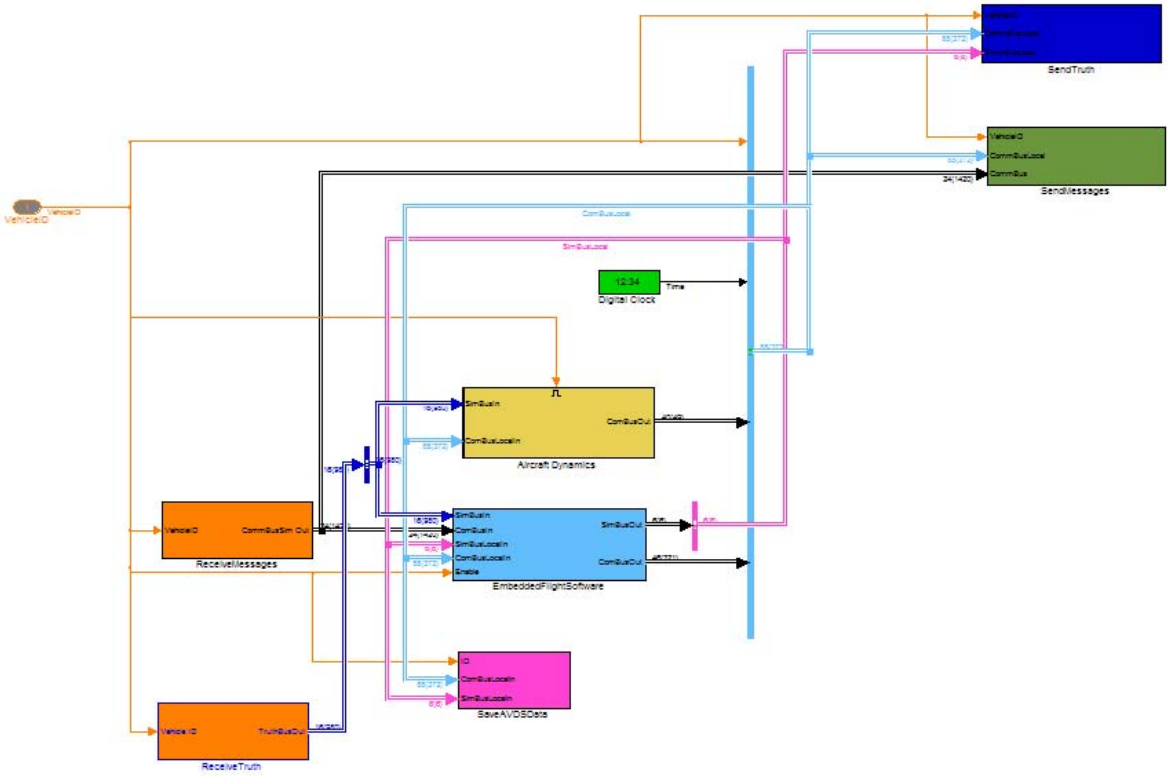

Fig. 4. MultiUAV2 Blocks that make up a vehicle.

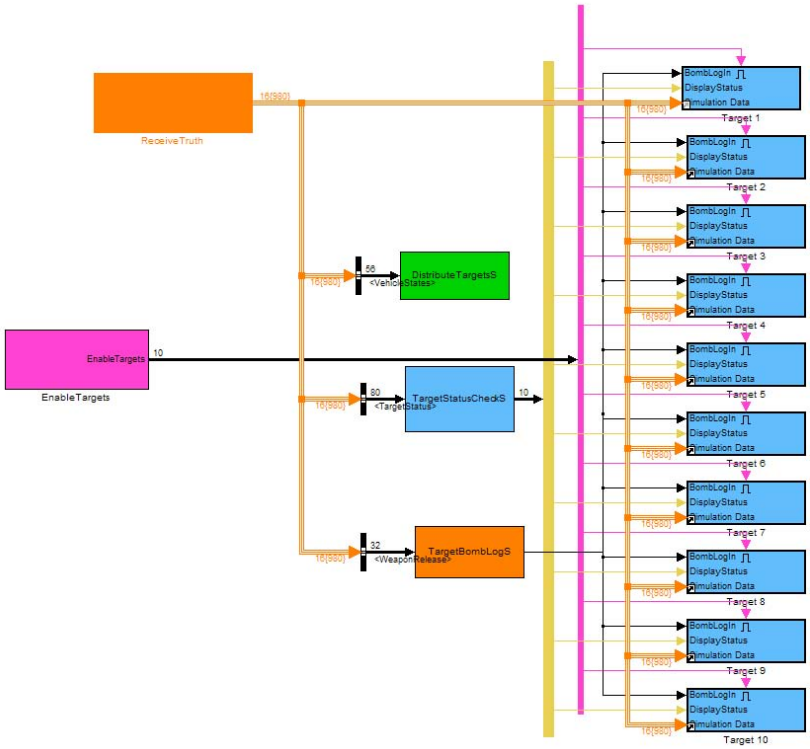

Fig. 5. MultiUAV2 Target blocks.

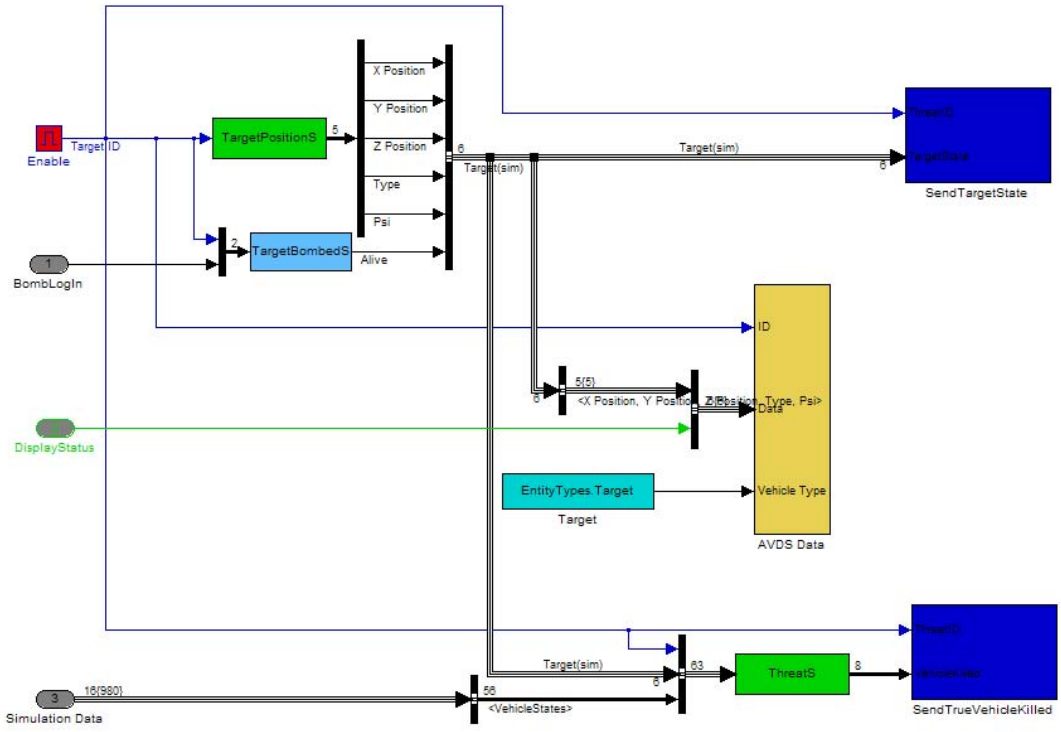

Fig. 6. MultiUAV2 Blocks that make up a target.

the block to the library is created so when the library is updated the linked blocks are also updated. Therefore the first vehicle or target block is the *real* block and the rest of the blocks are links to a copy of the *real* blocks in the block library.

Each vehicle model contains the Embedded Flight Software (EFS) blocks necessary to implement cooperative control of the vehicles. EFS is a collection of software managers that cause the vehicle to perform the desired tasks, see Figure 7. The following managers have been implemented:

Tactical Maneuvering Manager
: This manager is used to perform all of the functions necessary for near-term guidance of the vehicle. At this time the Tactical Maneuvering Manager is only being used to generate autopilot commands to cause the vehicle to follow given waypoints.

Sensor Manager
: This manager is used to perform all of the functions necessary to monitor the sensors and process sensed data. It also contains the sensor process part of the sensor simulation, see V-C.2.

Target Manager
: This manager creates and manages list of known and potential targets. Also manages the target state functions for the vehicle.

Cooperation Manager
: Calculates assignments for the vehicle based on information gathered from all of the vehicles.

Route Manager
: This manager is used to plan and select the route for the vehicle to fly. Part of the functionality is calculation of the lowest cost route to all known targets, based on each target's state. The Route Manager also monitors the status of the vehicle's assigned task.

Weapons Manager
: Weapons Manager Selects a weapon and then simulates its deployment.

While it is important to understand the operation of the EFS managers to be able to add new cooperative control algorithms to MultiUAV2 it is also important to understand the other major elements of MultiUAV2. To that end the following sections describe the major elements of MultiUAV2, i.e. targets/threats, vehicle dynamics, sensors, communications, and cooperative assignment algorithms.

A. Target/Threats

Since the targets in MultiUAV2 can also be configured to destroy UAVs, they can also act as threats. The nominal targets are fairly simple, i.e. they can calculate where they are, calculate damage from explosions, and calculate if they have shot down a UAV, see Figure 5 and 6. Nominally the targets are stationary, but they can be given trajectories to follow by modifying a MATLAB script file that sets the position of the target.

B. Vehicle Dynamics

Vehicle dynamics in MultiUAV2 are generated using a simulation called *Variable Configuration Vehicle Simulation* (VCVS). VCVS is a nonlinear six-degree-of-freedom vehicle simulation that includes a control system which reconfigures the simulation for new aerodynamic and physical vehicle descriptions. VCVS is written entirely in C++ and can run more that 20 times faster than real time. Vehicle dynamics are based on two configuration files, one containing aerodynamic data and the other physical and control system parameters. The aerodynamic configuration file contains tables of non-dimensional forces, moments, and damping derivatives. The vehicle model calculates aerodynamic forces and moments by using the vehicle's state and control deflections as independent variables to look-up values from the aerodynamic tables. During the look-up process, linear interpolation is used for states and deflections not found in the tables. The non-dimensional values obtain from the tables are combined with vehicle state data to calculate forces and moments acting on the center of gravity of the vehicle. These forces and moments are combined with external forces and moments, i.e forces and moments from an engine. The forces and moments are used, along with the physical parameters, to calculate the equations of motion. Included in the model are first-order actuator dynamics, including rate and position limits, and first-order engine dynamics.

VCVS uses a dynamic inversion control system with control allocation as its inner loop control system, see Figure 8. A rate control system was wrapped around the inner loop to move from angular acceleration commands to angular rate commands. The outer most loop is an altitude, heading, sideslip, and velocity command control system. Gains for the rate controllers and the outer-loop controllers can be adjusted by changing parameters in

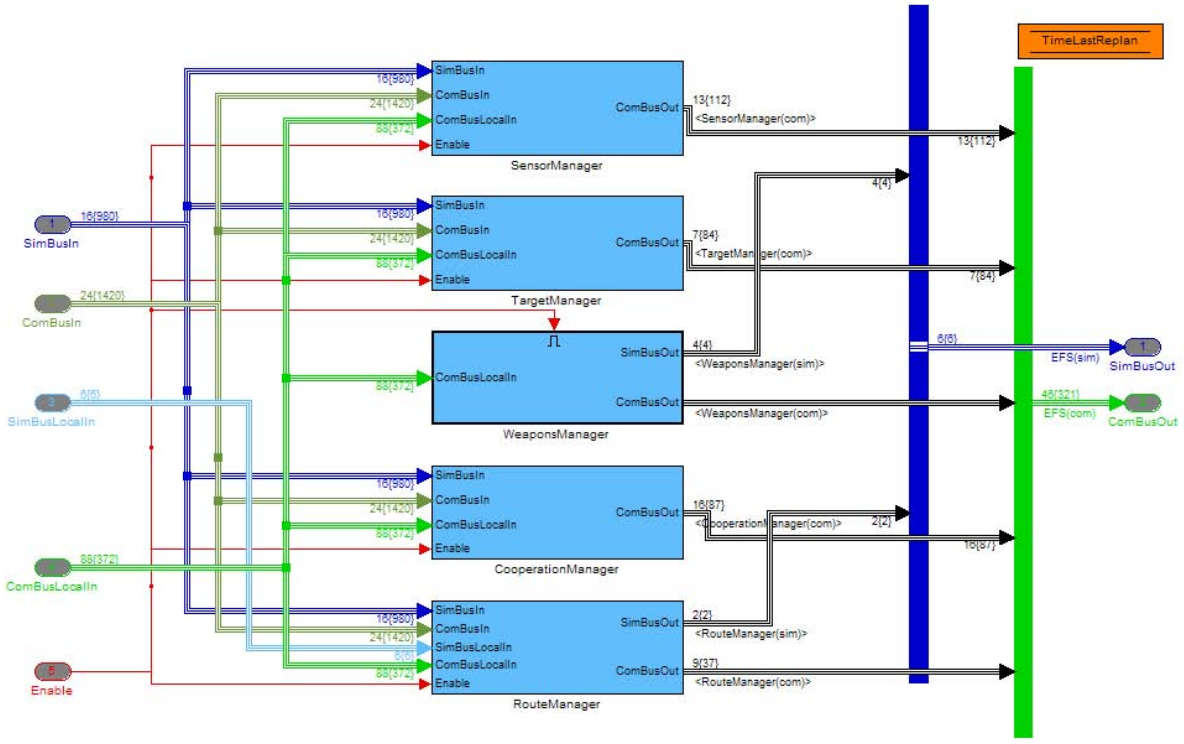

Fig. 7. MultiUAV2 Managers.

the parameter input file. New vehicle dynamics can be introduced by changing the physical and aerodynamic parameters. When new parameters are introduced, the control system uses control allocation to reconfigure for different numbers of control effectors and the dynamic inversion controller compensates for changes in response to control inputs.

Interfaces between VCVS and MATLAB/SIMULINK were constructed to aid in vehicle dynamics development and testing. This made it possible to run VCVS separately from MultiUAV2. VCVS has been used outside of MultiUAV2 to develop algorithms such as formation control, rejoin trajectory control, and surveillance trajectory control. Vehicle models ranging from the size of micro-UAVs to the size of the Global Hawk have been implemented in VCVS.

C. Sensors

MultiUAV2 separates the sensor simulation into two parts: sensor footprint and sensor process. The sensor footprint is used to determine if a target is visible to the vehicle and the sensor process is used to produce simulated output from the sensor. This makes it possible to implement the sensor footprint in C++ with the vehicle model as discussed in § V-C.1. The sensor process, in the released version of MultiUAV2, simulates an automatic target recognition (ATR) sensor, discussed in §V-C.2.

1) Sensor Footprint: In order to make sure that the sensor footprint reports every target that it passes over, it must be updated at the same rate as the vehicle dynamics model. Since the vehicle dynamics are updated at a faster rate than the rest of MultiUAV2 it was necessary to implement the sensor footprint in a C++ class, that is executed with the vehicle dynamics. This class contains functions that can calculate rectangular or circular sensor footprints. To check if targets are in the circular footprint the function compares the difference between each of the target positions and the calculated center of the footprint. If this distance is less than the sensor radius then the target is in the footprint. The rectangular footprint function transforms the coordinates of the targets into a coordinate system aligned with the rectangular footprint. After transformation the coordinates of each target are compared to the limits of the sensor and any targets inside the limits are reported.

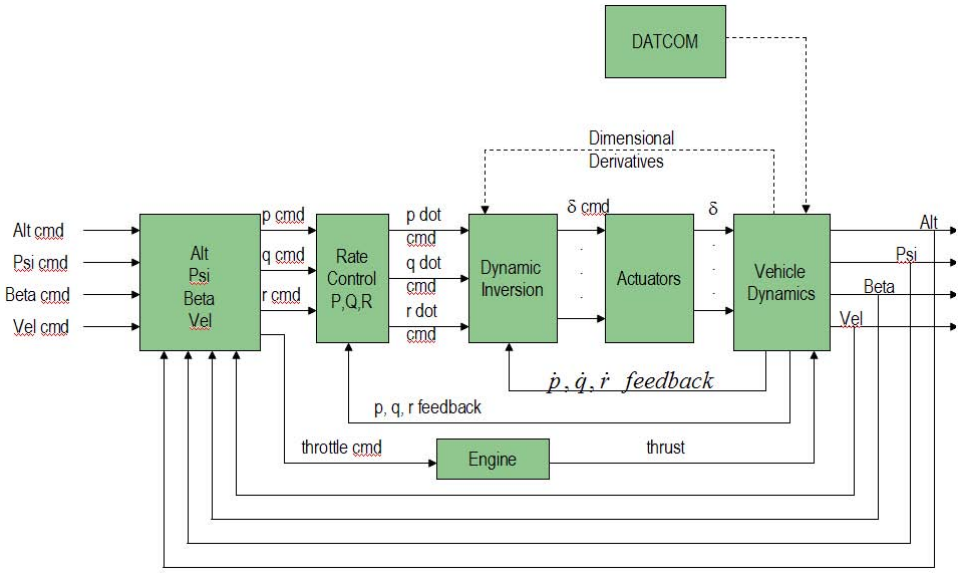

Fig. 8. VCVS Schematic.

2) Automatic Target Recognition Sensor: The major attribute of an ATR sensor simulated in the MultiUAV2 simulation is heading dependence. That is, the quality of target identification of many ATR sensors depends on the viewing direction of the target. In order to simulate the functionality of an ATR simulation heading dependent equations were developed. Given the targets length (L), width (W) and aspect angle[2] (θ) the single ATR value (ATR_s) is calculated using Equations 1a and 1b.

$$\text{ATR}_s = \begin{cases} \dfrac{W\arccos(\theta) + L\arcsin(\theta)}{L+W} \times \text{SF} & \text{for } 0 \le \theta \le \dfrac{\pi}{2} \\ \dfrac{-W\arccos(\theta) + L\arcsin(\theta)}{L+W} \times \text{SF} & \text{for } \dfrac{\pi}{2} < \theta \le \pi \\ \dfrac{-W\arccos(\theta) - L\arcsin(\theta)}{L+W} \times \text{SF} & \text{for } \pi < \theta \le \dfrac{3\pi}{2} \\ \dfrac{W\arccos(\theta) - L\arcsin(\theta)}{L+W} \times \text{SF} & \text{for } \dfrac{3\pi}{2} < \theta < 2\pi \end{cases} \tag{1a}$$

[2]Angle definitions are shown in Figure 9.

$$\text{SF} = 0.8\frac{L+W}{\sqrt{W^2+L^2}} \tag{1b}$$

A representative plot of ATR_s vs. θ is shown in Figure 10. Note that the maximum ATR value is 0.8. In the nominal mission an ATR value greater than or equal to 0.9 is required before a target can be attacked. In order to increase the ATR value, the value from a single sighting of the target can be combined with another sighting of the target. Given the values for ATR_s and the respective angles θ, two single ATR values for a target can be combined into to one (ATR_c) with the following equations:

$$\text{ATR}_c = (\text{ATR}_1 + \rho \times \text{ATR}_2) - (\text{ATR}_1 \times \rho \times \text{ATR}_2) \tag{2a}$$

$$\rho = 1.0 - e^{-0.3|\theta_2 - \theta_1|} \tag{2b}$$

If more than two single ATR values exist for a target, combined ATR values are calculated for all pairwise combinations of the single values. The largest value from the set of combined values is used for that target.

D. Inter-Vehicle/Simulation Truth Communications

The MultiUAV2 simulation has two mechanisms for passing messages between objects in the simulation, one for communication messages and one for simulation truth messages. Previous releases of the MultiUAV2 simulation provided vehicle-to-vehicle communication via a

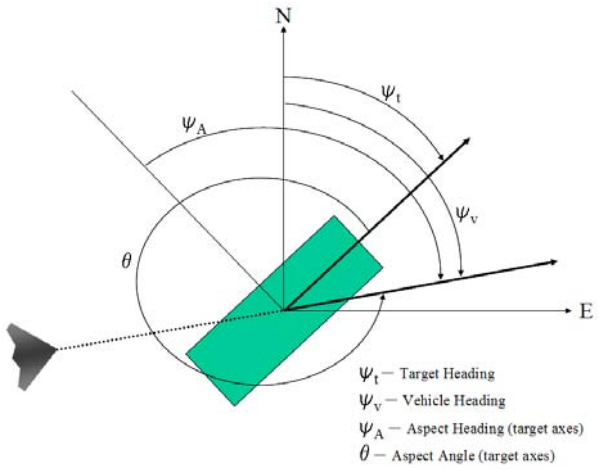

Fig. 9. Angle definitions for ATR.

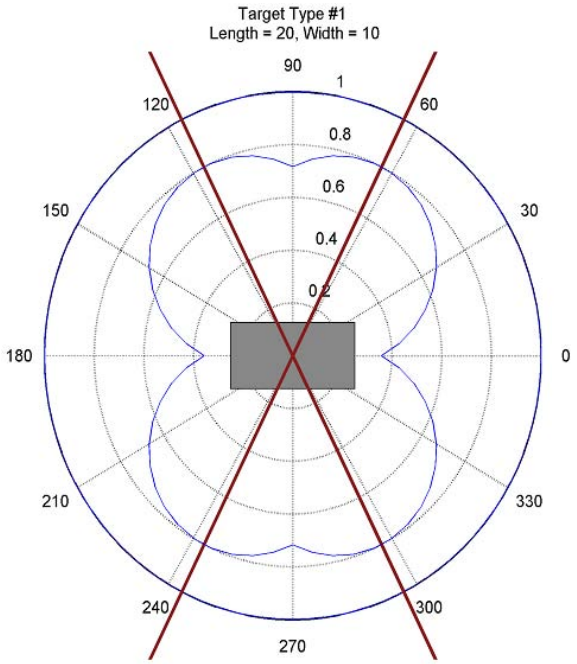

Fig. 10. ATR template.

signal bus denoted by `CommBus`, while a second aggregated signal bus, labeled `SimBus`, contained the *truth* information for the simulation. The combination of these two data buses represented the complete information state of the simulation. This *perfect* information state was available to all vehicles at every simulation timestep. From many perspectives, perfect information access is unacceptable, particularly when considering communication and processing delays. Thus, to incorporate communication and processing delays into MultiUAV2, a new communication framework was introduced, see

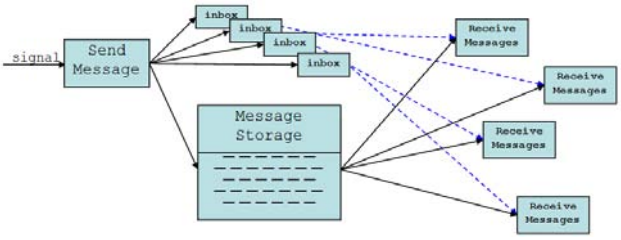

Fig. 11. Schematic of the communications framework.

Figure 11. In order to make it possible to distribute the MultiUAV2 over many computers, a similar framework was introduced for passing truth information between objects in the simulation.

Maximum design flexibility is a significant and yet vague requirement that must be met by any potential communication design. By maintaining genericity, we ensure that the resulting solution will accommodate the simulation of specific communication requirements, e.g. protocol-specific, theater-specific, or hardware-specific, while providing a simple and general framework to quantify vehicle-to-vehicle communication needs, e.g. peak or average data-rate.

To provide flexibility in implementation of communication simulations that contain varying levels of detail, a generic message passing scheme was chosen as the Virtual Communication Representation (VCR). In this design, specific message types and their format are defined centrally in the VCR and made globally available to the various Embedded Flight Software Managers (EFSMs) as context requires[3]. Minimally, a message definition must contain a unique message identifier, timestamp(s), message layout enumeration, and data field to be written by the EFSM context. Particular messages may be selected by the EFSM context as output resulting from a computation that must be remotely communicated. Outgoing messages, which include data, from each vehicle are stored centrally, and pointers to these messages are distributed to an individual input queue for each vehicle. These pointers are composed of the original message header and should minimally inform the receiver of the message type, time sent, quality or priority of the message, and which central repository contains the associated message data. A user defined rule component controls the distribution of incoming messages to all vehicles based on the message headers.

[3]The message structure discussed here refers to the format dictated by the MultiUAV2 package, rather than to messages related to a specific communication system model.

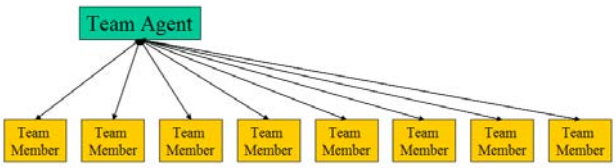

Fig. 12. UAV virtual team structure.

We avoid adhering to a specific communication model in MultiUAV2 by isolating the message delivery rules in user controlled components. Thus, end-users are free to choose any preferred communication model. Moreover, the genericity of the VCR specification provides for easy extension. For more information on the communications design see [3, 4, 5, 6]

E. Cooperative Assignment Algorithms

MultiUAV2 was constructed in parallel with the development of the following cooperative assignment algorithms. As the algorithms were developed they were implemented and tested using MultiUAV2. When deficiencies were found in MultiUAV2 new capabilities were added.

1) Redundant Central Optimization: Many of the cooperative control algorithms are implemented in a *Redundant Central Optimization* (RCO) manner to control the cooperation of vehicles while they carry out their mission to find, classify, kill, and verify the targets in the simulation. RCO consists of vehicles that are formed into a team that contains team members and a team agent as shown in Figure 12. The team agent makes and coordinates team member assignments through the use of a centralized optimal assignment selection algorithm that is based on partial information. The redundant portion of the RCO structure comes about because each team member implements a local copy of the team agent. Because of this, each of the team members calculates assignments for the entire team and then implements the assignment for itself.

2) Single Assignment Tour vs Multiple Tour Assignment: A *single task tour* assignment algorithm is an algorithm that assigns each UAV to a target to accomplish one task, i.e. classification, attack or verification. In order to make single assignments both trajectory planning and assignment algorithms must be considered. While trajectory planning for single assignment tours is not trivial, it is possible to use computational geometry to generate optimal trajectories. During the assignment process, trajectories are generated from all of the UAVs to all of the known targets based on the tasks that need to be accomplished on those targets. For task assignment, a capacitated transshipment algorithm can be used to assign the UAVs to the targets, based on the cost of traversing the candidate paths.

Assigning UAVs based on a single tour can be very inefficient, as it doesn't take into account coupling that occurs between performing tasks on targets. That is, when a UAV plans to accomplish a task on a particular target, such as classification, it is much more efficient if that UAV also can take into account the next required task for that target, such as attack. When more than one task is taken into account during the planning and assignment process, the algorithm can be said to be based on *multiple task tours*.

The need to included multiple-tours in path-planning and assignment algorithms increases the complexity of these algorithms significantly. This complexity is not only due the to possible combinatorial explosion of possible trajectories and assignments, but also due to the requirement that the tasks for each target must be accomplished in a specified order. The following sections describe different algorithms that have been implemented for both single and multiple task tour assignments. For more information see [7] and the references cited at the end of each section.

3) Capacitated Transhipment Network (Network Flow) (Single Task Tours): A network optimization model is used to calculate the vehicle task assignments. Network optimization models are typically described in terms of supplies and demands for a commodity, nodes which model transfer points, and arcs that interconnect the nodes and along which flow can take place. There are typically many feasible choices for flow along arcs, and costs or values associated with the flows. Arcs can have capacities that limit the flow along them. An optimal solution is the globally minimum cost (or maximum value) set of flows for which supplies find their way through the network to meet the demands. To model weapon system allocation, we treat the individual vehicles as discrete supplies of single units, tasks being carried out as flows on arcs through the network, and ultimate disposition of the vehicles as demands. Thus, the flows are 0 or 1. We assume that each vehicle operates independently, and makes decisions when new information is received. These decisions are determined by the solution of the network optimization model. For more information on the Network Flow algorithm see [8, 9].

4) Iterative Network Flow (Multiple Task Tours): Due to the integrality property, it is not normally possible to simultaneously assign multiple vehicles to a single target, or multiple targets to a single vehicle. However, using the network assignment iteratively, *tours* of multiple assignments can be determined. This is done by solving the initial assignment problem once, and only finalizing the assignment with the shortest estimated time of arrival. The assignment problem can then be updated assuming that assignment is performed, updating target and vehicle states, and running the assignment again. This iteration can be repeated until all of the vehicles have been assigned terminal attack tasks, or until all of the target assignments have been fully distributed. The target assignments are complete when classification, attack, and battle damage assessment tasks have been assigned for all known targets. Assignments must be recomputed if a new target is found or a UAV fails to complete an assigned task. For more information on the Iterative Network Flow algorithm see [10].

5) Iterative Auction (Multiple Task Tours): Using the same strategy as the Iterative Network Flow, the Iterative Auction builds up multiple task tours of assignments for the vehicles by using a Jacobi auction solver. The auction is used to find an initial set of assignments, freezes the assignment with the shortest estimated time of arrival and then repeats this process until all possible tasks have been assigned. For more information see the explanation of the Iterative Network Flow algorithm in §V-E.4

6) Relative Benefits (Multiple Task Tours): This method requires a relaxation of the optimality requirement, but can potentially produce good paths and assignments quickly. One major problem with this and other resource allocation methods is the absence of a good metric to judge its efficacy. There are some possible algorithms that will return results that are very close to optimum, but none of them have been implemented for this type of problem. The central theme of this algorithm is that multiple assignment tours can be developed by making single assignment tours and then trading assignments between the UAVs based on the relative benefit of one UAV taking on the assignment of another. For more information on the Relative Benefits algorithm see [7].

7) Distributed Iterative Network Flow (Multiple Task Tours): The Iterative Network Flow algorithm was initially implemented in a RCO manner, see V-E.1. For the Distributed Iterative Network Flow, the original Iterative Network Flow algorithms were implemented in a distributed manner, i.e. each vehicle calculates benefits for its self to complete the required tasks at each iteration and then sends these benefits to the other vehicles. All the vehicle run the Network Flow algorithms and then move on to the next iteration.

8) Distributed Iterative Auction (Multiple Task Tours): The Iterative Auction algorithm was initially implemented in a RCO manner, see V-E.1. For the Distributed Iterative Auction, the original Iterative Auction algorithms were implemented in a distributed manner, i.e. each vehicle calculates bids for the required tasks at each iteration and sends these bids to the other vehicles for use in an asynchronous distributed auction.

VI. Simulation Event Flow Example

During the progress of the simulation the EFS managers cause the vehicle to react to changes in target states, vehicle tasks, and task assignments. As an example of the information flow between EFS managers during the simulation, when the CapTransShip algorithm is selected the following is a sequence of events that occur when a previously undetected target is discovered:

1) Vehicle Dynamics senses target.
2) Based on the vehicle's heading and the target ID, the vehicle's SensorManager calculates a single ATRvalue.
   - The vehicle sensing the target sends an *ATRSingle* message to all of the vehicles. This message contains a time stamp and the target's single ATR value, sensed heading, estimated pose angle, and estimated type.
3) Each vehicle's SensorManager calculates combined ATR values for the target. These values are based on all of the information that the vehicle has obtained about the target.
4) TargetManagers on all vehicles update the target state based on the combined ATR calculated locally.
   - For each vehicle, if any of the targets change state, the vehicle sends a *TargetStatus* message. This message contains a time stamp and the status of the targets.
5) Based on the change in target status, the RouteManagers, on all of the vehicles, calculate the optimal route, estimated time of arrival (ETA), and cost to perform the required tasks on all of the known targets.

- Each vehicle sends the *ETACostToSearch* message to all of the other vehicles. This message contains the vehicle's ETA , cost of performing the tasks, and the return to search distance for each of the known targets.

6) CooperationManagers on all vehicles calculate optimal assignment for all of the vehicles to perform the next required task on each of the targets based on the supplied costs.
7) RouteManagers on all vehicles implement the route assigned for that vehicle.
8) Tactical Maneuvering Managers functions on all vehicles read assigned waypoints and calculate commands that will cause the autopilot to cause the vehicle to fly to the waypoints.
9) VehicleDynamics read the autopilot commands and run the vehicle dynamics simulation. This maneuvers the vehicle along the trajectory to perform the assigned task.
10) If the assigned task is attack, the vehicle's WeaponsManager determines where the explosion occurred and sends the *WeaponsRelease* truth message. This message contain a time stamp, weapon ID, weapon type, and weapon aim point.
11) Using the weapon type and aim point, each target calculates damage to it due to the explosion.
12) New replanning cycles are triggered each time a target changes state or if a vehicle fails its assigned task.

VII. Conclusion

The MultiUAV2 simulation has been constructed to be capable of simulating multiple UAVs performing cooperative control missions, while ensuring accessability to researchers. It is a very flexible simulation that can be modified to perform missions requiring modifications to the existing vehicles, targets, and tactics. The vehicle dynamics simulation, VCVS, uses 6DOF equations of motion and nonlinear aerodynamic look-up tables to produce good fidelity dynamics and can be used separately from MultiUAV2 for other types of research. MultiUAV2has been used to perform cooperative control research on areas such as task assignment, path planning, and communication effects. MultiUAV2has been released to the public and has been provided to many government agencies, universities and companies.

References

[1] MATLAB and SIMULINK. The Mathworks Inc. [Online]. Available: http://www.mathworks.com

[2] AVDS. RasSimTech Ltd. [Online]. Available: http://www.RasSimTech.com

[3] AFRL/VACA, *MultiUAV2 Simulation User's Manual*, AFRL/VACA, Wright-Patterson AFB, OH 45433, 2005.

[4] J. W. Mitchell, S. J. Rasmussen, and A. G. Sparks, "Communication requirements in the cooperative control of wide area search munitions using iterative network flow," in *Proceedings of the Fourth International Conference on Cooperative Control and Optimization*, 2003.

[5] J. W. Mitchell, C. J. Schumacher, P. R. Chandler, and S. J. Rasmussen, "Communication delays in the cooperative control of wide area search munitions via iterative network flow," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Austin, TX, 2003.

[6] J. W. Mitchell and A. G. Sparks, "Communication issues in the cooperative control of unmanned aerial vehicles," in *Proceedings of the Forty-First Annual Allerton Conference on Communication, Control, & Computing*, 2003.

[7] S. J. Rasmussen, C. Schumacher, and P. R. Chandler, "Investigation of single vs. multiple task tour assignments for uav cooperative control," in *Proceedings of the 2002 AIAA Guidance, Navigation, and Control Conference*, Monterey, CA, 2002.

[8] K. E. Nygard, P. R. Chandler, and M. Pachter, "Dynamic network flow optimization models for air vehicle resource allocation," in *Proceedings of the American Control Conference*, Arlington, Virginia, 2001.

[9] C. J. Schumacher, P. R. Chandler, and S. J. Rasmussen, "Task allocation for wide area search munitions via network flow optimization," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Montreal, Canada, 2001.

[10] ——, "Task allocation for wide area search munitions via iterative network flow optimization," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Monterey, CA, 2002.