Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

MultiUAV: A MULTIPLE UAV SIMULATION FOR INVESTIGATION OF COOPERATIVE CONTROL

S. J. Rasmussen

Veridian
Wright-Patterson AFB, OH 45433 U.S.A.

P. R. Chandler

Flight Control Division
Air Force Research Laboratory (AFRL/VACA)
Wright-Patterson AFB, OH 45433 U.S.A.

ABSTRACT

This paper describes MultiUAV, a simulation that is capable of simulating multiple unmanned aerospace vehicles which cooperate to accomplish a predefined mission. The simulation was constructed using the Mathwork's Simulink simulation software. Construction of the simulation satisfied the need for a simulation environment that researchers can use to implement and analyze cooperative control algorithms. The simulation is implemented in a hierarchical manner with inter-vehicle communication explicitly modeled. During construction of MultiUAV, issues concerning memory usage and functional encapsulation were addressed. MultiUAV includes plotting tools and links to an external program for post-simulation analysis. Each of the vehicle simulations include six-degree-of-freedom dynamics and embedded flight software. The embedded flight software consists of a collection of managers (agents) that control situational awareness and responses of the vehicles. Managers included in the simulation are: Tactical Maneuvering, Sensor, Target, Cooperation, Route and Weapons.

1 BACKGROUND

In order to implement and evaluate cooperative control strategies for unmanned aerospace vehicles (UAVs), a simulation environment was needed. During an earlier project, Control Automation and Task Allocation (CATA), a multiple-vehicle/multi-agent simulation was developed, see Boeing (1997). The CATA simulation was constructed in C++ and was modified for use in cooperative control research. This simulation was found to be very useful in early cooperative control studies, see Chandler, Rasmussen, and Pachter (2000). Since the CATA simulation was relatively large and written in C++ it proved to be difficult for other researchers to use. This prompted the development of a Simulink-based multi-vehicle/multi-agent simulation (MultiUAV). Simulink is a symbolic programming environment which makes the simulation relatively easy for researchers to understand and use, see The Mathworks, Inc (2002). The MultiUAV simulation was constructed using the organizational structure of the CATA simulation. CATA's "vehicle" and "tactical maneuvering manager" classes were extracted directly from the CATA simulation to be used in MultiUAV.

MultiUAV is capable of simulating 8 vehicles and 10 targets. Simulated vehicles include embedded flight software (EFS) and vehicle dynamics. EFS is the software that implements cooperative control algorithms. The vehicle dynamics are simulated with six-degree-of-freedom (SDOF) equations of motion. The vehicle model includes an autopilot that makes the vehicles capable of waypoint navigation.

2 MultiUAV IMPLEMENTATION

MultiUAV is organized hierarchically with two major top-level blocks, *Vehicles* and *Targets*, see Figure 1. The other two blocks at the top level, *Initialization* and *DataForPlotting*, call functions to initialize the simulation and save simulation data for plotting. The top-level blocks contain the sub-blocks and connections required to implement simulation of the 8 vehicles and 10 targets, see Figures 2 and 3.

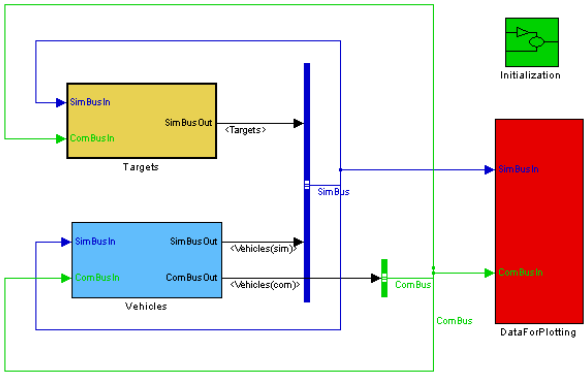


Figure 1: Top Level Blocks

There are three types of functions used in MultiUAV, script, compiled and Simulink built-in. The majority of functions in MultiUAV are written in MATLAB's script language and accessed from Simulink using S-function blocks. This makes it convenient for researchers since

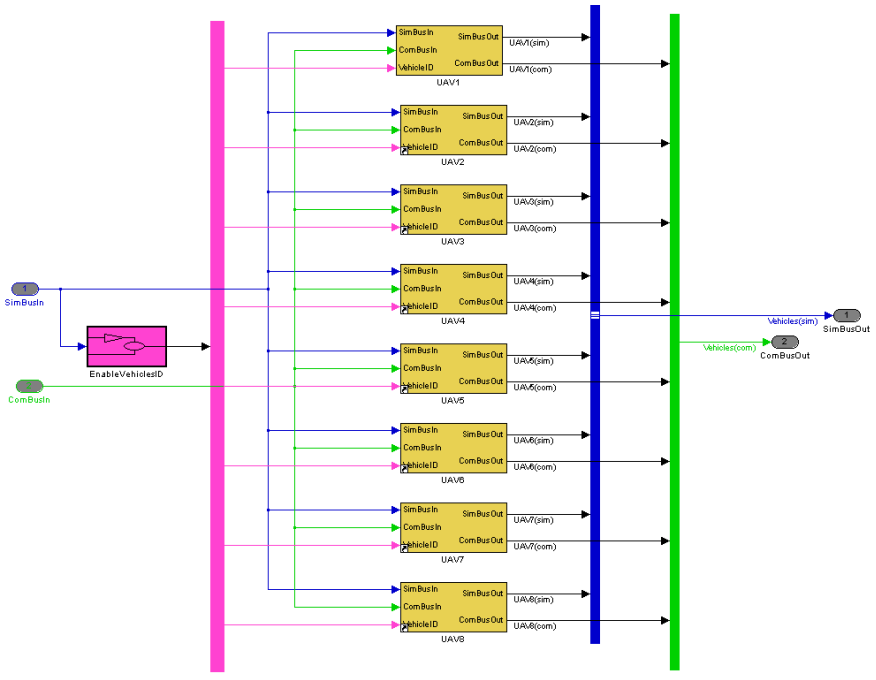


Figure 2: Vehicle Blocks

many know MATLAB script language. The compiled functions include the classes used from CATA and other functions that are available in other languages. The compiled functions are connected to Simulink through S-function blocks.

2.1 Simulation Data and Communications

In order to keep MultiUAV as accessible as possible, every effort is made to confine the transfer of data between Simulink blocks to the signal "wires" in the Simulink user interface. This is not always possible. For instance, passing variable length matrices as signals in Simulink would have caused too much overhead so the decision was made to transfer the waypoint matrices through global MATLAB memory.

To facilitate data flow between the elements of the simulation, two data busses are implemented, one for simulated communication signals (ComBus) and one for simulation signals (SimBus). The ComBus bus contains the data signals that would be communicated between real vehicles. The SimBus bus contains the data signals that are required only by the simulation, i.e. truth data. The Simulink *Bus Selector* block, makes it possible to access any of the signals from the data busses by selecting the signal label. These busses are fed back as inputs to all of the major sub-blocks. This makes it possible to access signals from one simulated vehicle deep in the sub-structure of another.

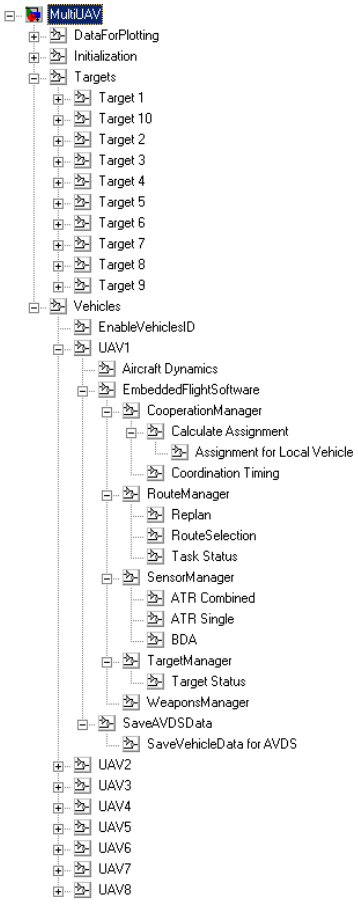


Figure 3: MultiUAV Hierarchy

2.2 Implementation

MultiUAV contains only homogeneous vehicle and 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 "Cooperation" block library was used to build the Vehicles and Targets blocks. When one uses a block from a block library, a link from 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 implemented block and the rest of the blocks are links to a copy of the implemented blocks in the Cooperation block library.

2.3 Functional Encapsulation and Data Storage

The structure of MultiUAV, makes it necessary to plan a strategy for functional encapsulation and data storage. Since the functions are shared by all of the vehicles and targets, care must be taken to encapsulate functions and data as much as possible. To do this, a few guidelines were established for function development and data storage.

Functions are not allowed to have local storage, such as persistent variables. This prevents one vehicle from changing/accessing data stored by another vehicle. This forces all of the data to be passed-in through the function call or global memory. Global memory is necessary because there are elements that can't be passed through the Simulink connections. To keep global memory encapsulated, each vehicle is allocated a space in global memory that it can use for storage. The vehicles are not allowed to access another vehicle's memory.

Since the reason for using Simulink was to make the simulation more accessible to researchers, the strategy is to use Simulink elements for data storage as much as possible. This keeps the data visible in the Simulink user interface. The other option is to use global MATLAB memory which makes the data less obvious. The different types of memory used in this simulation are outlined below.

- **Output of Blocks** – the outputs of blocks can act as memory. The value of the block outputs is held until the block is updated. If the block is disabled the output can be set to hold its last updated value.
- **"Data Store" Blocks** – These blocks can be used to store data inside of a block which is only visible within that block and inside subsystems of that block. This is a good way to save data in an object-oriented fashion.
- **Global Memory** – the use of global memory should be a "last choice" since it makes the simulation less modular and thus less flexible. For this simulation, global memory has been used for structured storage, constant variables and constant structures. Note, the term "constant" is used to imply that the value of the variable is not intended to change during the simulation. There is no mechanism in MATLAB/Simulink to enforce this. The following are examples of variables/structures implemented in global memory:
  TargetMemory – (array of structures) The "main" memory for the individual target blocks. Each structure in this array is used as memory for a target.
  VehicleMemory – (array of structures) The "main" memory for the vehicle blocks. Each structure in this array is used as memory for a vehicle. The structures contain structures for each of the managers. This gives each manager a structure for data storage.
  WaypointCells – (cell array) These cells are used to store the current waypoints for each vehicle. The vehicle s-function reads the waypoints from these cells.

3 SIMULATION ANALYSIS TOOLS

3.1 The Graphical User Interface

To assist users, a graphical user interface (GUI) was developed. The GUI contains buttons, a check block, and a pull-down menu that perform various functions:

- ***Vehicle#*** – This is the vehicle referred to by the "Enabled" check block.
- ***Enabled*** – When this block is checked the vehicle is active during the simulation. If it is not checked the vehicle is disabled during the simulation.
- ***Enable All*** – Set all of the vehicles to the enabled state. This causes all of the vehicles to be active during the simulation.
- ***Disable All*** – Set all of the vehicles to the disabled state. This clears all of the enable flags to make it more convenient to enable individual vehicles.
- ***Save AVDS Data*** – Saves the data necessary for AVDS playback from global memory to files.
- ***Plot Results*** – Plots the results of the last simulation run, see the next section.

3.2 Simulation Data Plot Window

After running the simulation the saved data can be plotted using the *PlotOutput* function. This function can be invoked directly or by pressing the "Plot Results" button on the GUI. The resulting plot, see Figure 4, shows a top-down plan-view plot of the simulation data. The plot shows vehicle positions (numbers), sensor footprints (large col-

ored rectangles), targets (small rectangles), markers (colored circles) indicating vehicle to target assignment, and the paths of the vehicles.

3.3 AVDS Data

Data can be saved for replay in AVDS, as seen in Figure 5. This give the analyst an animated 3-D perspective of the simulation results, see RasSimTech Ltd. (2002).

4 EMBEDDED FLIGHT SOFTWARE

MultiUAV simulation contains the Embedded Flight Software (EFS) blocks necessary to implement cooperative control of the vehicles. The EFS is a collection of software managers that cause the vehicle to perform desired tasks, see Figure 6. The following managers have been implemented: Tactical Maneuvering, Sensor, Target, Cooperation, Route, and Weapons. These managers are described in the following sections.

The vehicle simulations are implemented as part of a "Redundant Central Optimization" (RCO) structure to control the cooperation of vehicles while they carry out their mission to find, classify, kill, and verify the targets in the simulation. The RCO structure consists of vehicles that are formed into a team that contains team members and a team agent. 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.

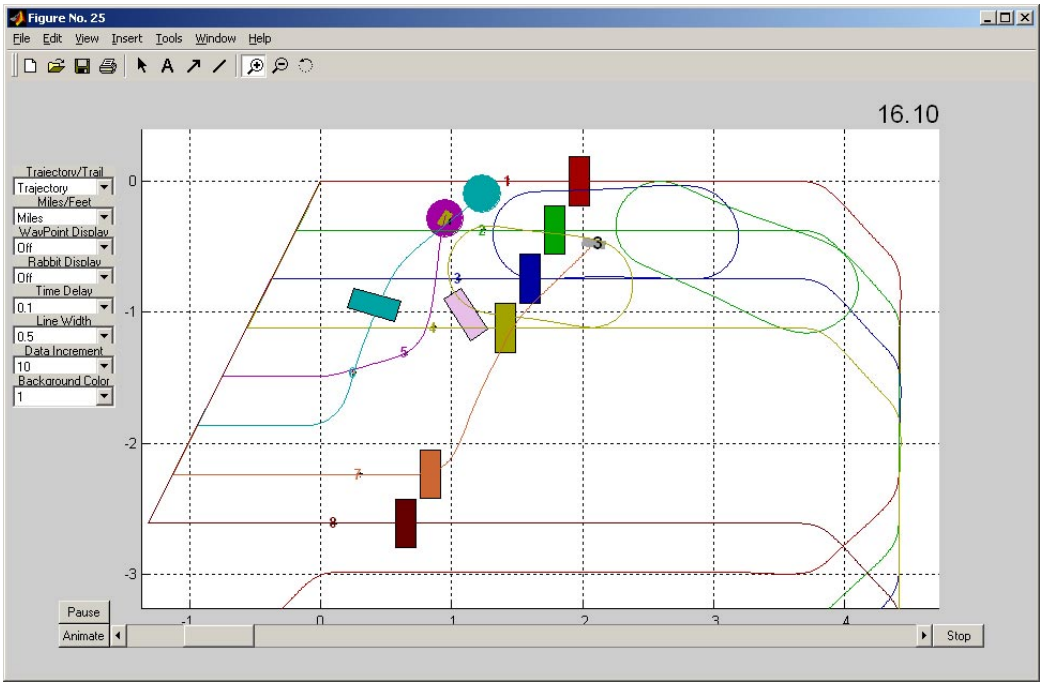


Figure 4: Simulation Data Plot Window

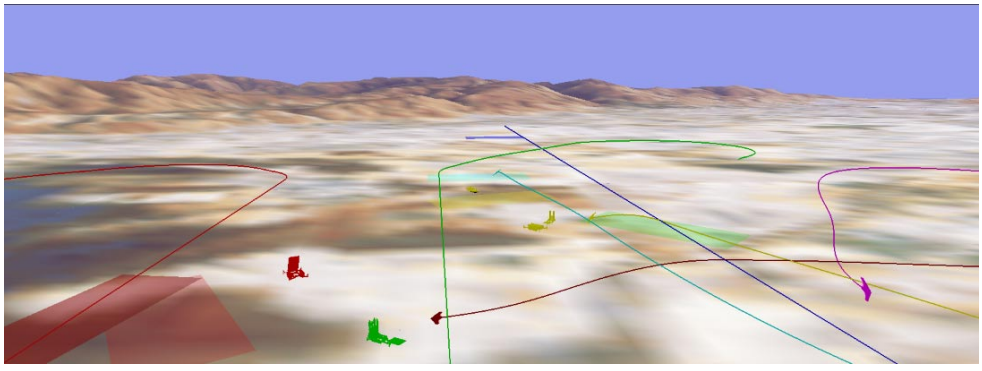


Figure 5: AVDS Example

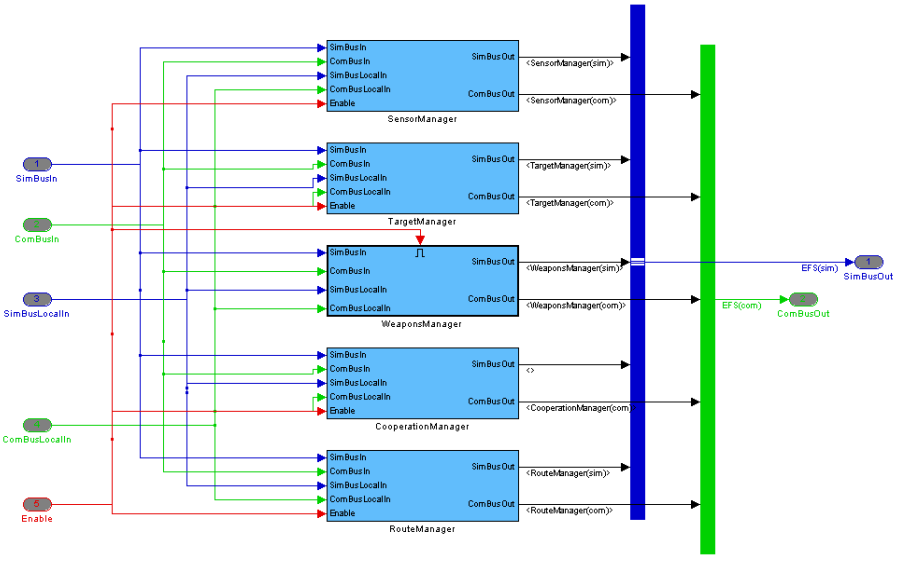


Figure 6: Embedded Flight Software Blocks

Communications between the simulated vehicles is facilitated through connections on the communication bus. 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, the following is a sequence of events that occur when a previously undetected target is discovered:

1. **Vehicle Dynamics** block senses target. Makes vehicle heading and target ID available to local vehicle.
2. The local **Sensor Manager** calculates ***single automatic target recognition (ATR)*** based on information from the Vehicle Dynamics block. Makes single ATR available to all vehicles.
3. **Sensor Managers** on all vehicles calculate ***combined ATR*** value based on information from all vehicles. Make combined ATR available to local vehicle.
4. **Target Managers** on all vehicles update the ***target state*** based on the combined ATR value from the local vehicle and the states communicated from the other vehicles. Makes the new target states available to all vehicles.
5. If any of the targets change state the ***Route Managers*** on all vehicles calculate ***optimal route and cost*** to the target based on its new state. Makes cost to service target available to all of the vehicles.
6. **Cooperation Managers** on all vehicles calculate ***optimal assignment*** of vehicles to targets based on the optimal costs. Makes assignment for local vehicle available to the local vehicle.
7. **Route Managers** on all vehicles ***implement assigned routes***. Makes assigned waypoints available to the local vehicle.
8. **Tactical Maneuvering Managers** on all vehicles read assigned waypoints and calculate ***commands*** that will cause the autopilot to cause the vehicle to fly to the waypoints. Makes autopilot commands available to the local vehicle.
9. **Vehicle Dynamics** reads autopilot commands and runs vehicle ***dynamics simulation***. Makes vehicle state available to local vehicle.

4.1 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. Based on the need for precise tracking of the planned trajectory between waypoints a trajectory following algorithm was added to the original CATA tactical maneuvering manager, see Fowler (2001). This algorithm calculates the autopilot commands necessary to cause the vehicle to follow a given trajectory between the waypoints. This ensures that the vehicle will arrive at a given waypoint at the heading required to complete the task.

4.2 Sensor Manager

This manager is used to perform all of the functions necessary to monitor the sensors and process sensed data. The Sensor Manager performs the following functions:

- Keeps track of which targets have been detected.

- Computes ATR based on sensed data from this vehicle. ATR value is calculated from an azimuth dependent function that produces templates such as the one shown in Figure 7.
- Combination of ATR values for each target from this and other vehicles.
- Calculation of a Battle Damage Assessment (BDA) value.

4.3 Target Manager

The Target Manager keeps track of the state of all of the known targets. It creates and manages list of known and potential targets. It also determines when to change a target's status to a different state based on the current state and a state transition function, see Figure 8. Specifically, the Target Manager performs the following functions:

- Classifies targets.
- Sends target information requests based on data needed for high level of confidence in classification.
- Manages the target states based on detection and specifications for probability of classification (P_c) and probability of kill (P_k), see Figure 8. The states are: Not Detected ($\overline{D}$), Detected/Not Classified ($D/\overline{C}$), Classified/Not Attacked ($C/\overline{A}$), Attacked/Not Killed ($A/\overline{K}$), Killed/Not Verified ($K/\overline{V}$), and Verified (V).
- Accounts for moving targets and target registration issues, not yet implemented.

4.4 Cooperation Manager

Calculates task and target assignments for the vehicle based on information gathered from all of the vehicles. Performs

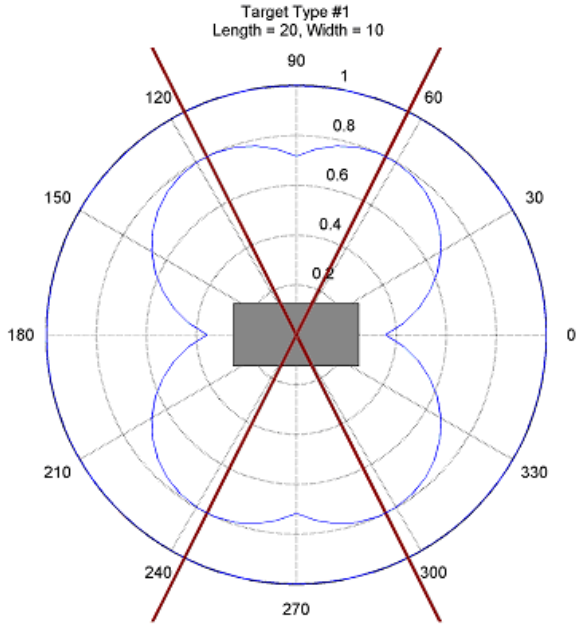


Figure 7: ATR Template

the required assignment calculations to assign each vehicle to a task. Tasks include *continue to search*, *cooperative classification*, *attack* and *verification of kill*. A network optimization model is used to calculate the vehicle task assignments, see Schumacher, Chandler, and Rasmussen (2001).

4.5 Route Manager

This manager is used to plan and select the trajectory for the vehicle to fly. The Route Manager is responsible for calculating the lowest cost route to all known targets, based on each target's state. It is also responsible for monitoring

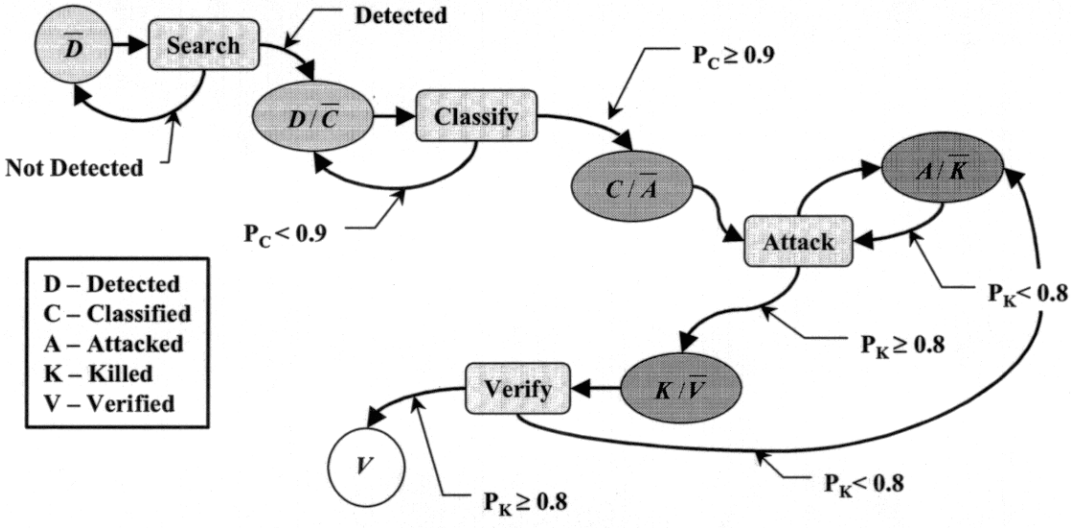


Figure 8: Target States

the status of the vehicle's assigned task. Specifically the Route manager is responsible for the following:

- Maintains matrices of waypoints that describe primary and alternate flight trajectories for the vehicle.
- Calculates new flight trajectories for the vehicle based on mission requirements.
- Relays parameters describing costs of using alternative flight trajectories to other vehicles.

4.6 Weapons Manager

Selects a weapon and then simulates its deployment. This manager is responsible for communicating a unique Bomb ID number, the type of weapon released and the weapon's impact coordinates. The targets use this data to calculate the effects of the weapon impact.

5 VEHICLE DYNAMICS SIMULATION

The vehicle dynamics are those contained in the Vehicle class of the CATA simulation. The functionality included from this class includes SDOF dynamics, a flight control system (FCS), an autopilot, and the sensor footprint. The SDOF dynamics simulate an air vehicle based on stability, control and damping derivatives and mass properties that are contained in the simulation code. An inner loop FCS was used to cause the vehicle to respond consistently to given commands at various flight conditions. The autopilot was implemented with a number of different modes to give researchers the ability to select the appropriate mode for the task. Examples of these modes include: Auto Takeoff, Altitude hold, Mach hold, and Maneuver. MultiUAV only uses the Maneuver mode, since this is the mode used by the Tactical Maneuvering manager for waypoint guidance. The sensor footprint is modeled as a rectangle a set distance offset in front of the vehicle. During the simulation, if a target is found inside of the rectangle, notification is set back to Simulink for processing through a sensor model.

In order to remove timing differences between the Tactical Maneuvering Manager and the vehicle dynamics, the interface to both of these functions was combined in one S-Function in the Aircraft Dynamics block. The S-Function " TacticalVehicle" S-function contains calls to the Vehicle class from the CATA simulation. The TacticalVehicle S-function uses the inputs to the "Vehicle Dynamics" block as well as the contents of the global variable "VehicleMemory.Dynamics" to initialize and update the vehicle mode.

6 SIMULATION EXAMPLE

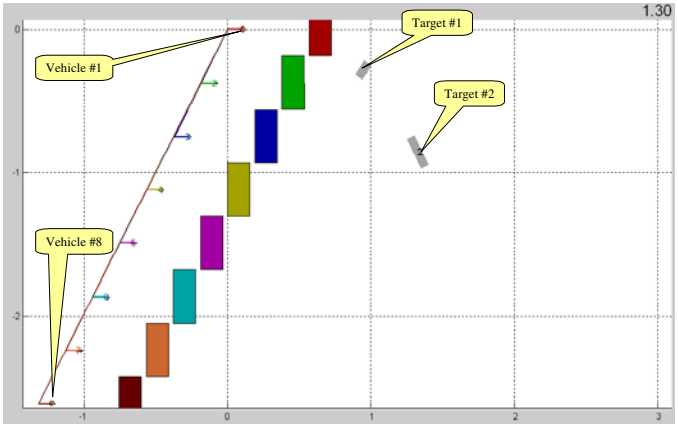
As an example, MultiUAV was setup with eight vehicles and two targets as shown in Figure 9(a). The vehicles are initialized with waypoints that cause them to follow a predefined search pattern. As the simulation progresses, the vehicles will continue to follow these predefined waypoints until they are assigned a different task. The simulation was operated for 200 seconds simulation time. The following are descriptions of the state of the simulation at the indicated times:

- **Time 8.1 seconds, Figure 9(b):**
  Vehicle #2 has just discovered target #1. At this point, vehicle #5 is assigned to classify target #1.
- **Time 23.5 seconds, Figure 9(c):**
  Vehicle #5 has classified target #1 and has been assigned to attack it. Also, vehicle #3 has discovered target #2 and vehicle #7 has been assigned to classify it.
- **Time 36.7 seconds, Figure 9(d):**
  Vehicle #5 has attacked target #1 and vehicle #4 has been assigned to verify it. Vehicle #7 classified target #2, was assigned to attack it and attacked. Vehicle #6 was assigned to verify target #2.
- **Time 55.9 seconds, Figure 9(e):**
  Vehicle #4 has verified the killing of target #1 and is returning to the point where it left its original search waypoints. Vehicle #6 has just verified the killing of target #2.
- **Time 74.9 seconds, Figure 9(f):**
  Vehicles #4 and #6 are following their original search waypoints.

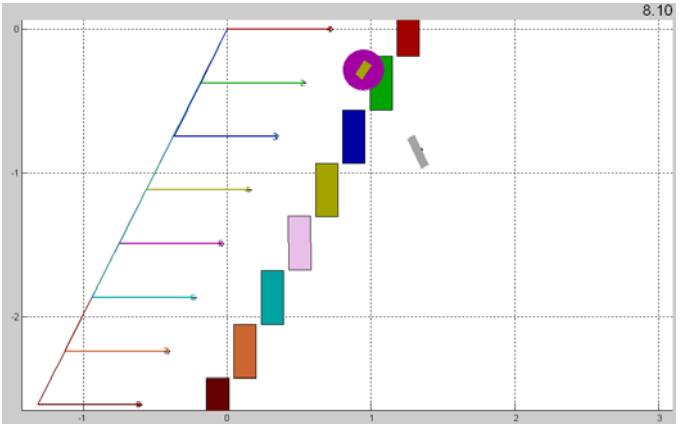
During the simulation MultiUAV prints out status messages to the MATLAB console window. The following are the status messages printed out in the first 8 seconds of the simulation example. Notice that many of the messages are redundant. This is because all of the vehicles track the target states. And all of them report their own status.

```
*** Start CATA/Simulink Simulation ***
0.01 luv 1 cycles past waypoint # 0
0.01 luv 2 cycles past waypoint # 0
0.01 luv 3 cycles past waypoint # 0
0.01 luv 4 cycles past waypoint # 0
0.01 luv 5 cycles past waypoint # 0
0.01 luv 6 cycles past waypoint # 0
0.01 luv 7 cycles past waypoint # 0
0.01 luv 8 cycles past waypoint # 0
5.77: LUV 2 Found Target ID# 1
7.60 Vehicle#1: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.60 Vehicle#2: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.60 Vehicle#3: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.60 Vehicle#4: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.60 Vehicle#5: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
```

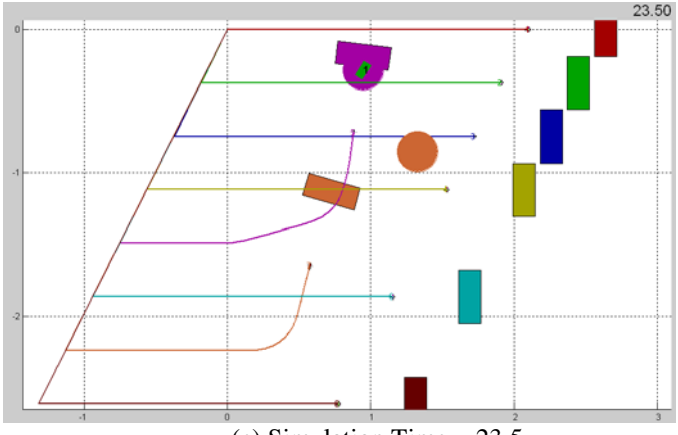
```
7.60 Vehicle#6: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.60 Vehicle#7: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.60 Vehicle#8: Target #1's state has been
changed to: "Detected-Not-Classified". ATR
Metric = 0.709971
7.70 Vehicle #1, Replanning
7.70 Vehicle #2, Replanning
7.70 Vehicle #3, Replanning
7.70 Vehicle #4, Replanning
7.70 Vehicle #5, Replanning
7.70 Vehicle #6, Replanning
7.70 Vehicle #7, Replanning
7.70 Vehicle #8, Replanning
8.00 Vehicle #5, Assigned Target #1 to: Clas-
sify Target
```



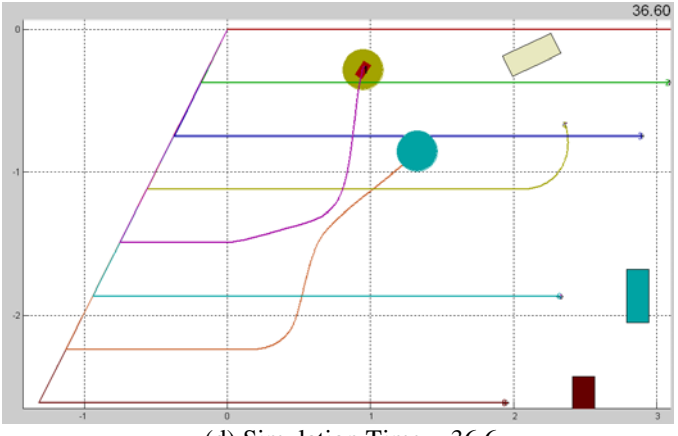
(a) Simulation Time = 1.3



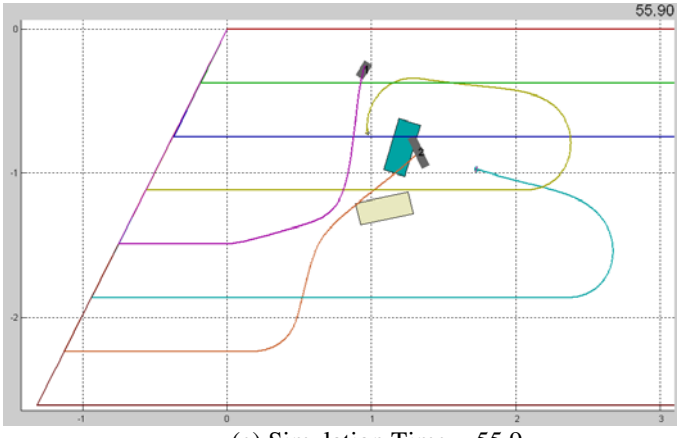
(b) Simulation Time = 8.1



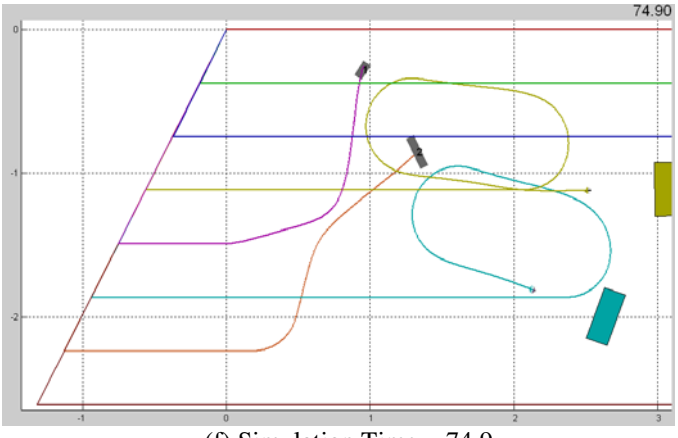
(c) Simulation Time = 23.5



(d) Simulation Time = 36.6



(e) Simulation Time = 55.9



(f) Simulation Time = 74.9

Figure 9: Snapshots Of The MultiUAV Example At Various Times During The Simulation

This simulation scenario demonstrates the capability of autonomous UAVs to find and prosecute targets. At the conclusion of the simulation both targets had been discovered, classified, attacked and verified.

7 CONCLUSION

This paper described a multiple UAV simulation used to investigate cooperative control algorithms. This simulation has made it possible for researchers knowledgeable in MATLAB and Simulink to evaluate their UAV cooperative control algorithms. In order to give research institutions access to it, MultiUAV was made available to public. This has made it possible for a number of academic institutions and contractors to integrate MultiUAV into their research programs.

Implementing cooperative control algorithms in MultiUAV and running scenarios such as the one in the previous section, has uncovered many issues in cooperative control of UAVs. These issues are currently being addressed in work at AFRL. As for the future, MultiUAV will be modified to meet simulation challenges that arise during AFRL's cooperative control research. There are immediate plans to make the connections between the vehicles in MultiUAV more flexible. This will enable users to increase the number of vehicles, add detailed simulated communications, and allow the vehicles to be connected to simulations outside of MultiUAV.

REFERENCES

Boeing. 1997. "Control Automation and Task Allocation". Air Force Research Laboratory. Final Report. Boeing.

Chandler, P., S. Rasmussen, and M. Pachter. 2000. "UAV Cooperative Path Planning". In *Proceedings of the 2000 American Institute of Aeronautics and Astronautics Guidance Navigation and Control Conference*. Denver, Colorado.

Fowler, Jeffery M. 2001. "Coupled Task Planning for Multiple Unmanned Air Vehicles". Technical Report. AFRL/VACA WPAFB, Dayton, Ohio.

RasSimTech Ltd. 2002. "AVDS". Available via <http://www.RasSimTech.com/.> [accessed August 26, 2002]

Schumacher, Corey, Phillip R. Chandler, and Steven J. Rasmussen. 2001. "Task Allocation For Wide Area Search Munitions Via Network Flow Optimization". In *Proceedings of the 2001 American Institute of Aeronautics and Astronautics Guidance Navigation and Control Conference*. Montreal, Canada.

The Mathworks, Inc. 2002. "Simulink". Available via <http://www.mathworks.com/> [accessed August 26, 2002]

AUTHOR BIOGRAPHIES

STEVEN RASMUSSEN is a Senior Aerospace Engineer for Veridian. He is currently assigned as a consultant to the U.S. Air Force Research Laboratory's Control Science Center of Excellence. His background is in flight control research and simulation development. He is the author/co-author of numerous conference and journal papers and one text book. His email address is <steven.rasmussen@afrl.af.mil>.

PHILLIP CHANDLER is the manager and principal investigator for UAV autonomous and cooperative control in-house basic research efforts for the U.S. Air Force Research Laboratory's Control Science Center of Excellence. He has over 20 years experience in basic, and advanced flight control research and has authored dozens of conference and journal papers. His email address is <phillip.chandler@afrl.af.mil>.