Preliminary Report: Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle

Mariusz Wzorek and **Patrick Doherty**
Department of Computer and Information Science
Linköping University, SE-58183 Linköping, Sweden
{marwz,patdo}@ida.liu.se

Abstract

In this paper, we present a motion planning framework for a fully deployed autonomous unmanned aerial vehicle which integrates two sample-based motion planning techniques, Probabilistic Roadmaps and Rapidly Exploring Random Trees. Additionally, we incorporate dynamic reconfigurability into the framework by integrating the motion planners with the control kernel of the UAV in a novel manner with little modification to the original algorithms. The framework has been verified through simulation and in actual flight. Empirical results show that these techniques used with such a framework offer a surprisingly efficient method for dynamically reconfiguring a motion plan based on unforeseen contingencies which may arise during the execution of a plan.

Introduction

The use of Unmanned Aerial Vehicles (UAVs) which can operate autonomously in dynamic and complex operational environments is becoming increasingly more common. While the application domains in which they are currently used are still predominantly military in nature, in the future we can expect widespread usage in the civil and commercial sectors. In order to insert such vehicles into commercial airspace, it is inherently important that these vehicles can generate collision-free motion plans and also be able to modify such plans during their execution in order to deal with contingencies which arise during the course of operation. Motion planners capable of dynamic reconfiguration will be an essential functionality in any high-level autonomous UAV system.

The motion planning problem, that of generating a collision-free path from an initial to goal waypoint, is inherently intractable for vehicles with many degrees of freedom. Recently, a number of sample-based motion planning techniques (Kavraki *et al.* 1996; Kuffner & LaValle 2000) have been proposed which tradeoff completeness in the planning algorithm for tractability and efficiency in most cases. The purpose of this paper is to show how one can incorporate dynamic reconfigurability in such motion planners on a deployed and fully operational UAV by integrating the motion planner with the control kernel of the UAV in a novel manner with little modification of the original algorithms.

Integrating both high- and low-end functionality seamlessly in autonomous architectures is currently one of the major open problems in robotics research. UAV platforms offer an especially difficult challenge in comparison with ground robotic systems due to the often tight time constraints present in the plan generation, execution and reconfiguration stages in many complex mission scenarios.

It is the intent of this paper to show how one can leverage sample-based motion planning techniques in this respect, first by describing how such integration would be done and then empirically testing the results in a fully deployed system. The techniques and solutions described are generic in nature and suitable for platforms other than the one used in this experimentation. An important point to note is that to our knowledge we are the first to use these sample-based motion planning techniques with fully deployed UAVs.

The structure of the paper is as follows. First we give an overview of the integrated hardware and software platform used in our UAV. Then an overview of two sample-based motion planning techniques, Probabilistic Roadmaps and Rapidly Exploring Random Trees is provided. Later we describe the basic architecture for integrating motion planners with the UAV control kernel. We explain the path execution mechanism in the static environments and describe the dynamic path replanning scheme in addition to providing timing constraints. At the end the empirical results from the preliminary experiments with the deployed system are presented. We then conclude with related work and a summary.

Figure 1: The WITAS RMAX Helicopter

WITAS System Overview

The hardware platform

The WITAS [1] UAV platform (Doherty *et al.* 2004) is a slightly modified Yamaha RMAX helicopter (Fig. 1). It has a total length of 3.6 m (including main rotor) and is powered by a 21 hp two-stroke engine with a maximum takeoff weight of 95 kg. The helicopter has a built-in attitude sensor (YAS) and an attitude control system (YACS). The hardware platform developed during the WITAS UAV project is integrated with the Yamaha platform as shown in Fig. 2. It contains three PC104 embedded computers.

The primary flight control (PFC) system runs on a PIII (700Mhz), and includes a wireless Ethernet bridge, a RTK GPS receiver, and several additional sensors including a barometric altitude sensor. The PFC is connected to the YAS and YACS, an image processing computer and a computer for deliberative capabilities.

The image processing (IP) system runs on the second PC104 embedded computer (PIII 700MHz), and includes a color CCD camera mounted on a pan/tilt unit, a video transmitter and a recorder (miniDV).

The deliberative/reactive (D/R) system runs on the third PC104 embedded computer (Pentium-M 1.4GHz) and executes all high-end autonomous functionality. Network communication between computers is physically realized with serial line RS232C and Ethernet. Ethernet is mainly used for CORBA applications (see below), remote login and file transfer while serial lines are used for hard real-time networking.

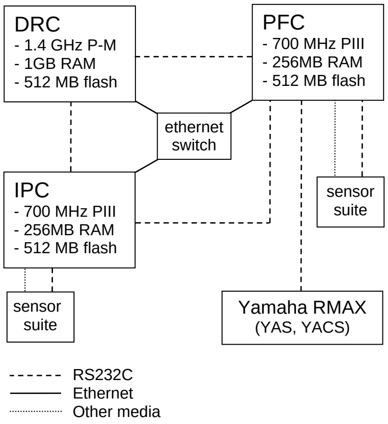

Figure 2: On-Board Hardware Schematic

The Software platform

A hybrid deliberative/reactive software architecture has been developed for the UAV and has also been used in a ground robot. Conceptually, it is a layered system with deliberative, reactive and control components. The architecture has a *reactive concentric* flavor where reactive task procedures use services provided by both deliberative and control components in a highly distributed and concurrent manner.

The software implementation is based on CORBA (Common Object Request Broker Architecture), which is often used as middleware for object-based distributed systems. It enables different objects or components to communicate with each other regardless of the programming languages in which they are written, their location on different processors or the operating systems they running on. A component can act as a client, a server or as both. The functional interfaces to components are specified via the use of IDL (Interface Definition Language). The majority of the functionalities which are part of the architecture can be viewed as CORBA objects or collections of objects, where the communication infrastructure is provided by CORBA facilities and other services such as real-time and standard 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.

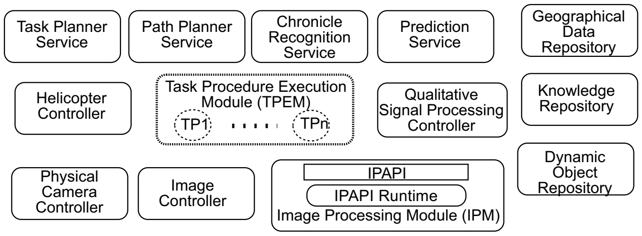

Figure 3: Some deliberative, reactive and control services

Fig. 3 presents some (not all) of the high-level services used in the WITAS UAV system. Those services run on the D/R computer and interact with the control system.

The control system is a hybrid distributed system that runs primarily on the PFC computer in a real-time environment (Merz 2004) constructed especially to integrate seamlessly with the rest of the architecture. Hierarchical concurrent state machines (HCSMs) are used to represent system states. 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 have developed and tested several autonomous flight control modes: take-off, landing via visual navigation, hovering, dynamic path following, and reactive flight modes for tracking and interception. A CORBA interface is setup on top of the control system kernel so high-level components can issue commands to initiate and sequentialize different flight modes. Helicopter states and events from the control system are in turn sent to the high-level system.

---

[1] WITAS is an acronym for the Wallenberg Information Technology and Autonomous Systems Lab which hosted a long term UAV research project (1997-2004).

The Path Planning algorithms

In this section, we provide a brief overview of the sample-based path planning techniques used in the experiments. The problem of finding optimal paths between two configurations in a high-dimensional configuration space such as a helicopter is intractable in general. Sample-based approaches such as probabilistic roadmaps (PRM) or rapidly exploring random trees (RRT) often make the path planning problem solvable in practice by sacrificing completeness and optimality.

Probabilistic Roadmaps

The standard probabilistic roadmap (PRM) algorithm (Kavraki *et al.* 1996) works in two phases, one off-line and the other on-line. In the off-line phase a roadmap is generated using a 3D world model. Configurations are randomly generated and checked for collisions with the model. A local path planner is then used to connect collision-free configurations taking into account kinematic and dynamic constraints of the helicopter. Paths between two configurations are also checked for collisions. In the on-line or querying phase, initial and goal configurations are provided and an attempt is made to connect each configuration to the previously generated roadmap using the local path planner. A graph search algorithm such as A* is then used to find a path from the initial to the goal configuration in the augmented roadmap.

Fig. 4 provides a schema of the PRM path planner used in the WITAS system. The planner uses an OBBTree-algorithm for collision checking and an A $*$ algorithm for graph search. Here one can optimize for shortest path, minimal fuel usage, etc. The following extensions have been made with respect to the standard version of PRM algorithm in order to adapt the approach to our UAV platform.

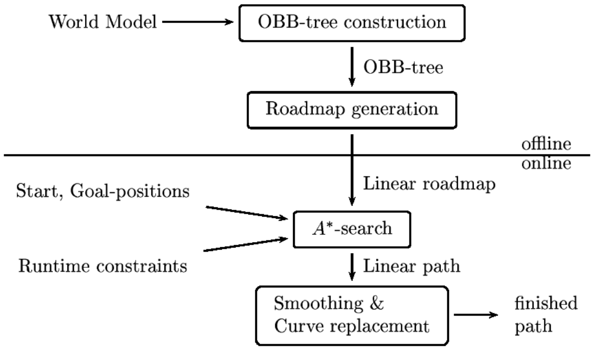

Figure 4: PRM path plan generation

- *Multi-level roadmap planning*
  The standard probabilistic roadmap algorithm is formulated for fully controllable systems only. This assumption is true for a helicopter flying at low speed with the capability to stop and hover at each waypoint. However, when the speed is increased the helicopter is no longer able to negotiate turns of a smaller radius, which imposes demands on the planner similar to non-holonomic constraints for car-like robots. In this case, linear paths are first used to connect configurations in the graph and at a later stage these are replaced with cubic curves when possible. These are required for smooth high speed flight. If it is not possible to replace a linear path segment with a cubic curve then the helicopter has to slow down and switch to hovering mode at the connecting waypoint before continuing. From our experience, this rarely happens.
- *Runtime Constraint Handling*
  Our motion planner has been extended to deal with different types of constraints at runtime not available during roadmap construction. Such constraints can be introduced at the time of a query for a path plan. Some examples of runtime constraints currently implemented include maximum and minimum altitude, adding forbidden regions (no-fly zones) and placing limits on the ascent-/descent-rate. Such constraints are dealt with during the A* search phase.

The mean planning time in the current implementation is below 1000 ms and the use of runtime constraints do not noticeably influence the mean. For a more detailed description of the modified PRM planner, see (Pettersson & Doherty 2004; Pettersson 2005).

Rapidly Exploring Random Trees

The use of rapidly exploring random trees (RRT) provides an efficient motion planning algorithm that constructs a roadmap online rather than offline. The algorithm (Kuffner & LaValle 2000) generates two trees rooted in the start and end configurations by exploring the configuration space randomly in both directions. While the trees are being generated, an attempt is made at specific intervals to connect them to create one roadmap. After the roadmap is created, the remaining steps in the algorithm are the same as with PRMs. In comparison with the PRM planner, the mean planning time with RRT is also below 1000 ms, but in this case, the success rate is much lower and the generated plans are not optimal which may sometimes cause anomalous detours (Pettersson 2005).

Path execution mechanism

The standard path execution scheme in our architecture for static operational environments is depicted in Fig. 5. A UAV mission is specified via a task procedure (TP) in the reactive layer of our architecture, (perhaps after calling a task-based

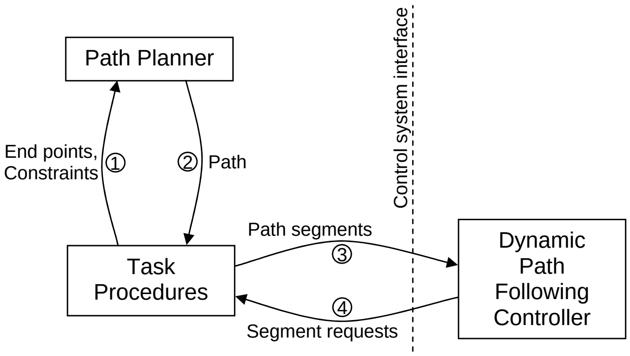

Figure 5: Plan execution scheme

planner). A TP is a high-level procedural execution component which provides a computational mechanism for achieving different robotic behaviors. For the purposes of this paper, it can be viewed as an augmented state machine.

For the case of flying to a waypoint, an instance of a navigation TP is created. First it calls the path planner service (step 1) with the following parameters: initial position, goal position, desired velocity and additional constraints.

If successful, the path planner (step 2) generates a segmented cubic polynomial curve. Each segment is defined by start and end points, start and end directions, target velocity and end velocity. The TP sends the first segment (step 3) of the trajectory via the control system interface and waits for the *Request Segment* event that is generated by the controller. At the control level, the path is executed using a dynamic path following controller (Conte, Duranti, & Merz 2004) which is a reference controller that can follow cubic splines. When a *Request Segment* event arrives (step 4) the TP sends the next segment. This procedure is repeated (step 3-4) until the last segment is sent. However, because the high-level system is not implemented in hard real-time it may happen that the next segment does not arrive to the control kernel on time. In this case, the controller has a timeout limit after which it goes into safety braking mode in order to stop and hover at the end of the current segment. The timeout is determined by a velocity profile, current position and current velocity.

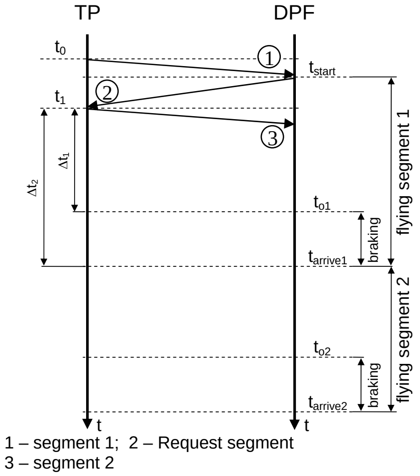

Figure 6: Execution timeline for trajectory consisted of 2 segments

Fig. 6 depicts a timeline plot of the execution of a trajectory (2 segments). At time t_0, a TP sends the first segment of the path to the DPF controller and waits for a *Request segment* event which arrives immediately (t_1) after the helicopter starts to fly (t_{start}). Typical time values for receiving a *Request segment* event ($t_1 - t_0$) are well below $200ms$. Time t_{o1} is the timeout for the first segment which means that the TP has a Δ_{t_1} time window to send the next segment to the DPF controller before it initiates the safety braking procedure. If the segment is sent after t_{o1}, the helicopter will start braking. In the current implementation it is not allowed to send segments after the timeout. This will be changed in a future implementation. In practice the Δ_{t_1} time window is large enough to reconfigure the path using the standard path planner. The updated segments are then sent to the DFP controller transparently.

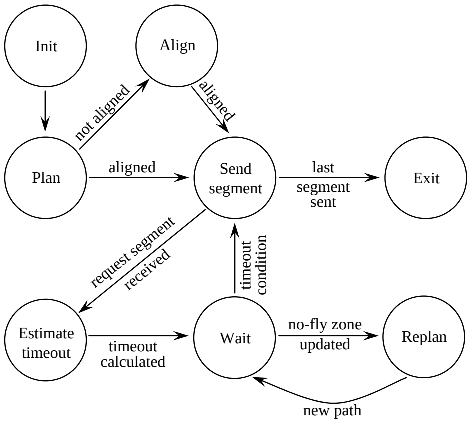

Figure 7: The reconfigurable path mode automaton

The augmented state machine associated with the TP used for reconfigurable path planning is depicted in Fig. 7. The TP takes a start and end point and target velocity as input. The TP then calls a path planning service (*Plan* state) which returns an initial path. If the helicopter is not aligned with the direction of the flight, a command to align is sent to the controller (*Align* state).The TP then sends the first segment of the generated path to the DPF controller (*Send segment* state) and estimates a timeout for the current segment based on the velocity profile (*Estimate timeout* state). Based on the segment timeout and system latency, a condition is calculated for sending the next segment. If new information about newly added or deleted forbidden regions (no-fly zone updated) arrives, the TP reconfigures the path by calling the path planner again. The TP terminates when the last segment is sent. There are several different policies that can be used during the reconfiguration step (Fig. 8):

Policy 1

Reconfiguration is done from the next waypoint (start point of the next segment) to the end point. This implies longer planning times and eventual replacement of collision-free segments.

Policy 2

Segments up to the colliding one are left intact and reconfiguration is done from the last collision-free waypoint to the end point.

Policy 3

Replanning is done only for colliding segments. The helicopter will stay as close to the initial path as possible.

Note that each of these policies progressively re-uses more of the plan originally generated, thus cutting down on planning times. We are currently experimenting with all three policies.

Fig. 9 shows the minimum distance required to detect a

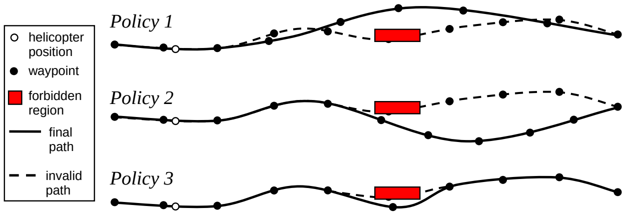

Figure 8: Replanning policies.

forbidden region (obstacle) in order to reconfigure the path. This is the worst case under the assumption of constant velocity along the path and when second policy is applied. Acceleration and deceleration is equal to 1.6 m/s^2. The minimum time for one reconfiguration including system latency is below 1200 ms.

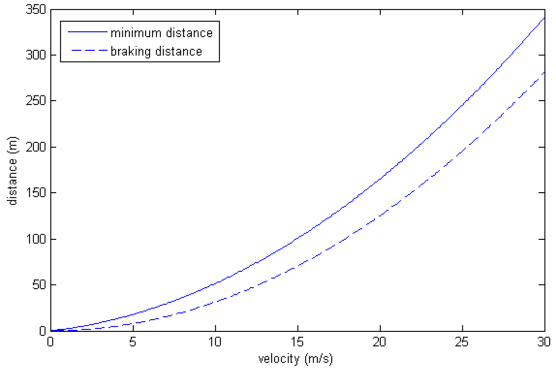

Figure 9: Critical distance for detecting an obstacle

Experimental results

In our experiments we have used both the PRM and the RRT planner with a TP that implements the first policy. Forbidden regions were randomly added by the ground operator during the flight.

The results of the experiments are presented in Table 1. The table shows typical values of parameters related to the execution and the planning phases.

The number of segments is taken from the final path. Observe that Δ_{t_1} is generally greater than four times the amount of time required to generate full plans using either the PRM or RRT planners.

Fig. 10 shows sample paths generated during one of the experiments in which four no-fly zones were added incrementally during plan execution where each of the newly generated plans is reconfigured as new no-fly zones are added.

Related work

Finding collision-free paths in dynamically changing environments is an open research problem in the motion planning community. As important as the problem is there are a limited number of contributions that address issues related to changing environments and even less in the context of UAVs. Results using probabilistic roadmap based planners focus mainly on the mobile manipulation domain (e.g. Jaillet & Siméon, 2004; Leven & Hutchinson, 2000). An example of planner that samples *state*×*time* space in order to deal with kinematic and dynamic constrains on robots, as well as moving obstacles is presented by Hsu *et al.* (2000). Some work has also been done with the elastic framework (Brock & Khatib, 1998; 1999; 2000) and with decomposition-based methods (Brock & Kavraki 2001). In the UAV domain, we believe we are the first to apply sample-based motion planning approaches.

Table 1: Results of the experiments

Planner	path length (m)	number of segments	added forbidden regions	min. segment length(m)	max. replanning time (ms)	min. Δ_{t_1} (ms)
PRM	422.52	6	4	34.87	519	3518
	420.55	6	4	40.95	486	2898
	432.17	6	4	62.50	568	3673
	427.94	6	5	53.15	524	3285
	536.98	7	5	50.22	631	3158
	472.40	7	6	45.25	603	2918
	539.18	8	6	53.24	728	3153
RRT	500.12	7	4	26.68	315	2862
	422.58	5	4	74.07	438	4079
	392.89	5	5	61.11	441	3625
	565.06	8	5	26.76	521	3648
	503.42	6	5	65.07	954	3773
	464.96	6	5	28.61	595	3866
	491.42	8	6	20.40	326	1803

Conclusions

We have presented a distributed software architecture for UAVs and considered how one can successfully integrate sample-based motion planning techniques in a robust and efficient manner. We have also shown how these techniques can be used to deal with random contingencies such as new no-fly zones during plan execution. This has been done by analyzing the course of plan execution and extracting upper bounds on the time that can be spent generating new plans or repairing old plans by calling a PRM or RRT planner. Experimental results show the feasibility of using these techniques in the UAV domain, but similar analyses and frameworks could in fact be used for other robotic platforms. The planning framework has been tested and used in a fully deployed autonomous UAV system.

Acknowledgements

This research has been supported in part by the WITAS UAV project grant under the Wallenberg Foundation, Sweden and an NFFP03 research grant (COMPAS).

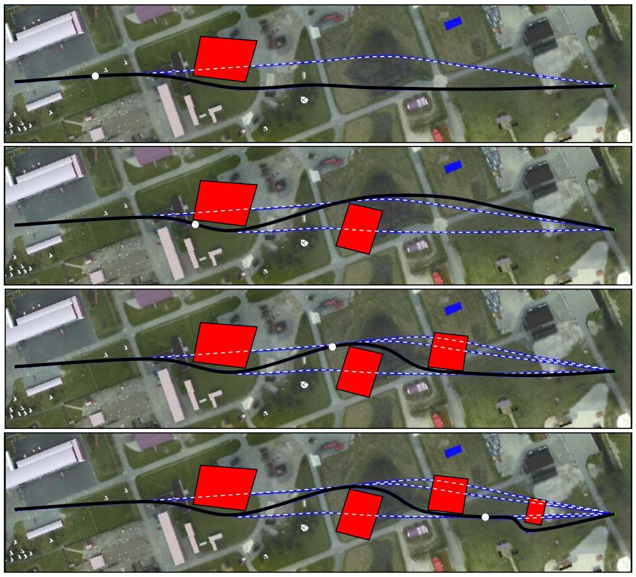

Figure 10: Paths generated during experimental flight. Solid black line - updated path (white dot - helicopter position); white dashed line - invalid path; polygon box - forbidden region.

References

Brock, O., and Kavraki, L. 2001. Decomposition-based Motion Planning: A Framework for Real-time Motion Planning in High-dimensional Configuration Spaces. In *Proc. of the Int. Conf. on Robotics and Automation*.

Brock, O., and Khatib, O. 1998. Mobile Manipulation: Collision-Free Path Modification and Motion Coordination. In *Proc. of the Int. Conf. on Computational Engineering in Systems Applications*, volume 4, 839–845.

Brock, O., and Khatib, O. 1999. Elastic Strips: A Framework for Integrated Planning and Execution. In *Proc. Int. Symp. on Experimental Robotics*.

Brock, O., and Khatib, O. 2000. Real Time Replanning in High-Dimensional Configuration Spaces Using Sets of Homotopic Paths. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 550–555.

Conte, G.; Duranti, S.; and Merz, T. 2004. Dynamic 3D Path Following for an Autonomous Helicopter. In *Proc. of the IFAC Symp. on Intelligent Autonomous Vehicles*.

Doherty, P.; Haslum, P.; Heintz, F.; Merz, T.; Persson, T.; and Wingman, B. 2004. A Distributed Architecture for Autonomous Unmanned Aerial Vehicle Experimentation. In *Proc. of the Int. Symp. on Distributed Autonomous Robotic Systems*, 221–230.

Hsu, D.; Kindel, R.; Latombe, J. C.; and Rock, S. 2000. Randomized Kinodynamic Motion Planning with Moving Obstacles. In *Proc. of the Workshop on Algorithmic Foundations of Robotics (WAFR00)*.

Jaillet, L., and Siméon, T. 2004. A PRM-based Motion Planner for Dynamically Changing Environments. In *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems*.

Kavraki, L. E.; Švestka, P.; Latombe, J.; and Overmars, M. H. 1996. Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces. *Proc. of the IEEE Transactions on Robotics and Automation* 12(4):566–580.

Kuffner, J. J., and LaValle, S. M. 2000. RRT-connect: An Efficient Approach to Single-Query Path Planning. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 995–1001.

Leven, P., and Hutchinson, S. 2000. Toward Real-Time Path Planning in Changing Environments. In *Proc. of the Workshop on Algorithmic Foundations of Robotics*.

Merz, T. 2004. Building a System for Autonomous Aerial Robotics Research. In *Proc. of the IFAC Symp. on Intelligent Autonomous Vehicles*.

Pettersson, P.-O., and Doherty, P. 2004. Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Aerial Vehicle. In *Proc. of the ICAPS-04 Workshop on Connecting Planning Theory with Practice*.

Pettersson, P.-O. 2005. Using Randomized Algorithms for Helicopter Path Planning. *Lic. Thesis Linköping University.* To appear.