# From Motion Planning to Control - A Navigation Framework for an Autonomous Unmanned Aerial Vehicle

**Mariusz Wzorek, Gianpaolo Conte, Piotr Rudol**
**Torsten Merz, Simone Duranti, Patrick Doherty**
**Department of Computer and Information Science**
**Linköping University, SE-58183 Linköping, Sweden**
**{marwz,giaco,pioru,g-torme,simdu,patdo}@ida.liu.se**

## ABSTRACT

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. In this paper, we present a fully deployed autonomous unmanned aerial vehicle, based on a Yamaha RMAX helicopter, which is capable of navigation in urban environments. We describe a motion planning framework which integrates two sample-based motion planning techniques, Probabilistic Roadmaps and Rapidly Exploring Random Trees together with a path following controller that is used during path execution. Integrating deliberative services, such as planners, seamlessly with control components in autonomous architectures is currently one of the major open problems in robotics research. We show how the integration between the motion planning framework and the control kernel is done in our system.

Additionally, we incorporate a dynamic path reconfigurability scheme. It offers a surprisingly efficient method for dynamic replanning of a motion plan based on unforeseen contingencies which may arise during the execution of a plan. Those contingencies can be inserted via ground operator/UAV interaction to dynamically change UAV flight paths on the fly. The system has been verified through simulation and in actual flight. We present empirical results of the performance of the framework and the path following controller.

## BIOGRAPHY

*Patrick Doherty* is a Professor at the Department of Computer and Information Science (IDA), Linköping University (LiU), Sweden. He is director of the Artificial Intelligence and Integrated Computer Systems Division at IDA and his research interests are in the area of knowledge representation, automated planning, autonomous systems, approximate reasoning and UAV technologies.
*Mariusz Wzorek* (MSc) is a graduate student at LiU. Research focus: automated planning techniques, autonomous unmanned systems and robotics.
*Gianpaolo Conte* (MSc) is a graduate student at LiU. Research focus: control and sensor fusion related problems.
*Piotr Rudol* (MSc) is a graduate student at LiU. Research focus: non-GPS navigation, mapping and obstacle avoidance for UAVs.
*Torsten Merz* (PhD) has been involved in the WITAS project at LiU for many years. Since 2006, he is working at the Autonomous Systems Lab at CSIRO (Australia) leading the development of an unmanned aerial vehicle for power line inspection.
*Simone Duranti* (MSc) is working at Saab Aerosystems and LiU. He has been involved in the WITAS project at LiU for many years.

# 1 Introduction

Navigating in environments cluttered with obstacles in the vicinity of building structures requires path planning algorithms which deliver collision-free paths, accurate controllers able to execute such paths even in the presence of inhospitable weather conditions (e.g. wind gusts) and a reliable mechanism that coordinates the two.

This paper describes an approach to combining path planning techniques with a path execution mechanism (including a robust 3D path following control mode) in a distributed software architecture used in a fully deployed rotor-based Unmanned Aerial Vehicle (UAV). Details of many of the software components used in the distributed architecture are provided. An emphasis is placed on the components responsible for path execution. The approach allows for interaction of a path planning algorithm with a path following control mode and copes with their different timing characteristics and distributed communication. It also includes a safety mechanism which is necessary for operating UAVs in urban environments. For the experiments we present in this paper, we assume a predominantly static environment which is described by a 3D model. An onboard geographic information system (GIS) is used to supply information about building structures, vegetation, etc. Certain types of dynamic changes in the environment are handled by the use of no-fly zones or pop-up zones which can be added or removed on the fly during the course of a mission.

Our hardware/software framework incorporates software distribution technologies for a number of reasons. Firstly, existing commercial off-the-shelf (COTS) hardware suitable enough for airborne systems, does not yet have sufficient computational power and storage space to encompass all the necessary software components needed to achieve sophisticated mission scenarios autonomously. Additionally, in order to use third-party software without compromising flight safety, it is necessary to separate software components that can crash the operating system from software that is crucial for the UAV flight operation. Another reason for using a distributed solution is to take advantage of additional resources which may be found on the Internet.

One of the long term goals which has guided our research is the idea of *push-button* missions where the ground operator supplies mission tasks to a UAV at a very high-level of abstraction and the UAV system does most of the work ranging from planning to actual execution of the mission.

The Autonomous UAV Technologies Laboratory [1] at Linköping University, Sweden, has been developing fully autonomous rotor-based UAV systems in the mini- and micro-UAV class. Our current system design is the result of an evolutionary process based on many years of developing, testing and maintaining sophisticated UAV systems. In particular, we have used the Yamaha RMAX helicopter platform and developed a number of micro air vehicles from scratch.

Much effort has also gone into the development of useful ground control station interfaces which encourage the idea of push-button missions, letting the system itself plan and execute complex missions with as little effort as possible required from the ground operator other than stating mission goals at a high-level of abstraction and monitoring the execution of the ensuing mission. The mission scenarios we use are generic in nature and may be instantiated relative to different applications. For example, the functionality required for the monitoring/surveillance mission described below can be modified slightly and used in mission scenarios such as power line inspection.

An example of such a push-button mission that has been used as an application scenario in our research is a combined monitoring/surveillance and photogrammetry mission out in the field in an urban area with the goal of investigating facades of building structures and gathering both video sequences and photographs of building facades. For this experiment, we have used a Yamaha RMAX helicopter system as a platform. Let us assume the operational environment is in an urban area with a complex configuration of building and road structures. A number of these physical structures are of interest since one has previously observed suspicious behavior and suspects the possibility of terrorist activity. The goal of the mission is to investigate a number of these buildings and acquire video and photos from each of the building's facades. It is assumed the UAV has a 3D model of the area and a GIS with building and road structure information on-line.

The ground operator would simply mark building structures of interest on a map display and press a button to generate a complete multi-segment mission that flies to each building, moves to waypoints to view each facade, positions the camera accordingly and begins to relay video and/or photographs. The motion plans generated are also guaranteed to be collision-free from

[1] `www.ida.liu.se/~patdo/auttek/`

static obstacles. If the ground operator is satisfied with the generated mission, he or she simply clicks a confirm button and the mission begins. During the mission, the ground operator has the possibility of suspending the mission to take a closer look at interesting facades of buildings, perhaps taking a closer look into windows or openings and then continuing the mission. This mission has been successfully executed robustly and repeatedly from take-off to landing using the RMAX. The plan generation and execution mechanism described in this paper is an integral part of such a complex mission.

### 1.1 Related Work

Many universities (20; 22; 1; 19; 9; 21; 23) have been and continue to do research with autonomous helicopter systems. Most of the research has focussed on low-level control of such systems with less emphasis on high-autonomy as in our case. The research area itself is highly multidisciplinary and requires competences in many areas such as control system design, computer science, artificial intelligence, avionics and electronics. We briefly mention a number of interesting university research projects representative of the type of research being pursued.

The Autonomous Helicopter Project at Carnegie Mellon University has done a great deal of research on helicopter modeling (16) and helicopter control (2), developing and testing robust flight control for full-envelope flight.

The School of Aerospace Engineering at Georgia Tech has developed an adaptive control system using neural networks (10) and has demonstrated the ability for high speed flight and operation with aggressive flight maneuvers using the Yamaha RMAX helicopter.

The motion planning problem for helicopters has been investigated by (6). Here the problem of the operation of an autonomous vehicle in a dynamic environment has been an issue of research and a method to reduce the computational complexity of the problem has been proposed based on the quantization of the system dynamics leading to a control architecture based on a hybrid automaton. The proposed approach has been tested in simulation.

### 1.2 Paper Outline

The paper is structured as follows. In section 2, we describe the hardware architecture developed and used on a Yamaha RMAX helicopter and on-board hardware components. In section 3, we describe the distributed CORBA-based software architecture and a number of software modules used in an integrated manner with the robotic system. Much of this work was completed in the WITAS [2] UAV project. In section 4, we describe the planning techniques used in the UAV system. In section 5, the path following control mode is described in detail and in section 6 we descibe the path execution mechanism. Dynamic path replanning capability is described in section 7 and experimental results of two example missions are presented in section 8. In section 9, we conclude and discuss future work.

## 2 The Hardware Platform

Figure 1: The WITAS RMAX Helicopter in an urban environment

The WITAS UAV platform (4) 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 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 (IPC)

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

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 (DRC) 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.

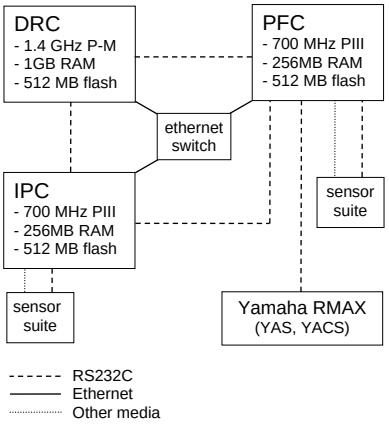


Figure 2: On-board hardware schematic

# 3 The Software Architecture

A hybrid deliberative/reactive software architecture has been developed for the UAV. Conceptually, it is a layered, hierarchical system with deliberative, reactive and control components, although the system can easily support both vertical and horizontal data and control flow. Fig. 3 presents the functional layer structure of the architecture and emphasizes its reactive-concentric nature. Fig. 4 depicts the navigation subsystem and main software components. With respect to timing characteristics, the architecture can be divided into two layers: (a) the hard real-time part, which mostly deals with hardware and control laws (also referred to as the Control Kernel) and (b) the non real-time part, which includes deliberative services of the system (also referred to as the High-level system) [3]. All three

[3]Note that distinction between the Control Kernel and the High-level system is done based mainly on the timing characterisitcs and it does not exclude, for example, placing some deliberative

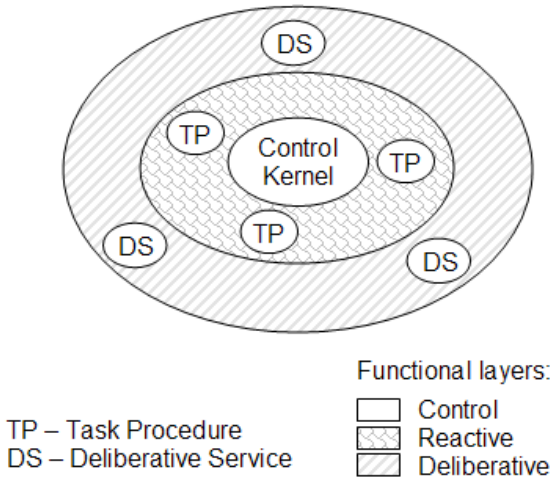


Figure 3: Funcional structure of the architecture

computers in our UAV platform (i.e. PFC, IPC and DRC) have both hard and soft real-time components but the processor time is assigned to them in different proportions. On one extreme, the PFC runs mostly hard real-time tasks with only minimum user space applications (e.g. SSH daemon for remote login). On the other extreme, the DRC uses the real-time part only for device drivers and real-time communication. The majority of processor time is spent on running the deliberative services. Among others, the most important ones from the perspective of this paper are the Path Planner, the Task Procedure Execution Module and the Helicopter Server which encapsulates the Control Kernel (CK) of the UAV system.

The CK is a distributed real-time runtime environment and is used for accessing the hardware, implementing continuous control laws, and control mode switching. Moreover, the CK coordinates the real-time communication between all three on-board computers as well as between CKs of other robotic systems. In our case, we perform multi-platform missions with two identical RMAX helicopter platforms developed in the WITAS UAV project. The CK is implemented using C language. This part of the system uses the Real-Time Application Interface (RTAI) (13) which provides industrial-grade real-time operating system functionality. RTAI is a hard real-time extension to a standard Linux kernel (Debian in our case) and has been developed at the Department of the Aerospace Engineering of Politecnico di Milano.

The real-time performance is achieved by inserting

services (e.g. prediction) in the Control Kernel.

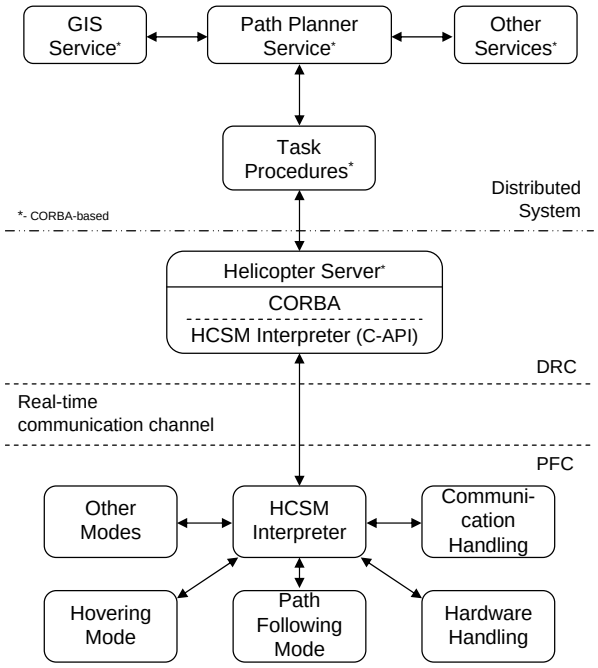


Figure 4: Navigation subsystem and main software components

a module into the Linux kernel space. Since the module takes full control over the processor it is necessary to suspend it in order to let the user space applications run. The standard Linux distribution is a task with lower priority, it runs preemptively and can be interrupted at any time. For that reason a locking mechanism is used when both user- and kernel-space processes communicate though shared memory. It is also important to mention that the CK is self-contained and only the part running on the PFC computer is necessary for maintaining flight capabilities. Such separation enhances safety of the operation of the UAV platform which is especially important in urban environments.

The Control Kernel has a hybrid flavor. Its components contain continuous control laws and mode switching is realized using event-driven hierarchical concurrent state machines (HCSMs). They can be represented as state transition diagrams similar to those of statecharts (8). In our system, tables describing transitions derived from such diagrams are passed to the system in the form of text files and are interpreted by a HCSM Interpreter at run-time in each of the on-board computers. Thanks to its compact and efficient implementation, the interpreter runs in the real-time part of the system as a task with high execution rate. It allows coordinating all *functional units* of the control system from the lowest level hardware components (e.g. device drivers) through control laws (e.g. hovering, path following) and communication to the interface used by the Helicopter Server.

The use of HCSMs also allows implementing complex behaviors consisting of other lower level ones. For instance, landing mode includes control laws steering the helicopter and coordinates camera system/image processing functionalities. When the landing behavior is activated, the CK takes care of searching for a pre-defined pattern with the camera system, feeding the Kalman filter with image processing results which fuses them with the helicopter's inertial measurements. The CK sends appropriate feedback when the landing procedure is finished or it has been aborted. For details see (15).

For achieving best performance, a single non-preemptive real-time task is used which follows a predefined static schedule to run all functional units. Similarly, the real-time communication physically realized using serial lines is statically scheduled with respect to packet sizes and rates of sending. For a detailed description see (14).

The high-level part of the system has reduced timing requirements and is responsible for coordinating the execution of reactive Task Procedures (TPs). This part of the system uses Common Object Request Broker Architecture (CORBA) as its distribution backbone. A TP is a high-level procedural execution component which provides a computational mechanism for achieving different robotic behaviors by using both deliberative and control components in a highly distributed and concurrent manner. The control and sensing components of the system are accessible for TPs through the Helicopter Server which in turn uses an interface provided by the Control Kernel. A TP can initiate one of the autonomous control flight modes available in the UAV (i.e. take off, vision-based landing, hovering, dynamic path following (see section 5) or reactive flight modes for interception and tracking). Additionally, TPs can control the payload of the UAV platform which currently consists of the video camera mounted on a pan-tilt unit. TPs receive data delivered by the PFC and IPC computers i.e. helicopter state and camera system state (including image processing results), respectively. The Helicopter Server on one side uses CORBA to be accessible by TPs or other components of the system, on the other side it communicates through shared memory with the HCSM based interface running in the real-time part of the DRC

software.

The software architecture described is used to achieve missions which require deliberative services such as path planners and control laws such as path following described in detail in sections 4 and 5, respectively. Details of the interaction between the TPs, path planners and the Control Kernel are presented in section 6.

# 4 The Path Planning Algorithms

In this section, we provide a brief overview of the sample-based path planning techniques used in our framework. More detailed descriptions of the two path planners can be found in (18; 17).

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.

## 4.1 Probabilistic Roadmaps

The standard probabilistic roadmap (PRM) algorithm (11) 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. 5 provides a schema of the PRM path planner used in the WITAS UAV system. The planner uses an OBBTree-algorithm (7) 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 the PRM algorithm in order to adapt the approach to our UAV platform.

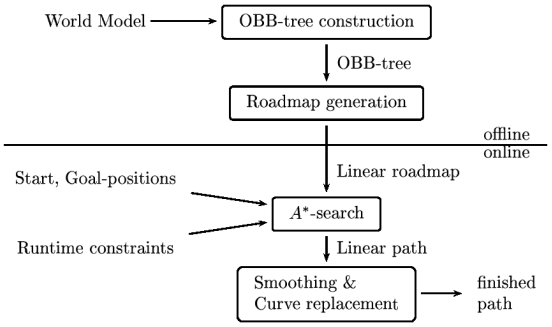


Figure 5: PRM path plan generation

- *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 (for our operational environments) is below 1000 $ms$ and the use of runtime constraints do not noticeably influence the mean.

### 4.2 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 (12) 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 (17).

Results of the path planning algorithms are used by the path following controller described in the following section.

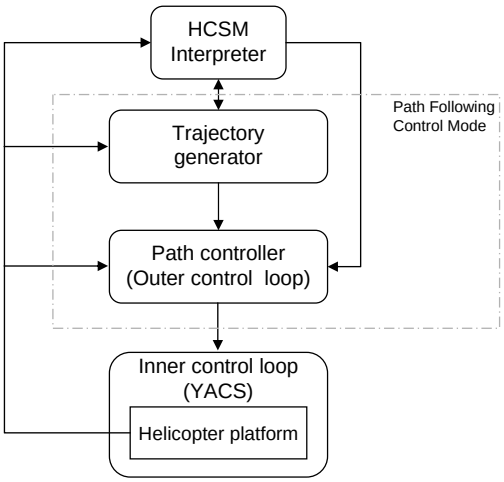


Figure 6: Interaction between the Path Following Control Mode and other components of the Control Kernel

## 5 Path Following Control Mode

In this section, we provide a detailed description of a path following control mode (Fig. 6, the bottom part of Fig. 4) which executes paths provided by the planner introduced in section 4. As described in section 3, the HCSM-based Control Kernel coordinates execution of different control modes available in the UAV system, including the path following control mode.

In the classical approach to the trajectory following problem, a trajectory is generated directly taking into account the dynamic constraints of the system. In our approach, however, we split the problem into two parts. First, the path planner generates a geometrical description of a path and the dynamic constraints are handled later by the path following mode by generating an appropriate velocity profile. In order to navigate safely in urban environments, the path following mode has been designed to minimize the tracking error during a path execution. Because paths generated by the planner are collision-free (relative to the static obstacles present in the model), staying closer to the geometric path assures safer navigation in the environment.

A path is composed of one or more segments (each described by a 3D cubic curve) which are passed sequentially to the control mode. The mode is implemented as a function which takes as input the segment geometry, the desired cruise and final velocities. It is called by the Control Kernel with a frequency of 50Hz and its output consists of four control signals (pitch, roll, yaw and the vertical channel). Additionally, the function returns a set of status flags which are used by the HCSM in control to generate appropriate events, i.e. path segment switching mechanism and safety braking procedure.

When the helicopter reaches the end point of the current segment, the controlling HCSM is informed and an $Arrived$ event is generated. The next call to the path following function is made using parameters describing the next segment to be flown. The process continues until all segments of the path are executed.

The safety braking procedure is activated in case the next segment is not provided by the High-level system (e.g. due to communication failure) in a specific time. This time point is calculated as the minimum distance necessary to stop the helicopter at the end of the current segment without overshooting.

The path tracking error is also available to the controlling HCSM and it can be used to take appropriate actions in case it becomes too large to guarantee safe operation. Such a situation can arise if the wind is too strong for the platform to keep it on the desired path.

The path following control mode is conceptually divided into two parts: (a) the *Trajectory Generator* (TG) which calculates the reference trajectory used by (b) the *Path Controller* (PC) to calculate the control signals. We consider each part in the next two subsections.

## 5.1 Trajectory Generator

A trajectory represents the evolution of a dynamic system in the state-space domain, where the state variables are parameterized in time. In our approach the generated reference trajectory (based on the planner output) depends not on the time but on the position of the helicopter relative to the path. In order to calculate the reference trajectory, a control point on the path which is the closest to the helicopter position must be found. Once this is done, the trajectory parameters can be calculated and used by the path controller (described in the next subsection) as set-points. A feedback method is used to find the control point, similar approach is used in (5).

The reference trajectory in this work is a vector represented by 9 components: $x(s)$, $y(s)$ and $z(s)$ represent the respective East, North positions and altitude of the helicopter relative to an initial point; $v_x(s)$, $v_y(s)$ and $v_z(s)$ the North, East and vertical velocity components; $\phi(s)$, $\theta(s)$ and $\psi(s)$ the roll, pitch and yaw angles; $s$ is the path segment parameter. Details of the calculation of the parameter $s$ through feedback can be found in (3).

The geometric path is composed of several segments represented by a 3D cubic polynomial. The motivation for using this type of curve is given in (17). A 3D geometric segment is represented by the following equation in a vector form:

$$\boldsymbol{P}(s) = \boldsymbol{A}s^3 + \boldsymbol{B}s^2 + \boldsymbol{C}s + \boldsymbol{D}$$

where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{D}$ are 3D vectors defined by the boundary conditions calculated by the path planner and passed to the control mode, $s$=[0,1] is the parameter of the curve and $\boldsymbol{P} = [x, y, z]$. 0nce the parameter $s$ is found the position coordinates $x$, $y$, $z$ can be calculated.

In order to achieve the necessary tracking performance, the path curvature is fed forward in the roll control law. The target roll angle value is calculated according to the following formula:

$$\phi(s) = \frac{V^2}{R_{xy}(s)g}$$

where $V$ is the helicopter speed, $g$ is the gravity acceleration and $R_{xy}(s)$ is the local curvature radius of the path projected on the horizontal plane. The curvature radius of the path is a 3D vector and it is calculated analytically as explained in (3).

The same approach could be applied in order to calculate the target pitch angle $\theta(s)$. However, for the helicopter flight envelope we are interested in, the dynamics of the path curvature in the vertical direction is not very fast. Therefore, the feed forward term is not used for the pitch channel.

The yaw angle $\psi(s)$ is calculated from the tangent vector of the path. The tangent is projected on the horizontal (East-North) plane and used as reference signal for the yaw control.

The target path velocity ($V_{tar}(s)$) is derived from two input parameters: the cruise velocity $V_c$ (desired velocity for the segment) and the final velocity $V_f$ (velocity that the helicopter must have at the end of the segment). Both velocities are given by the path planners.

The calculation of $V_{tar}(s)$ along the path segment is divided into three phases: *acceleration*, *cruise* and *braking*. The acceleration phase is active only during execution of the first segment of the path. During this phase the velocity increases with a constant rate until the cruise velocity $V_c$ is reached. Note that in this case $V_{tar}$ depends on time rather than on the path parameter ($s$) since it is not important at which position of the path the acceleration phase is terminated.

The braking phase is active when the following condition is satisfied: the remaining path length $d_{end}(s)$ is equal to the distance required to brake the helicopter from $V_c$ to $V_f$ for given deceleration $a_{brake}$:

$$d_{end}(s) = \frac{|V_c^2 - V_f^2|}{2a_{brake}}$$

The target velocity in the braking phase is a function of $d_{end}(s)$:

$$V_{tar}(s) = \sqrt{|2d_{end}(s)a_{brake} + V_f^2|}$$

This guarantees achieving the desired velocity at the end of the path segment. In case $V_f > V_c$, the helicopter accelerates in order to reach $V_f$ at the end of the path segment. The path planner assures the continuity of the velocity profile between segments. In order to make a coordinated turn a consistency check must be done with respect to the generated $V_{tar}(s)$. For such a maneuver, the helicopter must compensate the centripetal acceleration with a certain amount of roll angle. Because of safety reasons the maximum roll angle ($\phi_{max}$) and maximum yaw rate ($\omega_{max}$) are specified. Therefore, the maximum velocity during a turn maneuver is also restricted. The two velocity limits

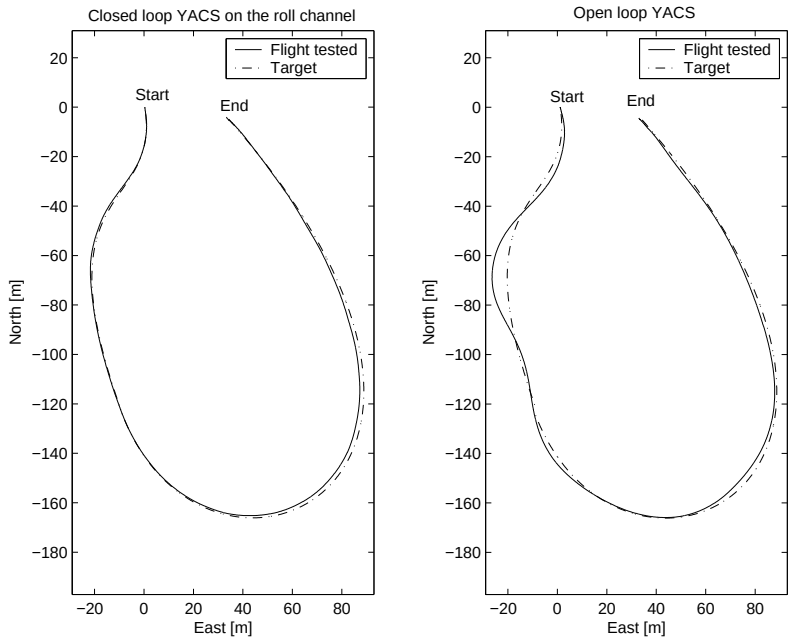


Figure 7: Comparison of path tracking performances using two different roll controller. The flight-test where performed at 36 km/h constant velocity for both paths.

are calculated as follows:

$$\begin{aligned} V_{max1}(s) &= \sqrt{R_{xy}(s) g \phi_{max}} \\ V_{max2}(s) &= \omega_{max}^2 R_{xy}(s) \end{aligned}$$

The minimum between $V_{max1}(s)$, $V_{max2}(s)$ and $V_{tar}(s)$ is taken as target velocity by the path controller described in the next subsection. Thus, the calculated velocity is compatible with the curvature radius of the path.

## 5.2 Path Controller

Usually the control system for a helicopter consists of an *inner loop*, which is responsible for stabilizing the attitude, and the *outer loop*, which controls the position and velocity.

In our system we use the Yamaha Attitude Control System (YACS) provided with the RMAX helicopter for the attitude control. For the outer loop we use four decoupled PID controllers. Their outputs are used as input to the YACS system. In (3) the control approach for the RMAX has been described and experimental results provided.

In this section, we present results of a modified controller, which uses an additional feedback loop on the roll channel. Several experiments were done closing the roll angle loop around the YACS in order to test if it is feasible to improve path tracking precision without a complete redesign of the attitude controller.

The modified controller has been flight-tested on the RMAX helicopter at a constant velocity of 36km/h on a path with changing curvature. Such a path is typically used to navigate in areas with obstacles. The results are shown in Fig. 7 where the same path was tested both with the roll loop closed around the YACS, and without. Note, that at the beginning of the path when the dynamic response of the controller is more important because of the changing curvature, the control with additional feedback loop over the roll channel performs much better than the other one. The error for the closed loop controller in this part is below 1 meter, while for the other is around 6 meters. The results obtained during flight-tests show that the loop on the roll channel reduces the path tracking error, what makes the controller more suitable for obstacle-cluttered environments.

The following section describes interaction between the path following control mode with a Task Procedure responsible for executing a planned path.

# 6 Path Execution Mechanism

The key element of the framework is the path execution mechanism. It allows for seemless integration of hard real-time componets (e.g. the path following controller) with inherently non real-time deliberative services (e.g. the path planners).

The execution of the path provided by the path planner is divided into two parts, namely the Task Procedure and the Path Following controller. The standard path execution scheme in our architecture for static operational environments is depicted in Fig. 8 (key functional componets involved in navigation are drawn in black). A UAV mission is specified via a Task

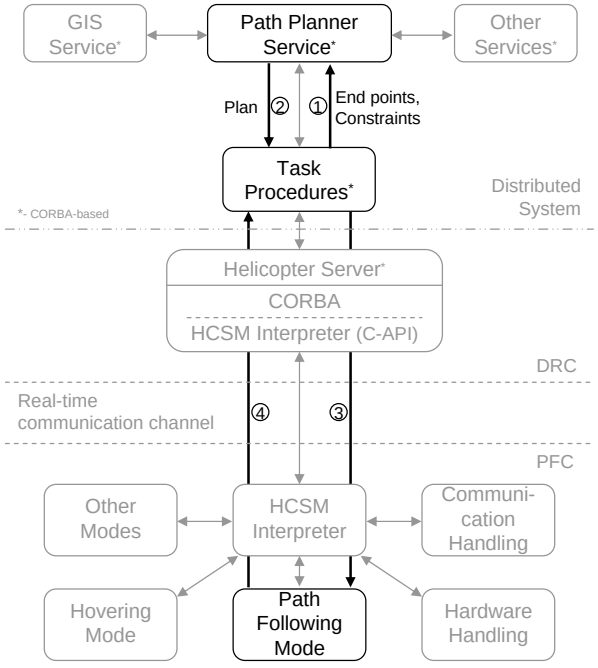


Figure 8: Path execution mechanism

Procedure in the reactive layer of our architecture, (perhaps after calling a task-based planner). For the purpose of this paper, a TP 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 path which is represented as a set of cubic polynomial curves. 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 path via the control system interface and waits for the *Request Segment* event. It is generated by the HCSM responsible for the path execution as soon as the path following (PF) controller output is fed with a path segment.

When a *Request Segment* event arrives (step 4) the TP sends the next segment data to the HCSM which coordinates the path execution. This procedure is repeated (step 3-4) until the last segment is executed. However, because the high-level system is not implemented in hard real-time it may happen that the next segment does not arrive at 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.

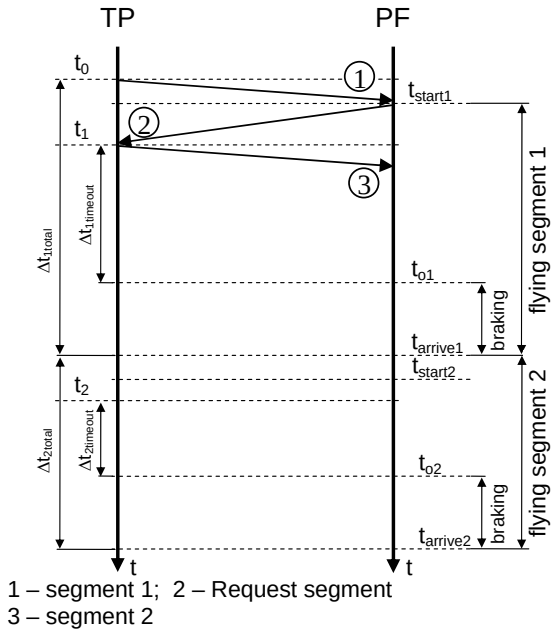


Figure 9: Execution timeline for trajectory consisting of 2 segments

Fig. 9 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 PF controller and waits for a *Request segment* event which arrives immediately ($t_1$) after the helicopter starts to fly ($t_{start1}$). 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 $\Delta_{t_1 timeout}$ time window to send the next segment to the PF controller before it initiates a safety braking procedure. If the segment is sent after $t_{o1}$, the helicopter will start braking. In the current implementation, segments are not allowed to be sent after a timeout. This will be changed in a future implementation. In practice, the $\Delta_{t_1 timeout}$ time window is large enough to replan the path using the standard path planner. The updated segments are then sent to the PF controller transparently.

The described path execution mechanism allows for dynamic replacement of path segments if necessary. The following section describes the process of dynamic path replanning.

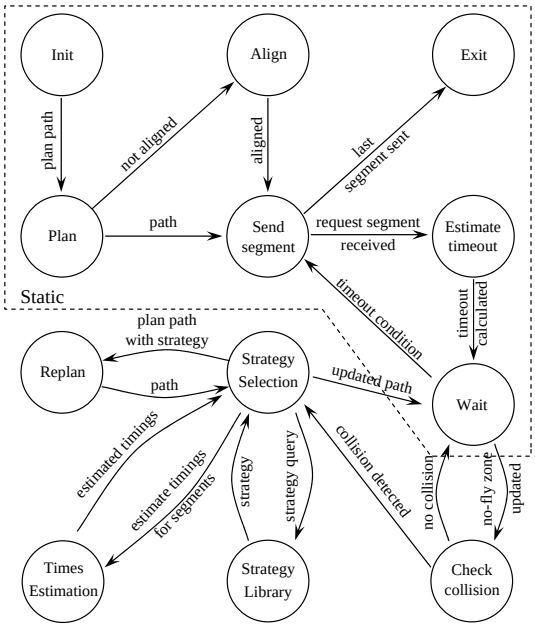


Figure 10: The dynamic path replanning automaton

# 7 Dynamic Replanning of the Path

The design of the path execution mechanism provides a method for feeding path segments to the PF controller iteratively. This is a particularly interesting part of the design because it gives the deliberative or decision-making layer of the architecture the opportunity to anticipate problems at longer temporal horizons and then to modify one or more segments in the original mission path plan based on any contingencies it discovers.

There are several services that are used during the path replanning stage. They are called when changes in the environment are detected and an update event is generated in the system. The augmented state machine associated with the TP used for the dynamic replanning of a path is depicted in Fig. 10. The TP takes a start and an end point and a 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 PF controller (*Send segment* state) and calls the Prediction service to estimate a timeout for the current segment (*Estimate timeout* state). Based on the segment timeout and system latency, a condition is calculated for sending the next segment. If there is no change in the environment the TP waits (*Wait* state) until a timeout condition is true and then sends the next segment to the PF controller.

In case new information about newly added or deleted forbidden regions (no-fly zone updated) arrives, the TP checks if the current path is in collision with the updated world model (*Check Collision* state). If a collision is detected in one or more segments the TP calls a Strategy Selector service (*Strategy Selection* state) to determine which replanning strategy is the most appropriate to use at the time. The Strategy Selector service uses the Prediction service for path timings estimation (*Times Estimation* state) to get estimated timeouts, total travel times etc. It also uses the Strategy Library service (*Strategy Library* state) to get available replanning strategies that will be used to replan when calling the path planner (*Replan* state). The TP terminates when the last segment is sent.

All time estimations that have to do with paths or parts of paths are handled by the Prediction service. It uses the velocity profile of a vehicle and path parameters to calculate timeouts, total times, and combinations of those. For instance, in the case of flying a two-segment trajectory (see execution timeline in Fig. 9) it can estimate timeouts ($\Delta_{t_1 timeout}$, $\Delta_{t_2 timeout}$), total travel times ($\Delta_{t_1 total}$, $\Delta_{t_2 total}$) as well as a combined timeout for the first and the second segment ($t_{o2}$-$t_1$).

When part of a path is not valid anymore, the path planner service can be called in order to repair an existing plan or to create a new one. There are many strategies that can be used at that step which can give different results depending on the situation. The Strategy Library stores different replanning strategies including information about the replanning algorithm to

be used, the estimated execution time and the priority. Example strategies are shown in Fig. 11.

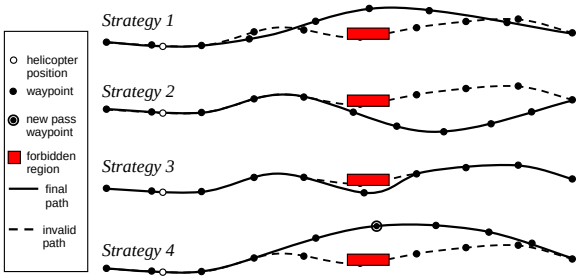


Figure 11: Examples of replanning strategies.

*Strategy 1*

Replanning is done from the next waypoint (start point of the next segment) to the final end point. This implies longer planning times and eventual replacement of collision-free segments that could be reused. The distance to the obstacle in this case is usually large so the generated path should be smoother and can possibly result in a shorter flight time.

*Strategy 2*

Segments up to the colliding one are left intact and replanning is done from the last collision-free waypoint to the final end point. In this case, planning times are cut down and some parts of the old plan will be reused. But since the distance to the obstacle is shorter than in the previous case, it might be necessary for the vehicle to slow down at the joint point of two plans, this can result in a longer flight time.

*Strategy 3*

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

*Strategy 4*

There can be many other strategies that take into account additional information that can make the result of the replanning better from a global perspective. An example is a strategy that allows new pass waypoints that should be included in the repaired plan.

Note that each of these strategies progressively re-uses more of the plan that was originally generated, thus cutting down on planning times but maybe producing less optimal plans. The decision as to which strategy to use is made by the Strategy Selector service. In the current implementation of the framework it uses a simple algoritm for chosing strategies based on user-predefined priorities.

More details on dynamic replanning are presented in (24). One example of using the dynamic replanning technique is presented in the following section.

# 8 Experimental results

In this section, we provide a description of two generic missions, instances of which were flown at the Swedish Rescue Services Agency facilities in Revinge in southern Sweden. The site, which is usually used by firefighters and other emergency rescue forces for training purposes, consists of a number of building structures, a road network and different types of terrain and vegetation (trees, bushes, etc.). A 3D map of the area is provided by an onboard geographical information system (GIS) which is used by the path planner service to generate collision-free paths according to the framework described in the previous sections.

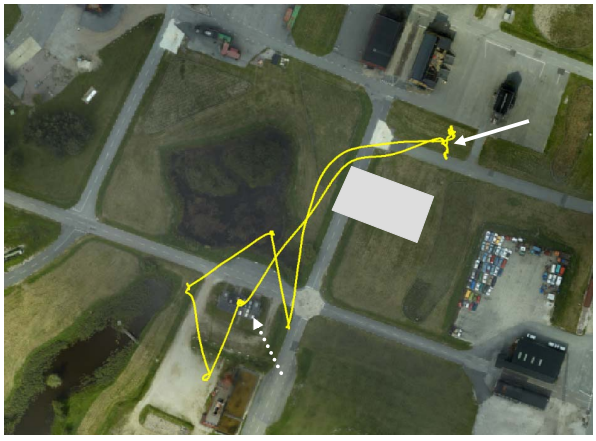

Figure 12: Mission 1. White solid arrow designates take off and landing position. White dotted arrow points to the target building. Gray polygonal area marks the no-fly zone created above the ground station vehicle. Solid line represents the actual flight path.

In the first mission, the UAV took off autonomously and hovered. It then flew towards a building previously designated by the ground operator. Upon arrival at the building, it gathered video footage of all the facades. It then flew back to home base and landed autonomously. The operator's task was to select a building of interest using a ground station user interface. The information was sent to the UAV and the mission plan was generated

Figure 13: Mission 1. Images captured during the mission. Clockwise, starting from the upper left corner of the figure: the North, West, South and Top view of the building.

on-board. As the helicopter was reaching successive hovering positions in front of each facade and over the building roof, the camera was controlled autonomously to keep the object of interest in the center of the image. This kind of mission was performed several times with different buildings chosen as observation targets. The logged flight-test data of one of the missions is plotted on the map in Fig. 12. Fig. 13 presents several frames taken from video footage from the mission demonstrations.

The second mission demonstrates the use of the dynamic replanning capability of the framework. The flight started with autonomous take off, and the helicopter began executing the planned path towards the designated waypoints. After arriving at the first one, the direction of flight changed to south and the ground operator added a no-fly zone intersecting the flight path. The information was sent to the helicopter and the on-board system activated the replanning mechanism. A new path was planned, and the flight continued avoiding the no-fly zone. After the helicopter arrived at the last waypoint, it was commanded to return to home base and land. Fig. 14 shows the logged flight-test data superimposed on the map of the area.

# 9 Conclusions and Future Work

A distributed hardware/software architecture has been described which includes a framework for integrating path planning techniques, a path following control mode, and a path execution mechanism which allow for UAV operation in obstacle-cluttered environments in addition to dynamic replanning of flight paths. The

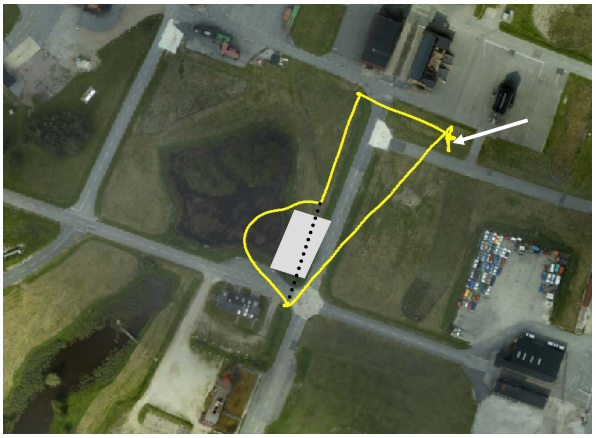

Figure 14: Mission 2. White solid arrow designates take off and landing position. Solid line represents the actual flight path executed counter-clockwise. Gray polygonal area marks the no-fly zone added during the flight by the ground operator. Black dotted line shows invalidated part of the path.

path planning algorithms are based on the use of sample-based probabilistic techniques which sacrifice completeness in plan generation for tractability. Details of the path following controller are provided in addition to experimental results using the framework. These results show that the controller keeps the UAV within one meter of the desired path during flights with a velocity of 10 m/s.

The proposed method for 3D trajectory execution views flight paths as sequences of segments where each segment is incrementally fed to the path following controller. This scheme supports dynamic replanning and replacement of flight path segments (e.g. because of adding no-fly zones). It also enhances the safety of UAV operations. The self-contained PFC computer's role is to request subsequent segments as a mission flight path is being flown. In cases where the new segments do not arrive on time, the PFC automatically switches to a hover control mode. This approach would help to avoid situations where a longer path which is passed to the controller becomes unacceptable as time progresses, but due to communication failure with the rest of the system, is not made aware of the problem. This could lead to potentially catastrophic situations.

The systems and techniques described here have been implemented and fully tested on our WITAS UAV systems. The framework proposed allows for the seamless coexistence of the hard real-time Control Kernel and the soft real-time high-level deliberative system by taking advantage of timing and other characteristics of both.

The Control Kernel is in charge of flight mode switching of the hybrid control system and coordinating the real-time communication among other things. It uses HCSMs to do this. The control kernel is easily extendible and allows for the implementation of additional functionality requiring a rigorous timing regime.

The use of CORBA provides a straightforward means of transparently distributing deliberative services such as task- and motion planners across different processors in a single platform, onto other airborne platforms or onto different ground control stations. Task procedures are used in the implementation of reactive behaviors which provide the software and conceptual *glue* between the lower control layer and the upper deliberative layer in the architecture. Because CORBA is being used, both reactive behaviors and deliberative functionalities can be implemented in many different languages as long as they have supporting IDL mappings.

Future work includes extending two of the services used in the dynamic replanning technique, namely, the Strategy Selector and the Strategy Library services. On the hardware side, inclusion of additional sensors necessary for perceiving the environment reactively are planned. In order to enhance the autonomy of the platform, the need for a 3D elevation map requirement must be loosened. This can be achieved by adding sensors enabling mapping and obstacle avoidance in unknown and dynamic environments. A new and enhanced version of the Control Kernel is also under evaluation. Among other features, it introduces data flow support into the state machine concept.

# 10 Acknowledgements

This work is funded by the Wallenberg Project under the WITAS UAV Project. Many members of the WITAS UAV project helped to make this work possible.

# References

1. MARVIN: TU Berlin. http://pdv.cs.tu-berlin.de/MARVIN/.

2. M. La Civita. *Integrated Modeling and Robust Control for Full-Envelope Flight of Robotic*

*Helicopters*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2003.

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

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

5. M. Egerstedt, X. Hu, and A. Stotsky. Control of mobile platforms using a virtual vehicle approach. *IEEE Transactions on Automatic Control*, 46(11):1777–1782, November 2001.

6. E. Frazzoli, M. Dahleh, and E. Feron. Robust Hybrid Control for Autonomous Vehicles Motion Planning. In *Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1999. Technical report LIDS-P-2468.*, 1999.

7. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

8. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

9. MIT/Draper Autonomous Helicopter Project. http://web.mit.edu/whall/www/heli/.

10. E. N. Johnson and S.K. Kannan. Adaptive flight control for an autonomous unmanned helicopter. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2002.

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

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

13. P Mantegazza *et. al.* RTAI: Real time application interface. *Linux Journal*, 72, April 2000.

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

15. T. Merz, S. Duranti, and G. Conte. Autonomous landing of an unmanned aerial helicopter based on vision and inertial sensing. In *Proc. of the 9th International Symposium on Experimental Robotics*, 2004.

16. B. Mettler, M.B. Tischler, and T. Kanade. System identification modeling of a small-scale unmanned helicopter. *Journal of the American Helicopter Society*, October 2001.

17. P-O Pettersson. Using Randomized Algorithms for Helicopter Path Planning. *Lic. Thesis Linköping University.*, 2006.

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

19. AVATAR: USC Autonomous Flying Vehicle Project. http://www-robotics.usc.edu/∼avatar.

20. BEAR: Berkeley Aerorobot Team. http://robotics.eecs.berkeley.edu/bear/.

21. Georgia Tech UAV. http://www.ae.gatech.edu/labs/controls/uavrf/.

22. Hummingbird: Stanford University. http://sun-valley.stanford.edu/users/heli/.

23. CMU Autonomous Helicopter Project. www.cs.cmu.edu/afs/cs/project/chopper/www.

24. M. Wzorek and P. Doherty. Preliminary Report: Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle. In *Proceedings of the 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG-05)*, 2005.