A Distributed Architecture for Autonomous Unmanned Aerial Vehicle Experimentation

P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman

Linköping University
Dept. of Computer and Information Science
SE-58183 Linköping, Sweden
patdo@ida.liu.se

In Proceedings of the 7th International Symposium on Distributed Autonomous Systems, 2004.

Summary. 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 distributed autonomous robotics systems. In this article, a prototype distributed architecture for autonomous 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 CORBA to support its infrastructure and it is based on a reactive concentric software control philosophy. A research prototype UAV system has been built, is operational and is being tested in actual missions over urban environments.

1 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 in the area of distributed autonomous robotics systems. Much previous research has focused on low-level control capability with the goal of developing controllers which support the autonomous flight of a UAV 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 and on abilities to interact with human operators and additional robotic platforms. 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 in turn, should also integrate well with larger network-centric based C^4I^2 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 distributed AI technologies.

The WITAS[1] Unmanned Aerial Vehicle Project[2] [4] is a 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 the development of a distributed architecture for autonomous unmanned aerial vehicles. In developing the architecture, the larger goals of integration with human operators and other ground and aerial robotics systems in network centric C^4I^2 infrastructures has been taken into account and influenced the nature of the base architecture. In addition to the software architecture, many AI technologies have been developed such as path planners, chronicle recognition and situational awareness techniques. The architecture supports modular and distributed integration of these and any additional functionalities added in the future. The WITAS UAV hardware/software platform has been built and successfully used in a VTOL system capable of achieving a number of complex autonomous missions flown in an interesting *urban* environment populated with building and road structures. In one mission, the UAV autonomously tracked a moving vehicle for up to 20 minutes. In another, several building structures were chosen as survey targets and the UAV autonomously generated a plan to fly to each and take photos of each of its facades and then successfully executed the mission.

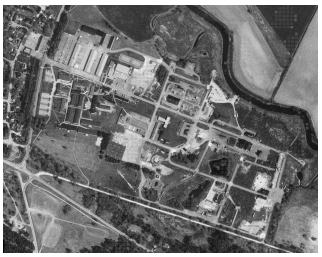

Fig. 1. Aerial photo over Revinge, Sweden

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, in addition to 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.

Fig. 2. The WITAS RMAX Helicopter

In the remainder of the paper, we will focus primarily on a description of the engineered on-board system itself, parts of the distributed CORBA-based software architecture and interaction with the primary flight control system. There are a great many topics that will not be considered due to page limitations, particularly in the area of knowledge representation [5, 7, 8], symbol grounding [14, 13], and deliberative capabilities [24, 6], task-based planning [6], specific control modes [3, 16, 27] and their support, image processing [18]

[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.

and in technologies such as dialogue management for support of ground operation personnel [26].

2 The VTOL and Hardware Platform

The WITAS Project UAV platform we use is a slightly modified Yamaha RMAX (figure 2). It has a total length of 3.6 m (incl. main rotor), a maximum take-off weight of 95 kg, and is powered by a 21 hp two-stroke engine. Yamaha equipped the radio controlled RMAX with an attitude sensor (YAS) and an attitude control system (YACS). Figure 3 shows a high-level schematic of the hardware platform that we have built and integrated with the RMAX platform.

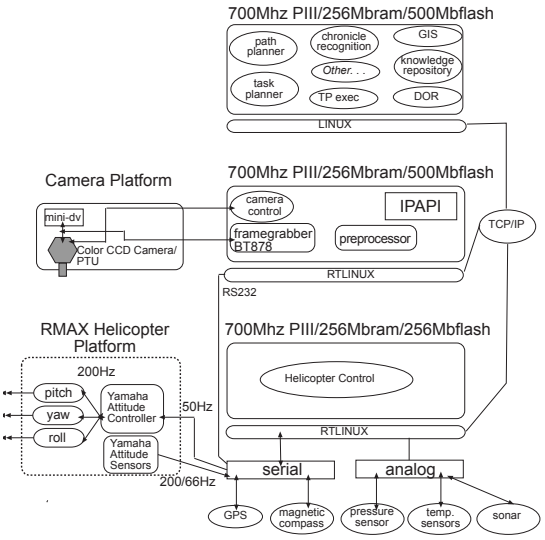

Fig. 3. On-Board Hardware Schematic

The hardware platform consists of three PC104 embedded computers (figure 3). The primary flight control (PFC) system consists of a PIII (700Mhz) processor, a wireless Ethernet bridge and the following sensors: a RTK GPS (serial), and a barometric altitude sensor (analog). It is connected to the YAS and YACS (serial), the image processing computer (serial) and the deliberative computer (Ethernet). The image processing (IP) system consists of a second PC104 embedded computer (PIII 700MHz), a color CCD camera (S-VIDEO, serial interface for control) mounted on a pan/tilt unit (serial), a video transmitter (composite video) and a recorder (miniDV). The deliberative/reactive (D/R) system runs on a third PC104 embedded computer (PIII 700MHz) which is connected to the PFC system with Ethernet using CORBA event channels. The D/R system is described in more detail in section 4.

3 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. We developed and tested the following autonomous flight control modes:

- take-off (TO-Mode) and landing via visual navigation (L-Mode, see [16])
- hovering (H-Mode)
- dynamic path following (DPF-Mode, see [3])
- reactive flight modes for interception and tracking (RTF-Mode).

These modes and their combinations have been successfully demonstrated in a number of missions at the Revinge testflight area. The primary flight control system (bottom PC104 in Figure 3) can be described conceptually as consisting of a device, reactive, behavior and application layer as depicted in figure 4a.[3]Each layer consists of several functional units which, with the exception of the application layer, are executed periodically with comparable period and worst case execution times. The implementation is based on RTLinux (GPL), which runs an ordinary Linux distribution as a task with lower priority. The application layer is realized in user space as no hard real-time execution is required, while the other layers contain functional units running as kernel modules in hard real-time.

A CORBA interface is set up between the PFC system and the deliberative/reactive system of the software architecture (top PC104 in Figure 3). Network communication between the two is physically realized using Ethernet with CORBA event channels and CORBA method calls. Task procedures in the D/R system issue commands to the PFC system to initiate different flight modes and receive helicopter states and events from the PFC system which influence the activity of the task procedure.

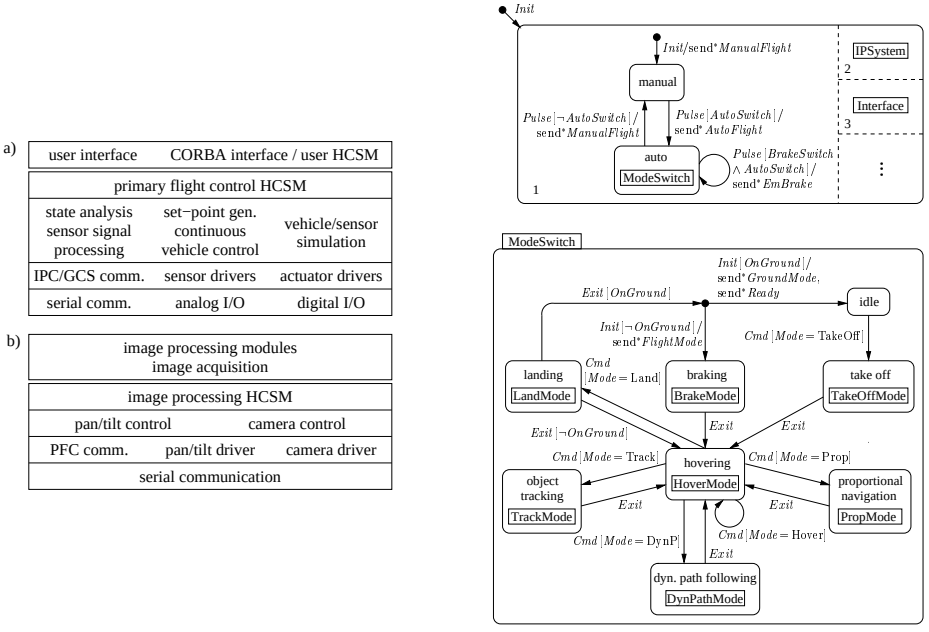

Fig. 4. PFC and IP System Schematic (left) and HCSMs for Flight Mode Switching (right)

Hierarchical concurrent state machines (HCSMs) are used to represent states of the PFC system. HCSMs are mixed Mealy and Moore machines with hierarchical and orthogonal decomposition. These are executed explicitly in the PFC system. We use a visual formalism similar to Harel's statecharts [12] to describe HCSMs.

Figure 4b shows that part of the flight control HCSM involving mode switching for different flight modes.[4]

[3]The image processing (IP) system (middle PC104 in Figure 3) has a similar structure, but will not be considered in this paper.

[4]Nested state machines are symbolized as rectangular boxes in a state node, *Pulse* is an event sent periodically, *Init* triggers a transition from an entry state (circular node) when condition holds, *Exit*

4 Software System

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 [1] and [15]. 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 [9]). 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.

We have chosen CORBA[5] as a basis for the design and implementation of a loosely coupled distributed software architecture for our 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 Request 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.

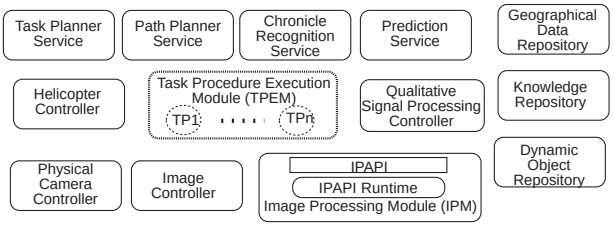

Fig. 5. Some deliberative, reactive and control services

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, using services such as standard and real-time event channels. [6] Figure 5 depicts an (incomplete) high-level schematic of some of the software components used in the architecture. Each of these functionalities has been implemented and are being used and developed in our applications. This architectural choice provides us with an ideal development environment and versatile run-time system with built-in scalability, modularity, software relocatability on various hard-

---

is sent to the superstate when entering an exit state (square node), superstate transitions are executed prior to substate transitions.

[5] We are currently using TAO/ACE [20]. **T**he **A**ce **O**rb is an open source implementation of CORBA 2.6.

[6] Event channels are specified in the CORBA Event Service standard (`http://www.omg.org/technology/document/corbaservices/_spec/_catalog.htm`) and allow decoupled passing of arbitrary data from one or more senders to one or more receivers.

ware configurations, performance (real-time event channels and schedulers), and support for plug-and-play software modules.

4.1 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 6 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.

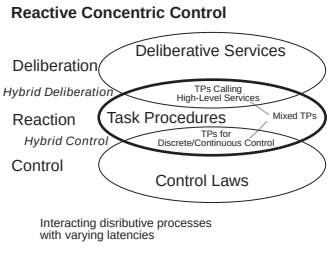

Fig. 6. Reactive Concentric Control

Formally, a TP is any CORBA object that implements the Witas::Task interface and adheres to the behavioral restrictions of the MTA specification.[7]

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 [19]. Currently TSL supports C++ and it would be straightforward to extend it to support languages such as 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 7 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 6 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 7b 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 7a. Some additional tags not listed allow for specification of jumps to other 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

[7] The TPEM (Task Procedure Execution Module) in Figure 5 represents the set of TP CORBA objects used. There is no centralized execution mechanism.

```
⟨tp name = Taskname⟩
⟨declarations⟩
        ⟨parameter . . . /⟩ . . . ⟨parameter . . . /⟩
        // other declarations, e.g. local variables and constants
⟨/declarations⟩
⟨services⟩
        // CORBA server objects, event channels used, etc.
⟨/services⟩
⟨init⟩
        // Host code for task specific initialization
⟨/init⟩
⟨destroy⟩
        // Host code for task specific cleanup
        // CORBA cleanup handled automatically
⟨/destroy⟩
⟨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⟩
```

(a) TSL tags and partial schematic for a TP specification.

```
⟨fsm⟩
⟨statename =sname⟩
⟨action⟩
        // Executed whenever TP enters this state
⟨/action⟩
        // State specific reactions to events
⟨reaction event = "event_name"⟩ . . . ⟨/reaction⟩
                        .
                        .
                        .
⟨reaction event = "event_name"⟩ . . . ⟨/reaction⟩
⟨/state⟩

//More state specifications . . .

        // Global reactions to events
⟨reaction⟩ . . . ⟨/reaction⟩
                        .
                        .
                        .
⟨reaction⟩ . . . ⟨/reaction⟩
⟨/fsm⟩
```

(b) TSL tags and partial schematic for an fsm specification

Fig. 7. TP Specifications with TSL tags.

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.

4.2 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 in the shape of finite state machines and the use of a declarative flight command language (FCL). Events from the flight controller (passed via event channels) create a partial view of the state of the UAV. Based on this, and its own state, a TP can dynamically generate appropriate flight commands, which are sent back to the control system.

Suppose the UAV has been given the task of finding and tracking a vehicle with a particular signature, believed to be in a certain region of Revinge. A TP for such a mission consists of several subtasks, each carried out by the top-level TP invoking other TPs: The first is NavToPt (depicted in Figure 8), which navigates the UAV safely to a waypoint in the region of interest (ROI).

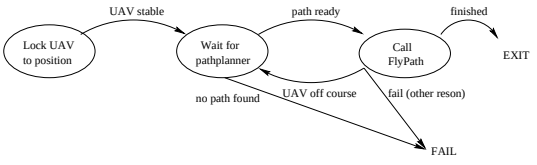

Fig. 8. The NavToPt automaton

To do so, NavToPt makes use of both deliberative and control services. First, it calls a path planning service [25] which provides a (segmented) path from the UAVs current position to the waypoint (provided such a path can be found). Then, it invokes a FlyPath TP

which takes care of passing the path segments, target velocity and other parameters to the dynamic path following flight mode in the control system. During flight, FlyPath receives events from the control system (via event channels) and responds appropriately, for example issuing a command to enter hovering mode when the end of the last path segment is reached. It also generates events reporting on the progress of the flight, which are in turn monitored by NavToPt.

Once the UAV has arrived at the goal point, the parent TP configures and starts image processing services [18] to attempt identification of vehicles in the area. If a vehicle matching the initial signature is found, the parent TP starts a new TP FlyTo (see Figure 9) which uses proportional navigation mode [27], one of the existing RTF-modes, to position the UAV relative to the vehicle. Since the vehicle is probably moving, FlyTo instances will be continually terminated and restarted with new parameters. Each instance of FlyTo, while running, generates a stream of flight commands passed to the PFC system, where they are handled by the PN flight mode functional unit, 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.

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 10 shows a number of representative flight commands used for the proportional navigation flight control mode.

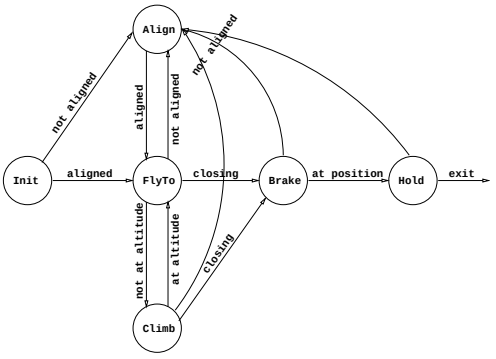

Fig. 9. The FlyTo automaton

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 dynamic path 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(), SlowDown()	Land(), FlyAtAltitude()
Slow(), Brake()	ClimbTo()
FlyToFlyObject(), FlyWithVelocity()	**Yaw Channel**
FlyTowards(), LockOnFlyObject()	FlyWithYaw(), FlyWithYawOfYourself()
LockOnPosition(), LockOnLookObject()	FlyCleanly()

Fig. 10. Representative Flight Commands for Proportional Navigation.

4.3 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 object and can be spawned or terminated using the Witas::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. In a similar manner, deliberative functionalities can be modularly added and packaged as CORBA objects. These may include legacy code. For example, the chronicle recognition software used is based on IXTET [11][8] and is wrapped directly as a CORBA object . These objects may physically be distributed between the UAV itself and various ground stations. This is useful if the computational resources required for a service exceed those provided by the UAV platform.

5 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 [21]. 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[10]. Rather than list publications, we refer the reader to the following (non-exhaustive) list of websites for information about interesting university and institute UAV activities: Georgia Tech[10], M.I.T[17], Carnegie-Mellon[2], U. of C., Berkeley[29], Stanford University[28], ONERA RESSAC [22].

References

1. R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.
2. Carnegie Mellon University. `http://www.ri.cmu.edu/projects/project_93.html`.
3. G. Conte, S. Duranti, and T. Merz. Dynamic 3D path following for an autonomous helicopter. In *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

[8] In fact, we use a new implementation, CRS, developed by C. Dousson at France Telecom (`http://crs.elibel.tm.fr/en/`).

4. P. Doherty, G. Granlund, K. Kuchcinski, K. Nordberg, E. Sandewall, E. Skarman, and J. Wiklund. The WITAS unmanned aerial vehicle project. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 747–755, 2000. http://www.liu.se/ext/witas.
5. P. Doherty, J. Kachniarz, and A. Szałas. Using contextually closed queries for local closed-world reasoning in rough knowledge databases. In *[23]*, 2003.
6. P. Doherty and J. Kvarnström. TALplanner: A temporal logic based planner. In *AI Magazine*, volume 22, pages 95–102. AAAI Press, 2001.
7. P. Doherty, W. Łukaszewicz, A. Skowron, and A. Szałas. Combining rough and crisp knowledge in deductive databases. In *[23]*, 2003.
8. P. Doherty, W. Łukaszewicz, and A. Szałas. Approximative query techniques for agents using heterogenous ontologies. In *Int'l Conference on Principles of Knowledge Representation and Reasoning (KR-04)*, 2004.
9. E. Gat. Three-layer architectures. In D. Kortenkamp, R. P. Bonassao, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, chapter 8, pages 195–210. MIT Press, 1998.
10. Georgia Tech University. http://controls.ae.gatech.edu/uavrf/.
11. M. Ghallab. On chronicles: Representation, on-line recognition and learning. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR-96)*, 1996.
12. D. Harel. Statecharts: A viusal formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
13. F. Heintz and P. Doherty. Dyknow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, 2004. Accepted for publication.
14. F. Heintz and P. Doherty. Managing dynamic object structures usin hypothesis generation and validation. In *Proceedings of 2004 AAAI Workshop on Anchoring Symbols to Sensor Data*, 2004. National Conference on Artificial Intelligence.
15. D. Kortenkamp, R. P. Bonassao, and R. Murphy, editors. *Artificial Intelligence and Mobile Robots*. AAAI Press/MIT Press, 1998.
16. T. Merz, S. Duranti, and G. Conte. Autonomous landing of an unmanned helicopter based on vision and inertial sensing. In *International Symposium on Experimental Robotics (ISER-2004)*, 2004.
17. M.I.T. http://gewurtz.mit.edu/research.htm.
18. K. Nordberg, P. Doherty, P-E. Forssen, J. Wiklund, and P. Andersson. A flexible runtime system for image processing in a distributed computational environment for an unmanned aerial vehicle. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004. To appear.
19. P. Nyblom. A language translator for robotic task procedure specifications. Master's thesis, Linköping university, Sweden, 2003. LITH-IDA-EX-03/034-SE.
20. Object Computing, Inc. *TAO Developer's Guide, Version 1.1a*, 2000. See also http://www.cs.wustl.edu/~schmidt/TAO.html.
21. USA Office of the Secretary of Defence. Unmanned aerial vehicles roadmap, 2002-2025, 2001. http://www.acq.osd.mil/uav_roadmap.pdf.
22. ONERA RESSAC. http://www.cert.fr/dcsd/RESSAC/.
23. S.K. Pal, L. Polkowski, and A. Skowron, editors. *Rough-Neuro Computing: Techniques for Computing with Words*. Springer–Verlag, Heidelberg, 2003.
24. P-O. Pettersson and P. Doherty. Probabilistic roadmap based path planning for autonomous unmanned aerial vehicles. In *Proceedings of the Workshop on Connecting Planning and Theroy with Practice*, 2004. 14th Int'l Conf. on Automated Planning and Scheduling, ICAPS'2004.
25. Per-Olof Pettersson. Helicopter path planning using probabilistic roadmaps. Master's thesis, Linköping university, Sweden, 2003. LITH-IDA-EX-02-56.
26. E. Sandewall, P. Doherty, O. Lemon, and S. Peters. Words at the right time: Real-time dialogues with the WITAS unmanned aerial vehicle. In *Proc. of the 26th German Conference on Artificial Intelligence*, 2003.
27. E. Skarman. On proportional navigation. Technical note, 2003.
28. Stanford University. http://sun-valley.stanford.edu/users/heli/.
29. University of California, Berkeley. http://robotics.eecs.berkeley.edu/bear/.