A Distributed Architecture for Intelligent Unmanned Aerial Vehicle Experimentation

P. Doherty, P. Haslum, T. Merz, E. Skarman,
G. Conte, S. Duranti, F. Heintz, P-O. Pettersson, T. Persson, B. Wingman
Dept. of Computer and Information Science
Linköping University
SE-58183 Linköping, Sweden
patdo@ida.liu.se

Abstract

The emerging area of intelligent unmanned aerial vehicle (UAV) research has shown rapid development in recent years and offers a great number of research challenges for artificial intelligence. In this article, a prototype distributed architecture for intelligent unmanned aerial vehicle experimentation is presented which supports the development of intelligent capabilities and their integration in a robust, scalable, plug-and-play hardware/software architecture. The architecture itself uses real-time CORBA to support its infrastructure and it is based on a reactive concentric software control philosophy. A number of capabilities of the architecture are presented including a multi-mode flight control system for a Yamaha RMAX VTOL platform, an on-board path planning service and a dynamically reconfigurable image processing system. A research prototype system has been built, is operational and is being used in actual missions. In the article, we emphasize the characteristics of the architecture which support the integration of numerous AI technologies.

Introduction

The emerging area of intelligent unmanned aerial vehicle (UAV) research has shown rapid development in recent years and offers a great number of research challenges for artificial intelligence. Much previous research has focused on low-level control capability with the goal of developing controllers which support the autonomous flight of UAVs from one way-point to another. The most common type of mission scenario involves placing sensor payloads in position for data collection tasks where the data is eventually processed off-line or in real-time by ground personnel. Use of UAVs and mission tasks such as these have become increasingly more important in recent conflict situations and are predicted to play increasingly more important roles in any future conflicts.

Intelligent UAVs will play an equally important role in civil applications. For both military and civil applications, there is a desire to develop more sophisticated UAV platforms where the emphasis is placed on development of intelligent capabilities. Focus in research has moved from low-level control towards a combination of low-level and decision-level control integrated in sophisticated software architectures. These should also integrate well with larger net-centric based C^3I systems. Such platforms are a prerequisite for supporting the capabilities required for the increasingly more complex mission tasks on the horizon and an ideal testbed for the development and integration of AI technologies.

The WITAS[1] Unmanned Aerial Vehicle Project[2] (Doherty *et al.* 2000) is an ambitious, long-term basic research project whose main objectives are the development of an integrated hardware/software VTOL (Vertical Take-Off and Landing) platform for fully-autonomous missions and its future deployment in applications such as traffic monitoring and surveillance, emergency services assistance, photogrammetry and surveying. Basic and applied research in the project covers a wide range of topics which include both the development of traditional AI technologies, core functionalities and their pragmatic integration with other technologies in a prototype experimental UAV system.

The following is a non-exclusive list of some of the activities in the project:

- Development of a generic distributed deliberative/reactive software architecture for (aerial) robotic systems.
- Development of a helicopter control system with flight modes for stable hovering, takeoff and landing, trajectory following, and reactive flight modes for interception and tracking of vehicles.
- Development and integration of numerous AI technologies. These include both path and task-based planning systems, a chronicle recognition system for identifying complex vehicular patterns on the ground, and other high-level services for reasoning about action and change.
- Development and integration of numerous knowledge representation technologies. These include an on-board geographical information system; a dynamic object repository to anchor, manage and reason about dynamic objects

Copyright © 2003, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

[1]WITAS (pronounced *vee-tas*) is an acronym for the Wallenberg Information Technology and Autonomous Systems Laboratory at Linköping University, Sweden.

[2]This work and the project is funded by a grant from the Wallenberg Foundation.

such as vehicles discovered during mission execution; a qualitative signal processing framework for dynamic construction of trajectories and histories of world behavior with associated reasoning mechanisms; and development of knowledge structures for signal-to-symbol conversion and approximate reasoning in soft real-time.

- Development of an on-board dynamically programmable image processing system.
- Development of multi-modal interfaces (including dialogue) for ground operator/UAV communication with both speech generation and recognition capability.
- Development of simulation environments with hardware-in-the-loop for both testing and visualization of control, reaction and deliberation functionalities.

Development of such a complex system with its many components is a multi-disciplinary effort and is dependent on competences from artificial intelligence, computer science, control theory, and signal processing. One of the most important aspects in the project is the actual development and engineering of an integrated hardware/software architecture with a supporting development environment. The system must be able to support both basic research endeavors and the development and testing of applied research in both the on-board system and in simulation environments on a continual basis.

Much progress has been made in these directions and the majority of technologies listed above have been tested in a prototype VTOL system with segments of actual missions flown in an interesting *urban* environment populated with building and road structures.

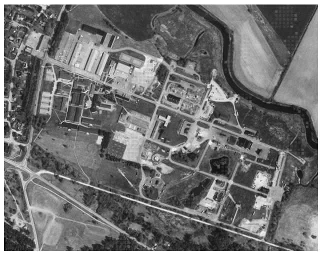
Figure 1: Aerial photo over Revinge, Sweden

Figure 1 shows an aerial photo of our primary testing area located in Revinge, Sweden. An emergency services training school is located in this area and consists of a collection of buildings, roads and even makeshift car and train accidents. This provides an ideal test area for experimenting with traffic surveillance, photogrammetric and surveying scenarios, and scenarios involving emergency services. We have also constructed an accurate 3D model for this area which has proven invaluable in simulation experiments and parts of which have been used in the on-board GIS.

The efforts described in this paper should be considered as *work in progress* and additional iterations with both the software architecture itself and the many technologies used in the architecture will be required in order to reach a level of integration and robustness necessary for safe, production versions of any future system. This being said, we do believe that any future production version of a UAV system will require many of the core functionalities which are part of the current prototype architecture in addition to the integrative and scalable nature of the architecture.

In the remainder of the paper, we will focus primarily on a description of the engineered on-board system itself and the integration of several of the AI-based services used in the software/hardware architecture. There are a great many topics that will not be considered due to page limitations, particularly in the area of knowledge representation capabilities, and in technologies such as dialogue management for support of ground operation personnel.

The VTOL and Hardware Platform

The VTOL platform used in the project is a slightly modified Yamaha RMAX helicopter manufactured by Yamaha Motor Company. It is commercially available in Japan as a radio-controlled platform.

The RMAX is approximately 2.7×0.7 meters with a main rotor 3 meters in length. It has an empty weight of 61 kg and a take-off weight of 95 kg. In addition to the RMAX sensors included with the platform, the following has been added: a Honeywell HMR3000 digital compass, a static pressure sensor, a Systron Donner digital quartz INS/GPS, temperature sensors for the system and environment, a video and a data link to the ground station, and a differential GPS. The platform is evolving and additional sensors may be added in the future.

A PC104 (Pentium P5, 266MHz) board using RTLinux[3] is currently being used as the heart of the control system. It executes all real-time control tasks involving helicopter control and collects all available sensor data. Another PC104 (Pentium PIII, 700MHz), with TimeSys Linux, is being used for control of the camera system and all on-line image processing computation. The helicopter is equipped with a Sony

FCB-EX470LP color composite video camera with a pan/tilt interface mounted on a stabilized gimbal which attenuates vibrations. A third PC104 (Pentium PIII, 700MHz) board, using (TimeSys)Linux, is used for the execution of deliberative and reactive components in the software architecture such as path planing, chronicle recognition, on-board GIS, and task procedure execution. Figure 2 shows the experimental UAV platform. Figure 3 shows a high-level schematic of the hardware platform that we have built and integrated with the RMAX platform.

The ground station includes a wireless modem connected to a ground station computer (GS) which receives telemetry from the UAV and sends commands and differential GPS data to the UAV with a downlink rate of 8kBytes/s and an uplink rate of 2 kBytes/s. There is also a receiver for composite video broadcasted from the onboard camera which can be viewed on the GS computer. A Leica reference station

[3]A migration to TimeSys Linux is in progress as part of the CORBA-ization process.

for a differential GPS is serially connected to the GS computer. An additional laptop is connected to the GS computer via an Ethernet connection and is used to visualize telemetry data. It also executes a multi-modal interface and dialogue manager. Experimentation with dialogue management is in progress where both voice commands can be sent to the UAV and voice responses are generated by the UAV(Lemon *et al.* 2001).

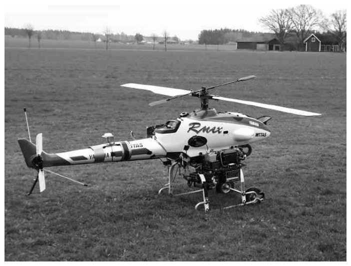

Figure 2: The WITAS UAV Platform

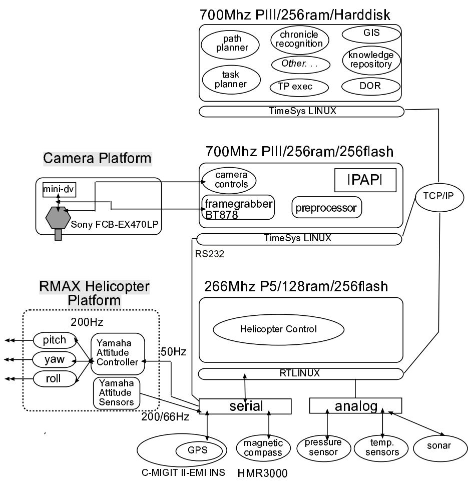

Figure 3: On-Board Hardware Schematic

Control

A great deal of effort has gone into the development of a control system for the WITAS UAV which incorporates a number of different control modes and includes a high-level interface to the control system. This enables other parts of the architecture to call the appropriate control modes dynamically during the execution of a mission.

The ability to switch modes contingently is a fundamental functionality in the architecture and can be programmed into the task procedures associated with the reactive component in the architecture. Examples will be provided later in the paper. We are currently developing and testing the following control modes:

- take-off and landing
- hovering (H-Mode)
- trajectory following (TF-Mode)
- reactive flight modes for interception and tracking (proportional navigation, PN-Mode).

All modes have been developed, implemented, tested and used in actual flights.

Proportional navigation is an interesting reactive control mode which is based on ideas from missile interception technology. It would be used most often in a scenario where information is provided about a moving object and the UAV is required to continually generate, update and follow a trajectory which would create an intercept situation between the moving object and the UAV. The moving object might be a vehicle driving on a road or another UAV. The basic idea for the control law is depicted in Figure 4. Given that A is the angle of line of sight from a helicopter to its goal relative to the North, if the change in bearing $\dot{A}$ is kept at 0, the UAV will eventually *collide* with its target. The basis for the control law for proportional navigation is that $\dot{A}$ can be controlled with the helicopter's acceleration vector.

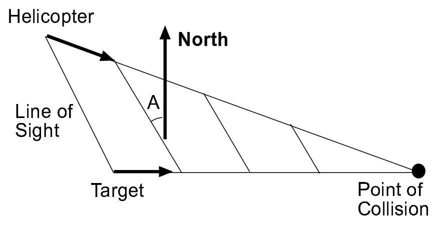

Figure 4: Proportional Navigation

Initial tests with the PN-Mode and the other modes have shown promising results. In recent flight tests, a number of fully autonomous flights have been demonstrated successfully. These include stable hovering, predefined 3D trajectory following including 360 degree banked turns, vehicle interception and road following. Acceleration and braking with no overshoot have been tested at speeds of 55 km/h and coordinated banked turns have been tested with a turn rate of 20 degrees/s. We have completed autonomous missions up to a duration of 20 minutes using a combination of control modes in addition to interaction with a ground operator.

Software System

Although the main goal of the project is to do basic research related to the development of core functionalities required in intelligent aerial robotic systems, the successful evaluation of the project requires the development of a prototype system which can in fact use these functionalities in real missions in an integrated manner. Consequently, much effort has been placed on the pragmatic development of a software architecture capable of meeting this requirement.

We have chosen Real-Time CORBA (Object Computing, Inc. 2000)[4] as a basis for the design and implementation of a loosely coupled distributed software architecture for our

[4]We are currently using TAO/ACE. **T**he **A**ce **O**rb is an open source implementation of CORBA 2.5.

aerial robotic system. We believe this is a good choice which enables us to manage the complexity of a deliberative/-reactive (D/R) software architecture with as much functionality as we require for our applications. It also ensures clean and flexible interfacing to the deliberative and control components in addition to the hardware platform via the use of IDL (Interface Definition Language).

In short, CORBA (Common Object Resource Broker Architecture) is middleware that establishes client/server relationships between objects or components. A component can be a complex piece of software such as a path planner, or something less complex such as a task procedure which is used to interface to helicopter or camera control. Objects or components can make requests to, and receive replies from, other objects or components located locally in the same process, in different processes, or on different processors on the same or separate machines. In our case, we have three onboard PC104s in addition to ground station computers.

Many of the functionalities which are part of the architecture can be viewed as clients or servers where the communication infrastructure is provided by CORBA facilities and other services such as real-time event channels. This architectural choice provides us with an ideal development environment and versatile run-time system with built-in scalability, modularity, software relocatability on various hardware configurations, performance (real-time event channels and schedulers), and support for plug-and-play software modules. Figure 5 depicts an (incomplete) high-level schematic of some of the software components used in the architecture. Each of these may be viewed as a CORBA server/client providing or requesting services from each other and receiving data and events through both real-time and standard event channels.

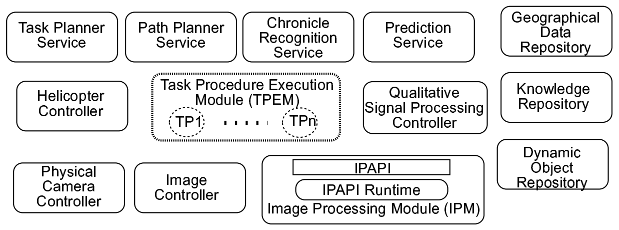
Task Planner Service
Path Planner Service
Chronicle Recognition Service
Prediction Service
Geographical Data Repository
Helicopter Controller
Task Procedure Execution Module (TPEM)
TP1
TPn
Qualitative Signal Processing Controller
Knowledge Repository
Physical Camera Controller
Image Controller
IPAPI
IPAPI Runtime
Image Processing Module (IPM)
Dynamic Object Repository

Figure 5: Some deliberative, reactive and control services

Each of these functionalities has been implemented and all are being used and developed in our applications.

The use of CORBA is commonly associated with providing a basis for loosely coupled scalable distributed component architectures on the Internet. There are also very good reasons for using CORBA as a basis for the type of deliberative/reactive architecture we are proposing here.

- **Characteristics of the D/R architecture** – The architecture is necessarily a distributed, concurrent, multi-component architecture. Communication and control is event-driven and asynchronous. Services require dynamic configuration of partial state representations of both the external environment and of the system's own internal environment. This is provided in part through CORBA event channel services and event filters. Quality-of-service guarantees are required for time-constrained communication between various services in the architecture. Real-time event channels and scheduling mechanisms integrated with the real-time OS's in the architecture contribute to meeting such guarantees.
- **Scalability and abstraction** – A long term goal of the project is the development of a scalable and modular software infrastructure for cooperative robotics where one moves from a single helicopter/ground operator system to one with helicopter fleets, ground robots, and multiple communication centers. In this case, abstraction mechanisms and scalability of the infrastructure is vital. CORBA (and similar middleware ideas) provide the uniform *glue* to support the development of large-scale net-centric systems. In addition, it supports the use of powerful design patterns associated with component-based architectures, for abstraction.
- **Development and prototyping** – We have mentioned the necessity for many different types of services associated with deliberation, reaction, control and sensing. Different programming languages are more or less suitable for the implementation of different types of services. In our case, we have implemented parts of the architecture using C, C++, Java, Erlang, LISP, Prolog, Perl, and XML tools. We have also worked with a number of legacy systems, where we have tried to integrate them rather than start from scratch. For example, we are experimenting with legacy applications and software such as Mnesia, PostgreSQL, Open-Agent Architecture, IXTET, Nuance, Festival, ArcInfo/Arcview. CORBA and the middleware idea provide the necessary tools to implement components in different languages and to integrate legacy software in a seamless manner.

Although there are a great many good reasons for using CORBA, experience has shown that there is also a downside. For example, the learning curve for CORBA is steep and real-time CORBA is a relatively new idea under continual development. It is still unclear just how deep into the control system one would like to extend the use of CORBA and real-time event channels. This is, in fact, an interesting avenue of empirical research for such architectures.

A great deal of effort in the artificial intelligence community has recently been directed towards deliberative/reactive control architectures for intelligent robotic systems. Two good sources of reference are (Arkin 1998) and (Kortenkamp, Bonassao, & Murphy 1998). Many of the architectures proposed can be viewed in a loose sense as layered architectures with a control, a reactive and a deliberative layer (see (Gat 1998)). The software architecture we have developed does have deliberative, reactive and control components, but rather than viewing it from the perspective of a layered architecture, it is best viewed as a *reactive concentric* architecture which uses services provided by both the deliberative and control components in a highly distributed and concurrent manner. At any time during the execution of a mission, a number of interacting concurrent processes at various levels of abstraction, ranging from high-level services such as path planners to low-level services such as execution of control laws, are being executed with various latencies.

The Modular Task Architecture

The conceptual layers used to describe an architecture are less important than the actual control flow and interaction between processes invoked at the various layers. What is most important is the ability to use deliberative services in a reactive or contingent manner (which we call *hybrid deliberation*) and to use traditional control services in a reactive or contingent manner (which is commonly called *hybrid control*). Figure 7 depicts some of these ideas. Due to the reactive concentric nature of the architecture, *task procedures* and their execution are a central feature of the architecture.

The modular task architecture (MTA) is a reactive system design in the procedure-based paradigm developed for loosely coupled heterogeneous systems such as the WITAS aerial robotic system. A *task* is a behavior intended to achieve a goal in a limited set of circumstances. A *task procedure* (TP) is the computational mechanism that achieves this behavior. A TP is essentially event-driven; it may open its own (CORBA) event channels, and call its own services (both CORBA and application-oriented services such as path planners); it may fail to perform its task due to the limited set of circumstances in which it is specified to operate; it may be called, spawned or terminated by an outside agent, or by other TPs; and it may be executed concurrently.

Formally, a TP is any CORBA object that implements the Witas::Idl::Tex::Task interface and adheres to the behavioral restrictions of the MTA specification.[5]

To increase the ease and flexibility of specifying and implementing individual TPs, a Task Specification Language (TSL) has been developed. TSL is an XML-based code markup language intended to simplify the implementation of TPs. The idea is that for any TP there is an application dependent or operative part which can be implemented in any host language supported by TSL. Currently TSL supports both Java and C++. The application independent part of the TP is set up automatically in a translation process from TSL to the host language. Figure 6 shows a schematic of a special type of TP specified in the TSL with some of the essential markup tags, including those for a finite state machine (fsm) block.

Task procedures can be used for many different purposes. Figure 7 depicts several types of usage as hybrid deliberation TPs, hybrid control TPs or mixed TPs.

A good way to view and represent a hybrid controller is as an augmented automaton. We have provided structural and macro tags for this purpose which can be used in a TP. Figure 8 shows a TSL schematic for the finite state machine specification which would be included in the $\langle\mathbf{fsm}\rangle \ldots \langle/\mathbf{fsm}\rangle$ tag block in Figure 6. Some additional tags not listed allow for specification of jumps to other

```
⟨tp name = Taskname⟩
⟨preamble⟩
    // Pure Host Code. Mainly used for # includes,etc.
⟨/preamble⟩
⟨declarations⟩
    ⟨parameter . . . /⟩ . . . ⟨parameter . . . /⟩
    ⟨constant . . . /⟩ . . . ⟨constant . . . /⟩
    ⟨local . . . /⟩ . . . ⟨local . . . /⟩
    ⟨native⟩ . . . ⟨/native⟩
⟨/declarations⟩
⟨services⟩
    // CORBA server objects, event channels used, etc.
⟨/services⟩
⟨init⟩
    // Host code for task specific initialization
⟨/init⟩
⟨clean⟩
    // Host code for task specific cleanup
    // CORBA cleanup handled automatically
⟨/clean⟩
⟨function⟩ . . . ⟨/function⟩ . . . //more fns
⟨start⟩
    // Executed with call to TP start() method
    // Host code plus host code macros
    // Typically will perform some setup then
    // a ⟨jump⟩ to FSM state
⟨/start⟩
⟨fsm⟩
    // Main behavioral specification in form
    // of a finite state machine
⟨/fsm⟩
⟨/tp⟩
```

Figure 6: TSL tags and partial schematic for a TP specification.

states, exits on failure and the setting up of execution checkpoints.

An important property of D/R architectures is their ability to integrate and execute processes at the deliberative, reactive and control levels concurrently and to switch between them seamlessly. Reactive components must be able to interface in a clean manner to helicopter and camera control and adapt activity to contingencies in the environment in a timely manner. Likewise, they must be able to interface to deliberative services in a clean manner. Consequently, use of TPs for hybrid control and hybrid deliberation is of fundamental importance. An example is provided in the following section.

Using Task Procedures for Flight Control

In this section, the interface between TPs for hybrid control and flight control modes, in addition to some limited hybrid deliberation, will be described. The main ingredients are TPs that support the representation of finite state machines and the use of a declarative flight command language (FCL). Based on the current state, a TP will continually generate flight commands which are sent to the flight controller and

[5]The TPEM in Figure 5 represents the set of TP CORBA objects used. There is no centralized execution mechanism.

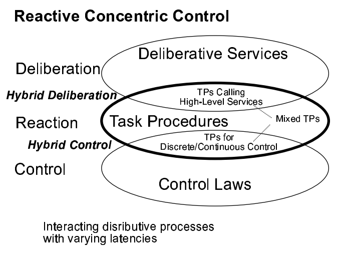
Reactive Concentric Control
Deliberation
Deliberative Services
Hybrid Deliberation
TPs Calling High-Level Services
Reaction
Task Procedures
Mixed TPs
Hybrid Control
TPs for Discrete/Continuous Control
Control
Control Laws
Interacting disributive processes with varying latencies

Figure 7: Reactive Concentric Control

⟨**fsm**⟩
⟨**statename** =*sname*⟩
⟨**action**⟩
    // Executed whenever TP enters this state
⟨/**action**⟩
    // State specific reactions to events
⟨**reaction**⟩ . . . ⟨/**reaction**⟩
⋮
⟨**reaction**⟩ . . . ⟨/**reaction**⟩
⟨/**state**⟩

//More state specifications . . .

    // Global reactions to events
⟨**reaction**⟩ . . . ⟨/**reaction**⟩
⋮
⟨**reaction**⟩ . . . ⟨/**reaction**⟩
⟨/**fsm**⟩

Figure 8: TSL tags and partial schematic for an fsm specification.

continually receive events from the event channels it subscribes to. These events make up its partial state representation.

Suppose a mission in Revinge has been generated at the deliberative level using a task-based planner such as TALPlanner (Doherty & Kvarnström 2001) to find and track a vehicle with a particular signature. A call from a deliberative component would start execution of a TP parameterized with the vehicles signature and a region to look for the vehicle. This parent TP might call another TP, NavToPt, to navigate to a waypoint in the region of interest (ROI). In order for NavToPt to do its job, it would require both deliberative and control services. It would first call a deliberative path planning service (described later) to provide a (segmented) trajectory to get the UAV to the ROI. If successful, the trajectory segments would be passed to the trajectory following flight mode in the control system (via the FlyPath TP and flight commands) and flight to the region would be initiated. During trajectory following, NavToPt would receive events from various event channels monitoring progress. An automaton for NavToPt is depicted in Figure 9.

Once the UAV arrived at (or approached) the final trajectory endpoint, the parent task procedure would make a call

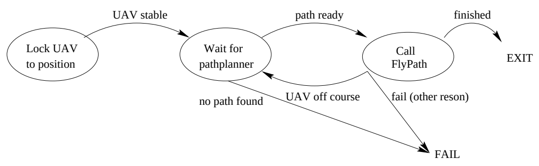

Figure 9: The NavToPt automaton

to enter hovering mode using a flight command. The parent TP would then call the image processing services (described later) and attempt identification of vehicles in the area. Suppose a vehicle matching the initial signature is identified. The parent TP would then call a new TP procedure FlyTo which would place the control of the UAV into proportional navigation mode. In fact an iteration on FlyTo with different parameters would ensue. An automaton for FlyTo is shown in Figure 10. Upon execution, the FlyTo TP generates a stream of flight commands passed to the PN flight mode controller and receives a stream of events from the flight control system and image processing system. Both automata described are in fact implemented using TPs with structure similar to that in the two TSL schemata.

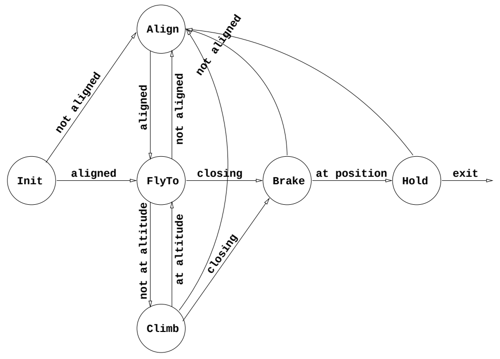

Figure 10: The FlyTo automaton

The flight command language (FCL) is used to bridge the abstraction gap between task procedures in the reactive component and flight and camera modes in the control component. Figure 11 shows a number of representative flight commands used for the proportional navigation flight control mode.

In this case the PN mode is controlled by providing commands to XY, Z and Yaw channels, in addition to an administration channel. The administration channel is used to set various parameters. It is also used to inform the PN mode which object to fly to (FlyObject), this may be a waypoint or a moving object on the ground previously identified, and the object to look at with the camera (LookObject). Additional flight commands are provided for other flight control modes and camera control. In the case of trajectory following, representations of parameterized curves are passed as arguments to flight commands, these arguments are generated via a task procedure call to the path planning service.

XY Channel	**Z Channel**
Cruise()	Land()
SlowDown()	FlyAtAltitude()
Slow()	ClimbTo()
Brake()	**Yaw Channel**
FlyToFlyObject()	FlyWithYaw()
FlyWithVelocity()	FlyWithYawOfYourself()
LockOnFlyObject()	FlyCleanly()
LockOnPosition()	
LockOnLookObject()	

Figure 11: Representative Flight Commands for Proportional Navigation.

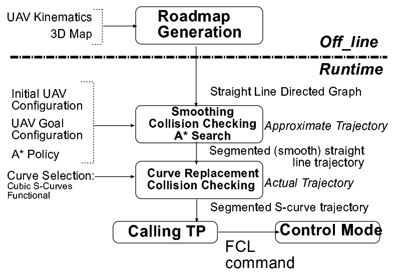

Figure 12: Generating a Flight Trajectory

Benefits of the Approach

The use of TSL with XML markup tags and a host language translator provides a modular, extensible means of constructing and experimenting with TPs that hides the syntactic overhead associated with initialization of CORBA event channels, services and error handling code. Each TP is automatically generated as a standard CORBA server and can be called, spawned, or terminated using the Witas::Idl::Tex::Task interface. We believe that the greatest flexibility is provided if the actual semantic content of a TP can be specified in a familiar language such as C++ or Java. The structuring of code in this manner also provides a clean way to analyze the behavior of a TP formally. The use of a flight command language provides a smooth transition from discrete to continuous control behavior between the reactive and control components. New control modes can be added in a modular manner and interfaced by adding new flight commands. The flight command streams themselves can be generated and analyzed in many different ways. The use of event channels and filters by TPs also provides a flexible means of dynamically constructing partial state representations which drive the behaviors of TPs.

Some Deliberative Services

Due to page limitations, we will only briefly consider two higher-level services developed, implemented and used in our applications. Emphasis will be placed on the integration of these services with the software architecture previously described.

The Path Planner Service

The path planner used for the helicopter is an adaptation of probabilistic roadmap (PR) algorithms (Kavraki *et al.* 1996) to our application domain. The problem of finding an optimal path between two robot configurations in a configuration space such as Revinge is intractable. For static configuration spaces where one assumes no dynamic objects except cars on the ground and a helicopter, PR algorithms *hedge* the intractability problem by outputing non-optimal paths and working in two phases, one off-line and the other during runtime. The main processing stages for our adaptation of the PR algorithm are shown in figure 12.

In the offline phase, a roadmap graph is generated for the area of interest (e.g.. Revinge). This process takes a 3D polygonal model of the area and the helicopter kinematics as input. Helicopter configurations[6] are randomly generated and checked for collisions with the model. An attempt is then made to connect collision-free configurations using a local path planner (see below) which takes into account the kinematic and dynamic constraints of the helicopter. Each of the local paths generated also have to be checked for collisions. The collision checker, used to check whether a given curve intersects any obstacle in the environment, is based on the OBBTree-algorithm (Gottschalk, Lin, & Manocha 1996), that uses a tree of oriented bounding boxes as its central component.

There are a number of choices that can be made for the local path planner at this stage dependent on how much or little work is chosen to be done during run-time. The choice described in the diagram is to initially ignore the non-holonomic constraints of the helicopter in the off-line phase and add them as refinements to the plan in smoothing and curve replacement phases during run-time (Sekhavat, P. Švestka, & Overmars 1998). In this case, the roadmap generated and used at run-time uses straight lines. The other alternative is to generate the roadmap with spline-curves already at this stage. We experiment with both approaches. Using these techniques, roadmaps can be generated with high-levels of coverage even in tight spaces. Using straight lines, an initial roadmap for Revinge was generated containing 5000 nodes with 97% coverage. Using spline-curves, an alternative roadmap was generated using 15000 nodes with 70% coverage.

During the mission or run-time phase, the path planning service is called with an initial and goal helicopter configuration. An attempt is made to connect the two configurations to the previously generated roadmap and, if successful, an A* search is used on the graph to generate a multi-segmented trajectory. In the diagram, the resulting straight lines are smoothed and if possible replaced with spline curves using the local path planner. In the curve replacement phase, the local path planner is separated into a horizontal and a vertical component. In the plane, two con-

[6]A helicopter configuration currently consists of three coordinates for position and one angle for direction. The orientation of the helicopter is omitted and is handled independently by the control system.

figurations are connected to each other by aligning the x-axis of the coordinate system through the two points and finding the unique cubic function that goes through these points with the appropriate directions. A change in altitude between two configurations is currently executed in a linear manner but we will in the near future move to true space curves, given as parameterized cubic polynomials.[7]

The multi-segmented trajectory generated by the path planner service is provided piecewise to the TF-Mode in the control system using an appropriate TP which monitors the flight progression as considered previously. For many of our missions in Revinge, we have used this service to generate path plans of reasonable complexity in under a second.

Image Processing

The Image Processing Module (IPM), like other services in the architecture is implemented as a CORBA server which can be called and used by other components, in particular task procedures used for achieving tasks specified at the mission level. For example, a typical scenario in traffic surveillance would be to find, identify and track a vehicle based on information about its signature (color, size, type, etc), when it was last seen and where. The services provided by the IPM include the execution of image processing algorithms used in achieving tasks. Since the goals and requirements at the mission level may change over time, one important aspect of the IPM is that it should allow for the dynamic modification and switching of IP algorithms at run-time. Internally, IP algorithms are represented using a Data Flow Graph (DFG) based model. Nodes in the graphs represent IP operations and the arcs represent data on which the operations are performed.

Conceptually, the Image Processing Module (IPM) consists of two parts,

- IPAPI - An Image Processing Application Program Interface – This is the declarative interface that other components in the D/R architecture can use to create, manipulate, configure and execute various image processing algorithms.
- IPAPI Run-Time - This is the run-time component that manages the configuration and execution of image processing algorithms, dynamically allocates memory for buffers needed during execution, interfaces to the image controller and other components in the architecture, and manages an existing library of predefined image processing algorithms.[8]

It is important to emphasize that tight integration is required between the low-level image processing capability and the deliberative or reasoning capabilities of the UAV which use qualitative models. Integration and run-time modification capabilities are achieved in part via the use of two other modules which the IPM communicates most closely with. These are the Task Procedure Execution Module

[7]The latter alternative has already been tested successfully in simulation.

[8]For details, see (Nordberg *et al.* 2002).

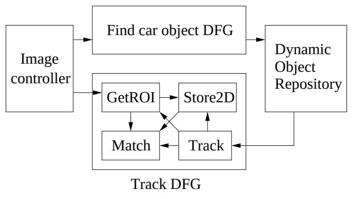

Figure 13: DFGs for identifying and tracking car objects. The ROI is chosen dynamically by sending requests to the Image Controller.

(TPEM) and the Dynamic Object Repository (DOR), depicted in Figure 5. The central role of the TPEM in the architecture has already been discussed.

The DOR is essentially a soft real-time database which keeps records of information about various objects, both static and dynamic, that the system needs to know about when achieving various tasks. This may include sighted ground vehicles during a traffic surveillance scenario, but may also include information about the helicopter itself and the camera. One task of the IPM is to provide up-to-date information about the objects it senses and to store this in the DOR. TPs determine which objects are interesting by accessing information in the DOR and using deliberative services. TPs may then redirect focus of interest by communicating with the IPM. The interaction between the TPEM and the IPM may be viewed as a form of higher-level active vision where the context which determines image processing policy is represented implicitly in the DOR and interpreted by the TPEM via the use of other deliberative services.

Recall two of the high-level tasks mentioned previously, “find vehicle” and “track vehicle”. An example of how the processing is set up for these tasks is illustrated in Figure 13. A typical result of the “find vehicle” operation is that a set of potential vehicles are identified. The terminal node of the corresponding data flow graph then exports the list of objects, acting as a CORBA client, to the DOR which is implemented as a CORBA server (right part of Figure 13). At this point, the reasoning layers of the system examine the “vision objects”, e.g., check if their velocities are consistent with the direction of the road, or if their positions are on a road, or are consistent with predictions from earlier positions. If this is the case, a hypothesis can be made that this is an “on-road object” or a “car object” after additional reasoning and linkages are created and maintained between the object structures. A chronicle recognition service can then be called to identify various patterns of interest, i.e., simple sequences of events such as changing lanes, stopping, turning, vehicle overtaking, etc. The linkage structures set up in the DOR are an important part of the signal to symbol conversions required for grounding qualitative symbol structures representing world objects such as vehicles to the sensory data associated with them.

Related Work

There is, without a doubt, much activity with UAVs in the military sector, primarily with fixed-wing high altitude vehicles. Much of the focus is on design of physical platforms, low-level control, and sensing (uavr 2001). Less focus has been placed on the type of system described here. This also applies to the majority of commercial attempts at marketing UAVs, although here, there has been more activity with VTOL platforms. Academic research with UAVs is increasing at a rapid pace. Here again, the majority of projects have focused on low-level control, although there is more activity with software architectures and integration of some deliberative capabilities. The work closest to ours is that being pursued at Georgia Tech(GT1). Rather than list publications, we refer the reader to the following (non-exhaustive) list of websites for information about interesting university UAV activities: Georgia Tech(GT1), M.I.T(MIT1), Carnegie-Mellon(CM1), U. of C., Berkeley(USCB1), Stanford University(SU1).

References

Arkin, R. C. 1998. *Behavior-Based Robotics.* MIT Press.

CM1. Carnegie Mellon University. `http://www.ri.cmu.edu/projects/project_93.html`.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. In *AI Magazine*, volume 22. 95–102.

Doherty, P.; Granlund, G.; Kuchcinski, K.; Nordberg, K.; Sandewall, E.; Skarman, E.; and Wiklund, J. 2000. The WITAS unmanned aerial vehicle project. In *Proceedings of the 14th European Conference on Artificial Intelligence*, 747–755. Project URL: `http://www.liu.se/ext/witas`.

Gat, E. 1998. *Artificial Intelligence and Mobile Robots.* AAAI Press/MIT Press. chapter 8: Three-Layer Architectures.

Gottschalk, S.; Lin, M.; and Manocha, D. 1996. Obbtree: A hierarchical structure for rapid interference detection. In *Proc. of the 23rd Int'l. Conf. on Computer graphics and interactive techniques*, 171–180.

GT1. Georgia Tech University. `http://controls.ae.gatech.edu/uavrf/`.

Kavraki, L. E.; Švestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transaction on Robotics and Automation* 12(4):566–580.

Kortenkamp, D.; Bonassao, R. P.; and Murphy, R., eds. 1998. *Artificial Intelligence and Mobile Robots.* AAAI Press/MIT Press.

Lemon, O.; Bracy, A.; Gruenstein, A.; and Peters, S. 2001. The WITAS multi-modal dialogue system I. In *Proceedings of EuroSpeech*.

MIT1. M.I.T. `http://gewurtz.mit.edu/research.htm`.

Nordberg, K.; Doherty, P.; Forssen, P.-E.; Wiklund, J.; and Andersson, P. 2002. A flexible runtime system for image processing in a distributed computational environment for an unmanned aerial vehicle. In *Proceedings of the 9th Int'l Workshop on Systems, Signals and Image Processing.*

Object Computing, Inc. 2000. *TAO Developer's Guide, Version 1.1a.* See also `http://www.cs.wustl.edu/~schmidt/TAO.html`.

Sekhavat, S.; P. Švestka, J.-P. L.; and Overmars, M. H. 1998. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *Int. Journal of Robotics Research.*

SU1. Stanford University. `http://sun-valley.stanford.edu/users/heli/`.

uavr. 2001. Unmanned aerial vehicles roadmap, 2002-2025. Office of the Secretary of Defence, USA. `http://www.acq.osd.mil/uav_roadmap.pdf`.

USCB1. University of California, Berkeley. `http://robotics.eecs.berkeley.edu/bear/`.