

# Georgia Tech UAV Software Systems

S. Kannan\*, J. Hur†, G. Saroufiem, I. Yavrucuk  
*Georgia Institute of Technology, Atlanta, GA 30332* ‡

In order for Unmanned Aerial Vehicles to perform complex missions and collaborate with other Unmanned Vehicles, significant advances in software architecture must be made. A particular configuration while valid at the start of a mission may not be ideal when a new situation arises. This paper briefly describes the *Virtual Resource Network* (VRN) concept which allows reconfiguration to take place during the mission. The VRN provides the unique ability to maximize the use of all available resources by reconfiguring to mission demands online. The Aerial Robotics Mission configuration and the set of resources required to accomplish it is also presented.

## Terminology

### Abbreviations

VRN	Virtual Resource Network
CORBA	Common Object Request Broker
COTS	Commercial-Off-The-Shelf
IDL	Interface Definition Language
OMG	Object Management Group
VTOL	Vertical Take-Off and Landing
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
USV	Unmanned Submersible Vehicle
Prolog	Programming in Logic

### Nomenclature

$V_{XC}$	Forward velocity command
$V_{YC}$	Sideward velocity command
$Z_C$	Altitude command
$\Psi_C$	Heading command
$\delta_c$	Collective
$\delta_e$	Longitudinal cyclic
$\delta_a$	Lateral cyclic
$\delta_p$	Pedal
$\Phi_C$	Roll angle command
$\Theta_C$	Pitch angle command

---

\*Team Lead.

†Pilot.

‡All authors belong to the UAV Research Facility, School of Aerospace Engineering, Georgia Institute of Technology. <http://uav.ae.gatech.edu>

## Introduction

AS the Aerial Robotics Mission has evolved over the past ten years, the intelligence exhibited by the total system has now become paramount. Most of the hardware may now be bought as robust Commercial-Off-The-Shelf (COTS) components. However, software for Unmanned Systems still lags behind in robustness, reliability, flexibility, reusability and repeated performance.

The intelligence of the system arises mostly from the decision making capability of the control algorithms. However, just having an inference system (intelligence) with many rules and the ability to make complex decisions is not enough. The inference system must have at its disposal, the mechanisms to initiate change in its software systems in order to effectively adapt to a changing environment.

Ongoing research at Georgia Tech has tackled this issue from a general view point. The requirements of such a reconfigurable software architecture were laid out without introducing any mission specifics. The main requirements were real-time performance and highly decoupled interaction between the different components of the system. This approach allows the same software/hardware components to be used repeatedly for different missions by simply assembling the necessary components and defining the interactions between

them. Additionally, the architecture also provides the mechanisms to rewire (or reconfigure) the components as the mission changes. This architecture, called the *Virtual Resource Network* is described later.

This paper also describes the components required to perform the Aerial Robotics Mission. The components that have been developed so far and the status of components yet to be developed.

## Virtual Resource Network

The *Virtual Resource Network* or *VRN* in short is a distributed computing system built in C++. It forms the core of the new generation software for UAVs at Georgia Tech. The central concept behind the *Virtual Resource Network* is networks of resources interacting with each other in order to achieve a certain functionality. Such functionality could be at a very low level such as the flight controller interacting with the sensors and actuators to provide stability or at a high level where a UAV interacts with a UGV to perform a certain task. Here, the sensors, flight controller, actuators, UAV and UGV are all simply considered to be resources on a network which may be wired up to perform a certain mission or reconfigured as the mission changes.

### Communication Protocol

The distributed communication system follows the standards set by the Object Management Group (OMG). This standard advocates Common Object Request Broker (CORBA)<sup>1</sup> technology to achieve seamless distributed communication between objects running on different computers in a network. One of the central features of CORBA technology is expressing the interfaces to different objects in a language independent manner. Figure 1 illustrates the use of the CORBA<sup>1</sup> Interface Definition Language (IDL) to express the interfaces between components. The IDL explicitly states how one object may talk across the network with other objects. The IDL also serves to document the functionality of the object without going into the details of how the

```
interface Sensors
{
    NavData getNavigationData();
}

interface Actuators
{
    setActuatorPosition(Position pos);
}
```

**Fig. 1 IDL for Sensors and Actuators**

```
uav.controller.neural-net
uav.controller.pid
uav.missionplanner.grid-search-algorithm
uav.sensors.gps
ugv.missionplanner.trajectory-commander
ugv.sensors.gps
```

**Fig. 2 Naming Convention for the VRN**

object is implemented. This standardizes the interfaces across the system. In this paper objects are sometimes referred to as resources or components and these terms will be used interchangeably depending on the context.

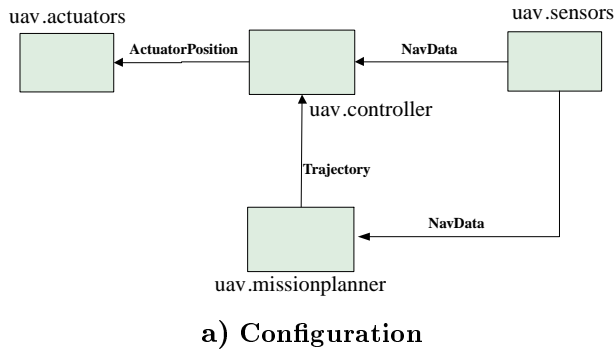
### Resource Network Naming

The *VRN* employs a network naming convention that uniquely places each resource logically into a tree. Resources can then be looked up on the network and configured to interact with each other. Figure 2 illustrates VRN Naming conventions.

Although resources may be named in any fashion, they usually follow a hierarchical tree structure. This naming system has been implemented using the industry standard CORBA Naming Service.<sup>1</sup> Each of the resources (usually software objects) could be running on any processor on the network. The user or any other resource may perform a lookup and interact with the software object or hardware.

### Configurator

Having established the ability to lookup software objects and interact with them irrespective of their physical location, the next step is defining how these resources interact with each other to perform a mission. Once all necessary resources are started up on their re-



```

uav.sensors.outputs (NavData)
uav.actuators.inputs (ActuatorPosition)
uav.controller.inputs (NavData, Trajectory)
uav.controller.outputs (ActuatorPosition)
uav.missionplanner.outputs (Trajectory)
uav.missionplanner.inputs (NavData)

```

### b) Configuration Script

**Fig. 3 Configuration and associated script that wires up the controller, sensors, actuators and mission planner**

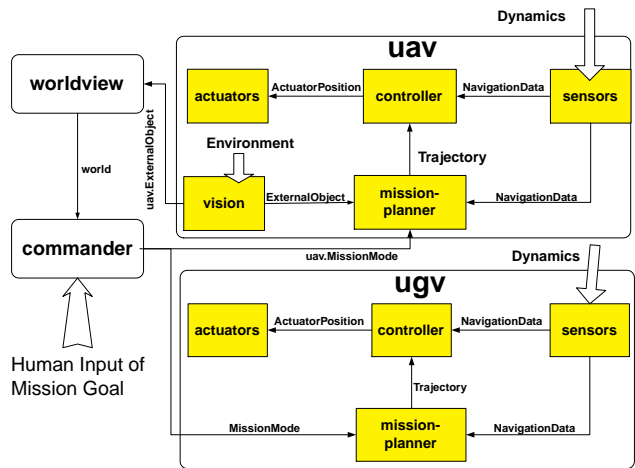
spective processors, a final configuration step is performed where resources are wired up. Figure 3(a) presents a typical wireup of sensors, flight controller, mission planner and actuators on a VTOL UAV. Figure 3(b) illustrates the script needed to wireup the components.

Although not powerful in a static configuration, this method of configuring resources provides the mechanism by which reconfiguration of existing resources may occur depending on the mission situation. Additionally it cuts development time in the laboratory because the same algorithms may be used in more than one UAV or even on UGVs. A good example is the commonality of `uav.sensors.gps` and `ugv.sensors.gps`.

## Aerial Robotics Mission Configuration

Using the *Virtual Resource Network* as a foundation, the software for the Aerial Robotics Mission may be configured.

Figure 4 illustrates the resources needed to perform the Aerial Robotics Mission. No discrimination between hardware and software resources is made. Each resource is an isolated



**Fig. 4 Top Level configuration of resources for the Aerial Robotics Mission.**

component providing a service. The interaction between the resources is in a decoupled fashion and may be reconfigured at any time. If for example a new resource, `uav2`, becomes available while the mission is underway, the new UAV resource may be dynamically added to the configuration and used in the mission.

Figure 8 lists all the components necessary to perform the Aerial Robotics Mission and the following sections describe the resources that have been developed at Georgia Tech to date.

### uav.sensors

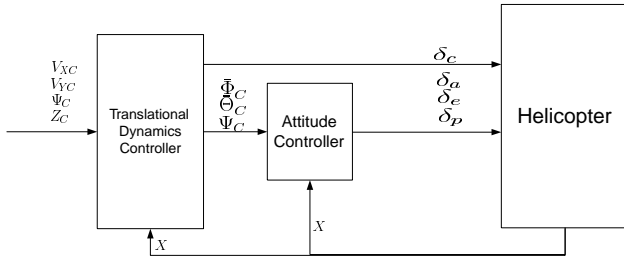
Sensor drivers for the Boeing IMU (`uav.sensors.imu`) and Novatel GPS (`uav.sensors.gps`) have been developed. Although these drivers have been flight tested on the Yamaha R-50 helicopter, they are stand-alone software systems that may be configured for use dynamically on other UAVs or UGVs.

### uav.controller

The flight controller consists of two cascaded controllers.

- The translation controller (`uav.controller.translation`)
- The attitude controller (`uav.controller.attitude`)

VTOL vehicles are manoeuvred using attitude control. Figure 5 is a schematic of



**Fig. 5 uav.controller schematic**

uav.controller. The inner most loop that has the fastest dynamics is the attitude controller. It takes commanded attitude as inputs and generates the actuator deflections that will result in that attitude. The outerloop controls the slower translational rate variables. It takes commanded velocity as inputs and generates the attitude commands that will produce the desired velocity.

The attitude controller is also called the Attitude Command Attitude Hold (ACAH) system. The ACAH controller deviates significantly from conventional methods of attitude control. One of the primary objectives was to provide a trajectory following system that is valid at any point on the flight envelope of the UAV. This has significant advantages in that the controller does not have to be re-tuned or calibrated for different payload configurations or atmospheric conditions.

Figure 6(a) illustrates the details of the attitude controller.<sup>2</sup> This controller was developed at Georgia Tech in collaboration with Boeing and Guided Systems Technologies. It has been flight tested on the Yamaha R-50 VTOL UAV<sup>3</sup> at Georgia Tech and is currently being transitioned for flight demonstration on the X-36 fixed wing UCAV at Boeing.<sup>4</sup>

The controller uses linear model inversion to generate control deflections. However, the linear model is only an approximation of the real dynamics of the aircraft and is subject to modeling errors arising from flight conditions and inaccurate modeling. Hence, an adaptive unit, the neural network, is used to cancel the inversion errors using feedback and a stable update law derived using Lyapunov theory. This same structure is used in all three channels, roll,

pitch and yaw. The adaptive unit also adjusts to changing atmospheric conditions and dynamics. Thus, the controller may be used at different points on the flight envelope without tuning. It is anticipated that the same controller may be used on different airframes such as the R-50 and Bergen UAVs without tuning.

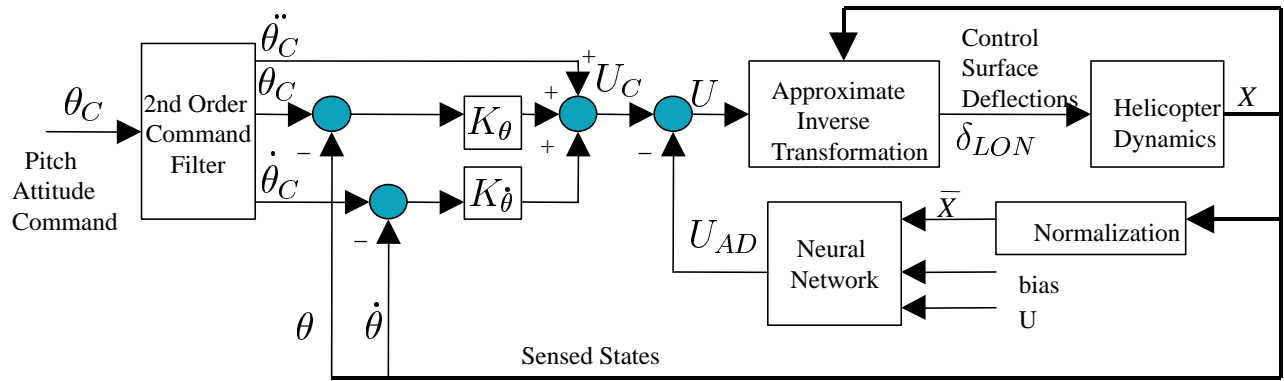
With a pilot in the loop, the attitude commands may be generated by the pilot, but, this being an autonomous system, it is more convenient to command velocities and have a translation controller<sup>5</sup> generate the attitude commands necessary to achieve the commanded velocity. Figure 6(b) shows the details of this combined system.

#### uav.vision

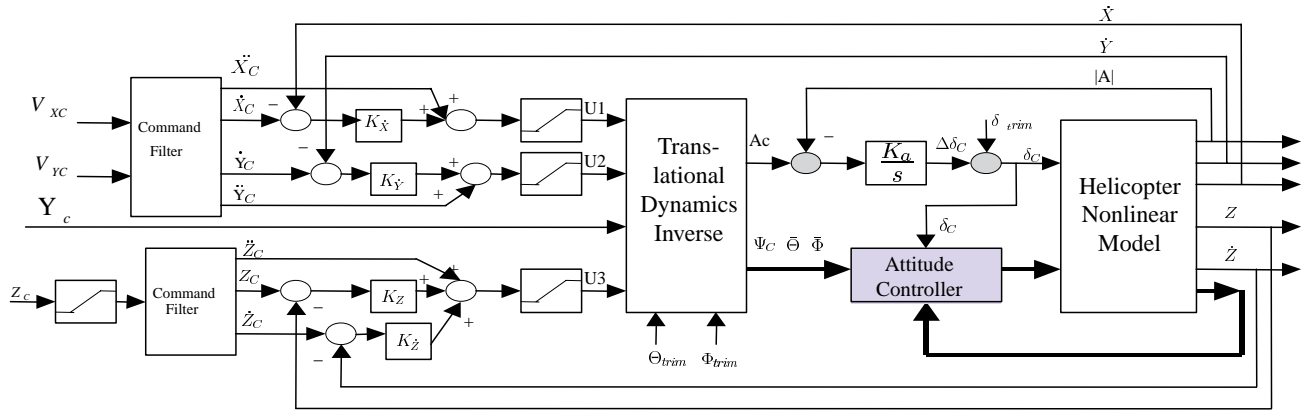
The vision system consists of a commercial PCI image processing board from Datacube systems and associated software algorithms for tracking and shape recognition. Additionally, custom filters are being developed in order to satisfy the requirements of the Aerial Robotics Mission. Analog video signals will be transmitted to a ground system running Windows NT where real-time processing of images will take place.

#### intelligence.worldview

This resource builds a virtual world of the environment sensed by the UAV. This virtual picture represents the global view of the known world to all unmanned vehicles on the network. Hence, even though the UGV has no vision system or any other external sensors, it has immediate access to its environment as sensed by the UAV. Any global decisions to be taken may be inferred from this virtual world. Note that this resource takes inputs of "ExternalObject" which could be sensed from a vision system or radar or any other system capable of sensing the environment. Additionally this information may come from more than one UAV or UGV or USV. The source of information and method of sensing is irrelevant. The end result is that each unmanned vehicle's environment is known to all resources



a) The helicopter attitude controller, Attitude Command Attitude Hold (ACA) system.



b) The overall trajectory controller containing both the translation controller and the attitude controller.

Fig. 6 uav.controller detailed block diagram

on the network. This resource is implemented as expert system rules on a blackboard using the Prolog (Programming in Logic) language.

### intelligence.commander

This is the only place where the human operator may interact with the whole system. It is here that the human sets the overall goal to be attained by the set of available resources. Prolog is a goal oriented language and is one of the primary languages used for Artificial Intelligence. It uses a set of known facts (intelligence.worldview) a set of rules (intelligence.missiongoal) and an inference system to process known facts in such a way as to attain the overall mission goal. The goal and rules to achieve the mission are stated together as illustrated in Figure 7 for the Aerial Robotics Mission.

```

goal :-
    recon(ArenaExtents, uav);
    investigate(Obj1, Obj2, ..);
    groundRescue(Survivors);
    endMission;

recon(ArenaExtents, uav) :-
    uav.MissionMode(takeoff, 15ft);
    uav.MissionMode(gridsearch, ArenaExtents);

investigate(Obj1) :-
    uav.MissionMode(loiter, 3ft, Obj1);

groundRescue(Survivors) :-
    ugv.MissionMode(goto, Survivor.position);

endMission :-
    uav.MissionMode(land, home);
    ugv.MissionMode(return, home);

```

Fig. 7 Mission Goal input by human

## Summary and Conclusions

Georgia Tech has taken a long term and generic approach to the development of systems for Unmanned Vehicles. Expertise in the

development of flight control systems, artificial intelligence and avionics hardware integration already exist. However, the central glue that will integrate both hardware and software in a real-time distributed computing environment has not existed in the past. This system called the Virtual Resource Network and the associated industry standards such as CORBA provide an elegant solution to this problem. It is anticipated that once these enabling technologies<sup>6,7</sup> have been fully developed and tested for robustness, the business of developing components specific to different missions may begin.

The *Virtual Resource Network* forms a part of the larger *Open Control Platform* technology development program at Georgia Tech funded under the DARPA Software Enabled Control Project. The first version of the *Virtual Resource Network* was successfully demonstrated to DARPA and the Air Force in May 1999. Following this demonstration, UAV specific components are currently being integrated into the Open Control Platform. This is expected to be complete by the end of 1999 with flight tests in early 2000.

## Acknowledgements

The authors would like to acknowledge the efforts of Prof. Linda Wills of Electrical Engineering, Prof. J.V.R Prasad and Prof. Dan Schrage of Aerospace Engineering, Dr. Eric Corban of Guided Systems Technologies, Brian Mendel of the Boeing Phantom Works and the invaluable contributions of Doug Schmidt's group at Washington University.

## References

<sup>1</sup>Object Management Group, *CORBA 2.2 Common Object Services Specification*, Dec. 1998, <http://www.omg.org>.

<sup>2</sup>Kim, B. S. and Calise, A. J., "Nonlinear Flight Control Using Neural Networks," *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 20-1, 1997, pp. 26-33.

<sup>3</sup>Prasad, J. V. R., Calise, A. J., and Corban, J. E., "Implementation of Adaptive Nonlinear Controller for

Flight Test on an Unmanned Helicopter," *Proceedings of the IEEE Conference on Decision and Control*, 1998.

<sup>4</sup>Calise, A. J., Lee, S., and Sharma, M., "Direct Adaptive Reconfigurable Control of a Tailless Fighter Aircraft," *AIAA Guidance, Navigation, and Control Conference*, Boston, MA, August 1998.

<sup>5</sup>Prasad, J. V. R., Calise, A. J., Corban, J. E., and Pei, Y., "Adaptive Nonlinear Controller Synthesis and Flight Test Evaluation on an Unmanned Helicopter," *1999 IEEE International Conference on Control Applications*, 1999, To be presented.

<sup>6</sup>Kannan, S. K., Prasad, J. V. R., Schrage, D. P., and Yavrucuk, I., "Simulation and Flight Control Integration using the Open Control Platform for Unmanned Aerial Vehicles," *18th AIAA Digital Avionics Conference*, 1999, To be presented.

<sup>7</sup>Freeman, R., Vachtsevanos, G., and Clements, N., "Software Enabled Control for Unmanned Aerial Vehicles," *18th AIAA Digital Avionics Conference*, 1999, To be presented.

<b>Resource</b>	<b>Inputs</b>	<b>Outputs</b>	<b>Function</b>
<b>uav.sensors</b>	Environment	uav.NavigationData	Fuses Attitude and Position Information to produce consolidated, filtered NavigationData
uav.sensors.gps	Satellite + Differential	uav.Position	Novatel GPS Driver
uav.sensors.imu	Dynamics	uav.Attitude	Boeing IMU / Watson AHRS Driver
uav.sensors.alt	Dynamics	uav.Altitude	Sonic, altimeter driver
<b>uav.actuators</b>	uav.ActuatorPosition	Analog Signal	Actuators driver, specific to vehicle, R-50 / Bergen
<b>uav.controller</b>	uav.Trajectory uav.NavigationData	uav.ActuatorPosition	Takes trajectory commands and produces actuator deflections that will produce the required velocity. In reality this resource uses one or more controller resources to achieve its objective.
uav.controller.translation	uav.Trajectory uav.NavigationData	uav.AttitudeCommand	Takes translational velocity commands and transforms them into the required attitude to achieve the commanded velocity vector
uav.controller.attitude	uav.AttitudeCommand	uav.ActuatorPosition	Neural Network adaptive controller which takes uav attitude commands and generates the actuator commands.
<b>uav.missionplanner</b>	uav.MissionMode	uav.Trajectory	Takes very high level mission mode commands and uses appropriate algorithms to generate trajectory that will put uav into the desired mode.
uav.missionplanner.takeoff	uav.MissionMode	uav.Trajectory	Generates trajectory to let uav takeoff and achieve a desired altitude.
uav.missionplanner.loiter	uav.MissionMode	uav.Trajectory	Puts the helicopter into loiter mode which usually means hover or circumnavigate object of interest from a given distance.
uav.missionplanner.land	uav.MissionMode	uav.Trajectory	Gets uav to land immediately or land at home base.
uav.missionplanner.recon	uav.MissionMode	uav.Trajectory	Generates trajectory that will map a closed area whose extreme coordinates are given.
<b>uav.vision</b>	uav.DigitalImages[]	uav.ExternalObject	Takes a series of digital images and applies different image processing algorithms to determine if there are any objects of interest in the external environment.
uav.vision.heatdetection	uav.DigitalImage	uav.ExternalObject	Takes a given digital image and generates an ExternalObject event if fire or heat is detected. Usually an Infra-Red sensor.
uav.vision.shaperecognition	uav.DigitalImage	uav.ExternalObject	Generates event if any shapes, drums/people are recognized.
uav.vision.motionrecognition	uav.DigitalImage[]	uav.ExternalObject	Generates event if any motion occurs within a time series of digital images.
<b>ugv.sensors</b>	Environment	ugv.NavigationData	Uses ugv.sensors.gps and ugv.sensors.direction to get position and heading of ugv.
<b>ugv.actuators</b>	ugv.ActuatorPosition	Analog Signal	Generates suitable signals to drive ground vehicle at given speed and direction.
<b>ugv.missionplanner</b>	ugv.MissionMode	ugv.Trajectory	Takes high level mission mode commands such as east at 1 ft/s and generates detailed trajectory of variables to get ugv into commanded mission mode.
<b>ugv.controller</b>	ugv.Trajectory, ugv.NavigationData	ugv.ActuatorPosition	Generates the required actuator position for ugv to follow given trajectory
<b>intelligence.worldview</b>	uav.ExternalObject		This resource keeps a global view of the environment, the position of the uav and ugv and also any external objects detected by the uav.
<b>intelligence.commander</b>	Overall Mission Goal	uav.MissionMode ugv.MissionMode	The overall commander of the mission. It generates very high level MissionMode commands that will fulfil the overall mission goal.

**Fig. 8** Details of resources required for the Aerial Robotics Mission