Linköping Studies in Science and Technology

Thesis No. 1509

Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems

by

Mariusz Wzorek

Linköping University
INSTITUTE OF TECHNOLOGY

Submitted to Linköping Institute of Technology at Linköping University in partial fulfilment of the requirements for degree of Licentiate of Engineering

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2011

ISBN 978-91-7393-037-6
ISSN 0280–7971
Printed by LiU Tryck 2011

URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-71147

Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems

by

Mariusz Wzorek

November 2011
ISBN 978-91-7393-037-6
Linköping Studies in Science and Technology
Thesis No. 1509
ISSN 0280–7971
LiU–Tek–Lic–2011:48

ABSTRACT

Unmanned aircraft systems (UASs) are an important future technology with early generations already being used in many areas of application encompassing both military and civilian domains. This thesis proposes a number of integration techniques for combining control-based navigation with more abstract path planning functionality for UASs. These techniques are empirically tested and validated using an RMAX helicopter platform used in the UASTechLab at Linköping University. Although the thesis focuses on helicopter platforms, the techniques are generic in nature and can be used in other robotic systems.

At the *control level* a navigation task is executed by a set of control modes. A framework based on the abstraction of hierarchical concurrent state machines for the design and development of hybrid control systems is presented. The framework is used to specify reactive behaviors and for sequentialisation of control modes. Selected examples of control systems deployed on UASs are presented. Collision-free paths executed at the control level are generated by path planning algorithms. We propose a path replanning framework extending the existing path planners to allow dynamic repair of flight paths when new obstacles or no-fly zones obstructing the current flight path are detected. Additionally, a novel approach to selecting the best path repair strategy based on machine learning technique is presented. A prerequisite for a safe navigation in a real-world environment is an accurate geometrical model. As a step towards building accurate 3D models onboard UASs initial work on the integration of a laser range finder with a helicopter platform is also presented.

Combination of the techniques presented provides another step towards building comprehensive and robust navigation systems for future UASs.

This work has been supported by the National Aeronautics Research Programs NFFP04-S4202, NFFP04-S4203, NFFP05, and the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII as well as the Excellence Center at Linköping-Lund in Information Technology (ELLIIT) project grants.

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Acknowledgements

I would like to thank my supervisors: Patrick Doherty, Jonas Kvarnström and Andrzej Szałas for giving me the opportunity to work on a variety of challenging and exciting projects and for creating an excellent and stimulating work environment.

Special thanks to my research fellow and a great friend Piotr Rudol with whom I have worked on many projects, coauthored papers and shared the experience of graduate studies.

I am also very grateful to all former and current members of AIICS with whom I have worked and coauthored many publications: Torsten Merz, Gianpaolo Conte, Simone Duranti, Fredrik Heintz, Karol Korwel, Rafał Zalewski, Łukasz Majewski, David Landén, Patrik Haslum, Per Nyblom, Björn Wingman and Tommy Persson.

I also like to extend my thanks to Jonas Kvarnström, Fredrik Heintz, Piotr Rudol and Gianpaolo Conte for invaluable input and reviewing various drafts of this thesis.

Last but not least, I would like to express my deepest gratitude to my wife for her unconditional and untiring support in the process of my studies and writing of this thesis.

Contents

List of Figures

List of Tables

Abbreviations

CK	Control Kernel
DRC	Deliberative Reactive Computer
ESM	Extended State Machine
HCSM	Hierarchical Concurrent State Machine
ICP	Iterative Closest Point
IPC	Image Processing Computer
LRF	Laser Range Finder
OBB	Oriented Bounding Box
PC	Path Controller
PFC	Primary Flight Computer
PFCM	Path Following Control Mode
PRM	Probabilistic Roadmap
RRT	Rapidly exploring Random Trees
SMO	Sequential Minimal Optimization
SVM	Support Vector Machines
TG	Trajectory Generator
TP	Task Procedure
UAS	Unmanned Aircraft System
UAV	Unmanned Aircraft Vehicle
YACS	Yamaha Attitude Control System

Chapter 1

Introduction

The development of unmanned aircraft systems (UAS) has gained tremendous momentum over the last two decades. These platforms offer many advantages over both manned aircraft and other types of autonomous robots. They provide the ability to gather rich bird's eye view information in areas that may not be easily accessible to mobile ground vehicles. Because of their mobility they also provide fast response times which is crucial in many applications such as search and rescue. A UAS can operate in hazardous environments, for example military conflicts or natural catastrophes, without endangering the lives of human pilots or operators. All of this contributes to reducing the cost of operation/missions and contributes to an increasingly high interest from both military and civil markets. Previous market studies [89] estimated that by 2013 the worldwide UAS market will be worth $USD10.6 billion. Some of the most recent estimates show that the U.S. military UAS market alone will generate $USD62 billion revenues during 2010-2015 [56].

Although the early UAS application areas have been predominantly military (e.g. reconnaissance, surveillance, and target tracking) the technology has reached a high maturity level and is economically feasible for many civil areas. Example civil applications include security surveillance [4, 35], power line inspection [42, 54] and support for emergency services in natural catastrophes [33, 78]. Many new applications will also arise as countries develop new regulatory policies allowing UAS usage in unsegregated areas. Consequently, unmanned aircraft are currently the subject of intensive research in numerous fields.

The desired level of autonomy for unmanned aircraft may vary depending on the type of mission being flown, where certain missions need to be controlled in some detail by a ground operator while others should preferably be fully autonomous. But for some aspects of a mission, automation is almost always desirable. One such aspect is path planning and navigation, given a map of static and dynamic obstacles fly through or visit a set of

Figure 1.1: The UASTechLab RMAX helicopter platform.

waypoints.

This thesis focuses on selected aspects of navigation functionalities and path planning for UAS platforms. The techniques that are presented have been implemented and tested on an autonomous helicopter platform developed in the UASTechLab[1] (Figure 1.1).

1.1 Thesis Contributions

The thesis extends existing navigation capabilities of UASTechLab RMAX UAV and builds on previous work done in motion planning [74], control [17], system modeling and software architectures [24, 65].

The four main contributions of this thesis are:

- A modeling framework for hybrid control systems used to specify reactive behaviors and for sequentialisation of control modes.
- A path replanning framework extending the previously used path planners to allow dynamic regeneration or repair of flight paths in case of newly detected obstacles.
- A new approach to selecting the best path repair strategy to use in any given situation, using machine learning to adapt to different maps and different computational hardware.
- Integration of a laser range finder with the existing UASTechLab RMAX UAV to provide environment models for navigation tasks.

[1] `http://www.ida.liu.se/divisions/aiics/aiicssite/uastech/`

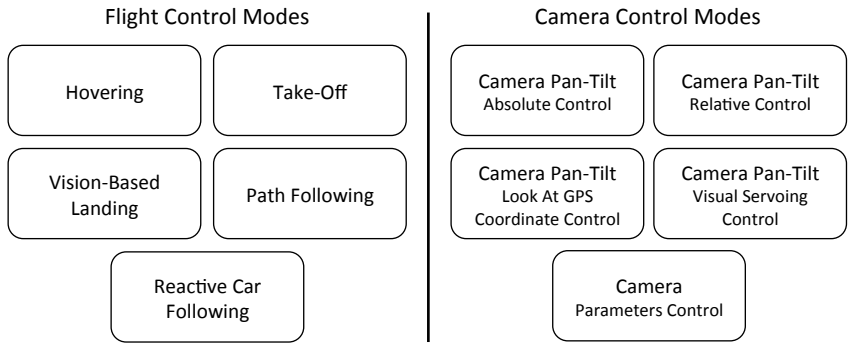


Figure 1.2: Control modes implemented on the UASTechLab RMAX helicopter platform.

Before we can describe these contributions in more detail, we provide a short overview of the different types of algorithms used on a typical robotic platform.

Generally in any autonomous robotics systems one can distinguish functionalities of different complexity and thus different timing requirements. Typically, at the lowest level a set of control functions are executed. Those functions require fast update rates in order to react in a timely manner to perceived changes in the environment. At a higher-level a set of algorithms related to solving more complex tasks are implemented. Those components typically include planners (e.g. task or motion planners).

Control System Modeling

The first contribution of this thesis is at the lowest level, where a set of *control modes* provide an interface to the hardware components. The role of each such control mode is to generate control signals to meet a well-defined control objective. A good example is a hovering function where the objective is to keep the helicopter in the desired position, heading and altitude given current sensor readings. More sophisticated modes may contain multiple stages, as in the take-off mode where the control strategy is more complex and several control objectives are desirable during different phases of the execution. Those can be viewed as basic low-level control behaviors.

Flight control modes implemented on the UASTechLab RMAX UAV system (Figure 1.2) include hovering, take-off, path following, vision-based landing, and reactive car following. Additionally, a set of control modes for payloads has been developed. Those include, for example, control over a camera pan-tilt unit and camera parameters.

When executing a mission a set of low-level control modes are invoked. They can be called in a sequence or in some cases in parallel in order to

achieve a certain mission objective. An example mission is building surveillance, where the goal is to gather video streams or images of each of the facades of a building. Control modes executed in such a mission include take-off, hovering, path following, camera control and landing.

The problem of executing a sequence of control modes is non-trivial. Switching between different control modes requires a number of conditions to be satisfied. The switching conditions vary depending on the particular mode transition. For example, in the case of switching between the hovering and path following modes, a check needs to be done if the current helicopter heading is aligned with the path to be flown. If the heading difference is too large, an additional yaw maneuver is necessary before the transition to the path following mode. Otherwise, the execution of the path could result in a maneuver that potentially leads to a crash.

The most common way of modeling complex reactive systems and hybrid control systems is to use the concept of state machines [39, 48]. One of the contributions of this thesis is the development of a modeling framework for hybrid control systems based on the idea of state machines, called hierarchical concurrent state machines (HCSM).

The HCSM framework enables us to efficiently model all low-level components and solves the problem of mode switching. Its components contain functions implementing continuous control laws and mode switching is realized using the proposed framework. The HCSMs can be represented as state transition diagrams similar to those of statecharts [39]. 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 an 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 periodic task with high execution rate. It allows all *functional units* of the control system to be coordinated, from the lowest level hardware components (e.g. device drivers) through control laws (e.g. hovering, and dynamic path following) and communication to the high-level deliberative components.

Path planning

At a higher level, our UAS platform uses a set of path planners to find collision-free paths from a given start position to a goal position. The second and third contributions of the thesis are related to path planning.

The problem of finding optimal paths for a helicopter platform is intractable in general. This is because the planning is performed in a high-dimensional state space which not only includes a physical 3D position but also additional dimensions (e.g. velocity and heading).

A number of sample-based methods for generating motion plans have been proposed in the literature, including probabilistic roadmaps (PRM [44]) and rapidly exploring random trees (RRT [49]). Sample-based approaches

often make the path planning problem solvable in practice by sacrificing completeness and optimality. A successful deployment of both PRM and RRT algorithms in the static environments for the UAS domain has been presented in Pettersson [74].

Unfortunately in real applications, maps are not perfect, and new obstacles may be detected during flight. If such obstacles appear along the planned flight path, the proper course of action depends on the amount of time available for collision avoidance.

If very little time is available, we must rely on reactive sense-and-avoid procedures, even though this may turn the aircraft in a direction that is far from optimal considering the known obstacles and the current destination.

However, given typical detection ranges and airspeeds, there may be up to a few seconds to decide exactly what to do. One of the contributions of this thesis is the development of a technique that takes advantage of the available time in order to improve the overall path quality and avoid the need of using non-optimal reactive sense-and-avoid procedures.

In many situations there can be sufficient time to invoke a motion planner once again to repair the plan before reaching the point where the aircraft has to divert from its original trajectory. The new trajectory will then take both new and old obstacles into account, potentially saving considerable amounts of flight time. This is especially true for fixed-wing aircraft, where the minimum turn radius is often large and where slowing down and hovering is not an option.

In replanning, we can choose which parts of the original path are replaced and which parts are retained. For example, we can replan from the next waypoint all the way to the goal or only the part of the plan that is actually intersected by the newly detected obstacle. This choice will have a significant effect on not only the quality of the repaired plan, but also the time required for replanning [100].

Many motion planning methods are also parameterized in various ways. For example, PRM planners generate a roadmap graph in a pre-processing phase and search this graph whenever a plan is required. Increasing the number of nodes in the graph will increase plan quality, but again, this will also affect the time required for plan generation.

A replanning *strategy* represents a specific choice of which parts of a path are replanned and which parameters are given to the motion planning algorithm. Our objective is to always choose the strategy that will yield the highest quality possible within the available time. But while there may be a general trend for one strategy to be better or faster than another, the exact time requirements for most strategies will vary considerably depending on factors such as the local environment around the original path and the remaining distance to the destination. Thus, we essentially have two options: Always choose a simple strategy for which we can find a low upper bound on the time requirements, or generate better and more informed predictions by *learning* how the local environment affects timing and quality. The solution

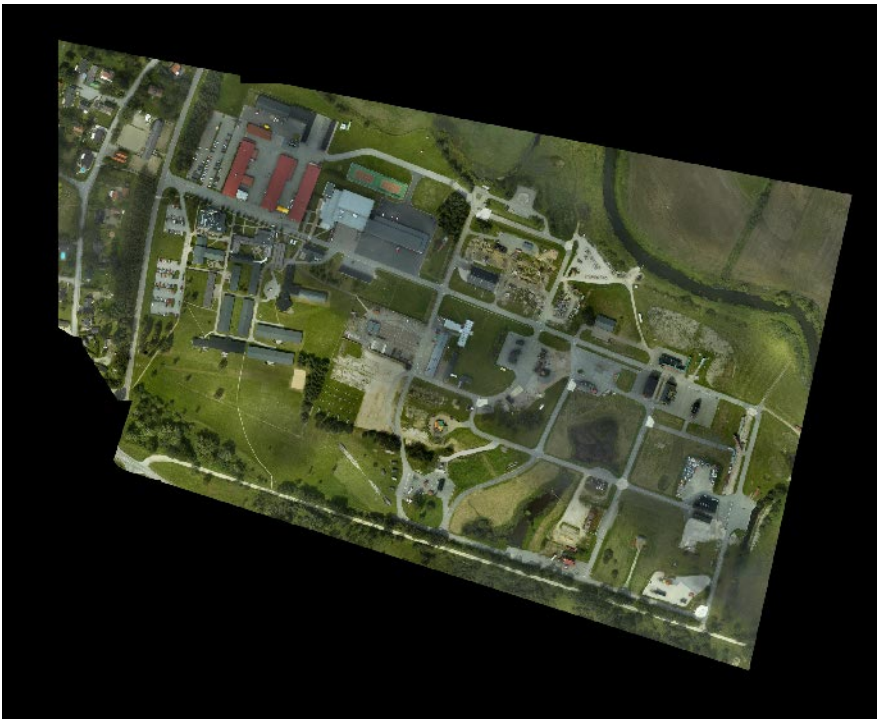


Figure 1.3: Top view of the Revinge training site.

presented in this thesis uses machine learning techniques to generate a set of models useful for performing such predictions. The empirical testing that has been performed shows promising results: In each test environment, flight times could be improved up to 25% compared to the use of a fixed replanning strategy, resulting in times close to the best achievable with the available planning algorithms.

Environment Mapping

The fourth contribution of this thesis relates to the very important aspect of navigation, of acquiring a 3D model of the environment. In order to navigate safely in a real-world environment a geometrical model is required. As previously described, the path planning algorithms rely on an accurate description of the environment in order to produce collision-free paths. Additionally, environments often change over time (e.g. new building structures are added) and existing models have to be updated in order to ensure safe navigation.

Previously, the UASTechLab RMAX UAV system has been using 3D models delivered by a third-party company. The main environment used

for experimentation with our systems is a training area for rescue workers in Revinge in southern Sweden. Figure 1.3 presents a view of the environment from above. The area is approximately 1 square kilometer in size and contains a road network, several buildings, and a number of other structures such as masts, lampposts and fences. This makes it a very well suited environment for many interesting and advanced scenarios, such as traffic monitoring or search and rescue missions.

Acquiring 3D maps has gained considerable attention over the last two decades as the required hardware technology matures. There exist a variety of sensors that can provide 3D geometrical information and algorithms that make use of them. Examples of such sensors and techniques include stereo vision systems [47], structure from motion using a single camera [22], and laser range finders [88]. Laser range finders have gained a tremendous popularity in many robotic applications because of their relatively small size, high accuracy, and low price.

We present initial results of integration of a laser range finder with the UASTechLab RMAX UAV platform. The sensor is mounted on a rotational mechanism that has been developed within the group. This solution allows for obtaining 3D point clouds even when the helicopter is stationary.

Two applications of the laser range finder for map building are considered in this thesis. In the first, a 3D map of the environment is built offline, after an exploratory flight over all building structures is performed and all the data is collected. The second application relates to the use of a laser range finder sensor for collision avoidance in the context of the proposed dynamic replanning framework. An analysis of potential uses of such a sensor in this setting is provided.

1.2 List of Publications

The work presented in this thesis is mainly based on the following publications.

[102] *Mariusz Wzorek, Jonas Kvarnström and Patrick Doherty.* Choosing Path Replanning Strategies for Unmanned Aircraft Systems. *In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2010).*

[63] *Mariusz Wzorek and Patrick Doherty.* Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle. *In Proceedings of the International Conference on Automated Planning and Scheduling, Extended Abstract, (ICAPS 2006).*

[99] *Mariusz Wzorek and Patrick Doherty.* Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle. *In Proceedings of the IEEE International Conference on Hybrid Information Technology (ICHIT 2006).*

[**100**] *Mariusz Wzorek, Gianpaolo Conte, Piotr Rudol, Torsten Merz, Simone Duranti and Patrick Doherty.* From Motion Planning to Control - A Navigation Framework for an Autonomous Unmanned Aerial Vehicle. *In Proceedings of the 21th Bristol International UAV Systems Conference 2006.*

[**62**] *Mariusz Wzorek and Patrick Doherty.* The WITAS UAV Ground System Interface Demonstration with a Focus on Motion and Task Planning. *System Demonstration during the International Conference on Automated Planning and Scheduling (ICAPS 2006), Extended Abstract.*

[**67**] *Torsten Merz, Piotr Rudol, Mariusz Wzorek.* Control System Framework for Autonomous Robots Based on Extended State Machines. *In Proceedings of the International Conference on Autonomic and Autonomous Systems (ICAS 2006).*

[**61**] *Mariusz Wzorek and Patrick Doherty.* Preliminary Report: Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle. *In Proceedings of the 24th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2005).*

[**80**] *Piotr Rudol, Mariusz Wzorek, Rafał Zalewski, and Patrick Doherty.* Report on sense and avoid techniques and the prototype sensor suite. *National Aeronautics Research Program NFFP04-031, Autonomous flight control and decision making capabilities for Mini-UAVs, 2008.*

Other publications that are not directly related to this thesis:

[**101**] *Mariusz Wzorek, David Landén and Patrick Doherty.* GSM Technology as a Communication Media for an Autonomous Unmanned Aerial Vehicle. *In Proceedings of the 21th Bristol International UAV Systems Conference 2006.*

[**81**] *Piotr Rudol, Mariusz Wzorek, and Patrick Doherty.* Vision-based pose estimation for autonomous indoor navigation of micro-scale unmanned aircraft systems. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2010).*

[**79**] *Piotr Rudol, Mariusz Wzorek, Gianpaolo Conte and Patrick Doherty.* Micro Unmanned Aerial Vehicle Visual Servoing for Cooperative Indoor Exploration. *In Proceedings of the 2008 (AEROCONF 2008).*

[**25**] *Simone Duranti, Gianpaolo Conte, David Lundström, Piotr Rudol, Mariusz Wzorek, Patrick Doherty.* Linkmav, a Prototype Rotary Wing Micro Aerial Vehicle. *In Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace (IFAC 2007).*

[**19**] *Gianpaolo Conte, Maria Hempel, Piotr Rudol, David Lundström, Simone Duranti, Mariusz Wzorek, and Patrick Doherty.* High Accuracy

Ground Target Geo-location Using Autonomous Micro Aerial Vehicle Platforms. *In AIAA Guidance, Navigation, and Control Conference, 2008, volume 26, Honolulu, Hawaii, 2008.*

[46] *A. Kleiner, C. Dornhege, R. Kümerle, M. Ruhnke, B. Steder, B. Nebel, P. Doherty, M. Wzorek, P. Rudol, G. Conte, S. Durante, and D. Lundström.* Robocuprescue - Robot League Team Rescuerobots Freiburg (Germany). *In RoboCup 2006 (CDROM Proceedings), Team Description Paper, Rescue Robot League, 2006.*

1.3 Thesis Outline

Chapter 2 presents background information on the autonomous helicopter used as a testbed platform for the techniques presented in the thesis. The chapter starts with a description of the RMAX helicopter platform itself as well as the avionics system that has been developed in the UASTechLab. The second part of the chapter provides details on the software system. Specifically, the UASTechLab software architecture is discussed. The last section describes the path following control mode (PFCM) which is used for the execution of segmented paths.

Chapter 3 provides a description of the hierarchical concurrent state machine (HCSM) framework used for modelling the hybrid control system. The description includes HCMS syntax and semantics, and provides a set of practical examples of HCSM use for the UASTechLab RMAX platform.

The dynamic path replanning framework which uses a set of replanning strategies together with available path planners to provide a collision avoidance mechanism is presented in chapter 4.

A new method for selecting path replanning strategies using machine learning techniques is the topic of chapter 5. This selection mechanism is used in the dynamic replanning framework when a new obstacle occluding the currently executed flight path is added.

Chapter 6 presents initial results in the development of a set of environment mapping techniques targeted for navigation applications. The chapter discusses the integration of a laser range finder with the UASTechLab RMAX system. Data sets gathered during a number of flight test campaigns are included. The last section presents results of using these generated maps with path planning algorithms.

Conclusions and future work is presented in chapter 7.

Chapter 2

The UASTechLab RMAX System

A wide range of research with unmanned aircraft systems is conducted in the UASTech Laboratory[1]. Over the years a number of platforms have been successfully developed and deployed. The platforms vary in size, weight and flight capabilities. Figure 2.1 presents a number of micro air vehicles with sizes of 50-70 cm in the largest dimension and weight from 0.5 kg to 1.4 kg.

Figure 2.1: From left to right: LinkQuad (a quadrotor platform); PingWing (a fixed wing platform); LinkMAV (a coaxial helicopter).

One of the main platforms used in our lab is a Yamaha RMAX helicopter (Figure 2.2). The techniques described in this thesis have been developed for the RMAX helicopter. This chapter starts with a description of the hardware platform which includes the Yamaha RMAX helicopter and an avionics system developed in our lab [23]. The second section begins with background information on several aspects of the system itself as well as the navigation functionalities. It starts with a high-level overview of the architecture, followed by a short description of a control kernel (CK). The CK includes a set of basic control modes used in the system. The chapter ends with a detailed description of a path following control mode used for the execution of segmented paths.

[1] `http://www.ida.liu.se/divisions/aiics/aiicssite/uastech/`

2.1 The Hardware Platform

The RMAX helicopter has been developed by the Yamaha Motor Company[2] mainly for agricultural applications (e.g. crop spraying). The helicopter has a total length of 3.6 m (including the main rotor) and is powered by a 21 hp engine with a maximum takeoff weight of 95 kg. The RMAX has a built-in attitude sensor (YAS) and an attitude control system (YACS).

Figure 2.2: The Yamaha RMAX helicopter with a laser range finder mounted on a rotation mechanism.

The RMAX helicopter platform itself is radio-operated but with an additional upgrade it is possible to use a computer system to implement autonomous flight capabilities. Due to vibrations as well as limitations in power and cooling, unmanned aircraft require very robust computational hardware with low power requirements. Our hardware platform which interfaces to the helicopter systems (Figure 2.3) contains three embedded PC104 computers connected by RS232 serial lines for hard real-time networking as well as Ethernet for distributed applications, remote login and file transfer. The onboard Ethernet switch is also connected to an 802.11b wireless Ethernet bridge for communication with the ground station.

The deliberative/reactive system (DRC) runs on a 1.4 GHz Pentium Mobile computer. The functionalities executed on the DRC system include high-level deliberative algorithms and a reactive execution mechanism which interacts with a low-level control system.

The primary flight control system (PFC) executes functionalities related to flight capabilities. The PFC runs on a 700 MHz Pentium III computer, and is connected to a number of sensors. A real-time kinematics (RTK) GPS receiver provides high accuracy position information. The altitude is provided

[2]Homepage: `http://www.yamaha-motor.co.jp/global/`

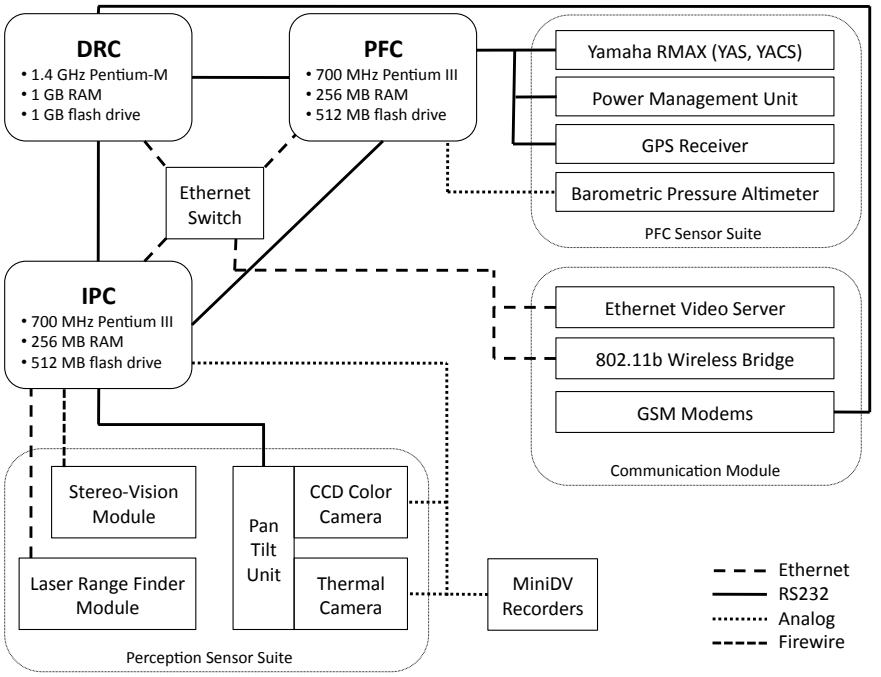


Figure 2.3: On-Board Hardware Schematic.

by an absolute barometric pressure sensor. The PFC is also connected to a software-controlled power management unit which provides information on voltage levels and current consumption. Additionally, it allows for remote device power control i.e. switching on/off. The unit controls all three PC104 computers, sensors, and the wireless Ethernet bridge.

The image processing system (IPC) runs on another 700 MHz Pentium III computer, which is equipped with a 4-channel framegrabber. Current vision sensors include a color CCD[3], a thermal camera[4], and a stereo vision system (STH-DCSG) from Videre Design[5]. The CCD and thermal cameras are mounted on a pan/tilt unit. All of the images can be processed onboard the UAS, saved on the computer's flash drive and transmitted to the ground station by an Ethernet video server. Due to the limited bandwidth of the 802.11b Wi-Fi Ethernet channel, the video is transmitted in lower than nominal PAL resolution. Additionally, two MiniDV recorders are used to store the full resolution video for offline analysis and to support computer vision algorithm development.

A popular SICK LMS-291[6] laser range finder has also been integrated on

[3]FCB-EX780BP from Sony. Homepage: `http://www.sony.com`
[4]ThermalEye from L-3 Communications. Homepage: `http://www.l-3com.com`
[5]Homepage: `http://www.videredesign.com`
[6]SICK AG. Homepage: `http://www.sick.com`

the UAS platform. The original sensor has been modified in order to reduce its weight and is mounted on an in-house developed rotation mechanism. Details about the integration and sensor use are provided in chapter 6.

The platform has also been equipped with two GSM modems: an EDGE modem[7], and a GPRS modem[8]). The GSM technology is a mature, commercially available communication infrastructure which is likely to be continuously developed, refined and supported in the future. It permits the operation of UASs at large distances, out-of-sight from the ground operator. It provides a good redundant system alternative in the case where other communication frequencies are jammed. A multi-modal graphical user interface has been designed, constructed and implemented on a Sony Ericsson P900 mobile telephone. The results of the experiments are presented in Wzorek et al. [101].

At the time of writing this thesis a new hardware platform is being developed. The new version includes a custom in-house designed avionics box where all the hardware components are enclosed. All of the computer hardware is upgraded to provide more computational power while keeping adequate power consumption. Computers will use the newest solid state disk (SSD) technology for storage. This will increase the capacity of storage as well as performance over compact flash cards that are used at the moment. On the sensor side, a new and more accurate GPS receiver will be integrated. Furthermore, the absolute barometric pressure sensor will be improved. It is currently connected to the PFC analog-to-digital extension board which introduces additional unnecessary sensor noise due to electromagnetic field disturbances. The analog to digital conversion will be performed by a microcontroller mounted on the same custom made printed circuit board (PCB) as the sensor. The sensor reading will then be sent via the RS232 serial line. The power management unit has also been redesigned to extend its current/voltage measuring accuracy and to make it easier to extend in the future.

2.2 The Software System

One of the necessary foundations for any autonomous system is an appropriate software system architecture. This is important from many perspectives. First of all, it provides a clear division between components with different characteristics. It also defines how all the functionalities present in the system interact with each other. Additionally, it helps to deal with the complexity of the system.

By far the most popular and successful type of architecture used in robotic systems today is a *hybrid architecture*. It combines the best features of reactive techniques (e.g. real-time, tight hardware integration) and

[7]Aplicom 12 from Aplicom Oy. Homepage: `http://www.aplicom.com`

[8]Trizium from Telit Communications PLC. Homepage: `http://www.telit.com`

deliberative techniques (e.g. solving complex tasks) into one architecture. Many variations of hybrid systems have been proposed in the literature and successfully deployed [90, 91]. Probably the most recognizable category is a *three-layer architecture* [32].

2.2.1 The UASTechLab Software Architecture

A hybrid deliberative/reactive software architecture (D/R) based conceptually on the idea of a three-layer architecture has been developed for the UASTechLab RMAX system [23]. 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. Figure 2.4 presents the functional layer structure of the architecture and emphasizes its reactive-concentric nature.

The three abstraction layers differ in several aspects, including timing requirements, the amount of knowledge of the environment (models) that they require, and the type of functionality which they encapsulate.

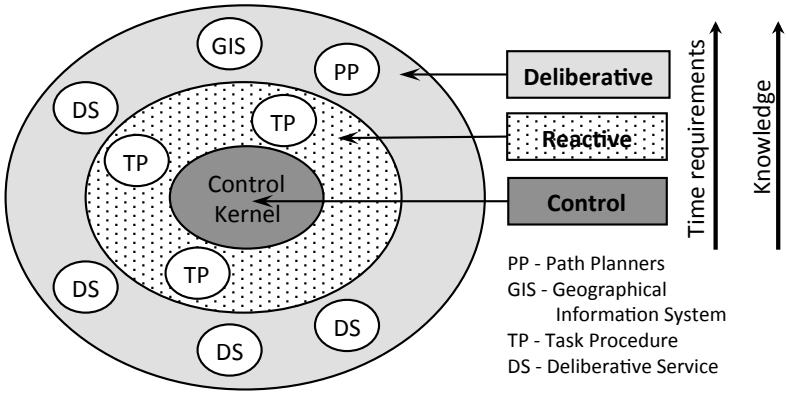


Figure 2.4: The UASTechLab hybrid deliberative reactive architecture.

The deliberative layer includes high-level components that generate solutions to complex tasks, typically using planners (e.g. task or motion planners). The world models used by such algorithms usually have rich semantics and because of the complexity of the tasks they are solving, the time requirements are high. The decision cycle in this layer is in the magnitude of seconds or even minutes.

Components in the reactive layer coordinate the execution of high-level plans and implement a set of robot behaviors (e.g. surveillance of a region). The integration with both control modes and deliberative services puts higher timing requirements than on the deliberative layer. Additionally, a set of efficient mechanisms for modeling behaviors is required. The world models used in this layer are usually simpler than the ones used by

the high-level deliberative components.

The lowest control level encapsulates a set of functions implementing control laws (e.g. hovering), tightly coupling perceived sensor data to robot actuators.

The remainder of this section focuses on describing the timing aspects of the UASTechLab architecture and provides an overview of the components used in this thesis.

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)[9].

All three computers in our UAS platform (i.e. PFC, IPC and DRC, see Figure 2.3) 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 (non-real-time) 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 its time is spent on running the deliberative services. Among others, the most important ones from the perspective of this thesis are the Path Planner, the Geographical Information System (GIS), the Task Procedure Execution Module and the Helicopter Server which encapsulates the Control Kernel (CK) of the UAS system. The CK which is distributed over all three computers deals with basic low-level behaviors by execution of continuous control modes. Figure 2.5 presents the navigation subsystem and the main software components.

The high-level part of the system (the reactive and deliberative layers) has reduced timing requirements and is responsible for coordinating the execution of reactive Task Procedures (TPs [23, 73]). This part of the system uses the Common Object Request Broker Architecture (CORBA [96]) 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 UAS (i.e. take-off, vision-based landing, hovering, path following described in section 2.2.3 or reactive flight modes for interception and tracking). Additionally, TPs can control the payload of the UAS platform which currently consists of CCD and thermal cameras mounted on a pan-tilt unit. TPs receive data delivered by the PFC and IPC computers, i.e. helicopter state and camera system state

[9] Note that distinction between the Control Kernel and the High-level system is done based mainly on the timing characteristics and it does not exclude, for example, placing some deliberative services (e.g. prediction) in the Control Kernel.

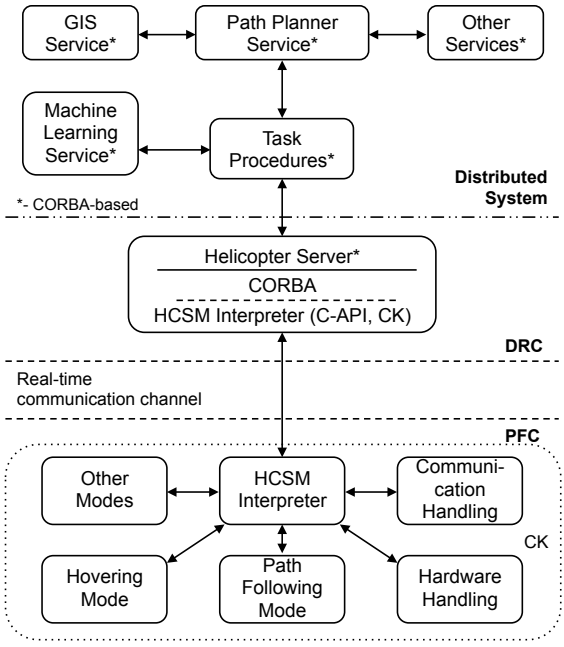


Figure 2.5: Navigation subsystem and main software components

(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 hierarchical concurrent state machines (HCSMs) running in the real-time part of the DRC software.

The presented software architecture is used to achieve missions which require continuous control laws such as path following (section 2.2.3) as well as deliberative services such as path planners (chapter 4). Details of the interaction between the TPs, path planners and the Control Kernel are presented in section 3.3.2.

The next section describes some of the practical implementation details of the Control Kernel.

2.2.2 The Control Kernel

The Control Kernel (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 UASTechLab.

The CK is implemented using C code. This part of the system uses the Real-Time Application Interface (RTAI [60]) 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 Aerospace Engineering of Politecnico di Milano[10] (DIAPM).

Real-time performance can be achieved by using either specially created user-space programs or kernel modules. In our case, one kernel module is created and inserted into the Linux kernel space. One of the RTAI features allows for creation of a kernel module that takes full control over the processor. Because of that, 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 is run preemptively and can be interrupted at any time. For that reason a locking mechanism is used when both user- and kernel-space processes communicate through 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 the safety of the operation of the UAS platform which is especially important in urban environments.

The Control Kernel has a hybrid flavor. Its components contain functions implementing continuous control laws and mode switching is realized using the HCSMs presented in chapter 3.

2.2.3 Path Following Control Mode

One of the control modes executed and coordinated by the Control Kernel is the Path Following Control Mode (PFCM [18, 20], Figure 2.7 and the bottom part of Figure 2.5) which executes paths consisting of a set of segments. Figure 2.6 presents an example path consisting of three segments.

Before describing the details about the PFCM mode two important concepts are described, namely the *inner* and *outer control*. The role of the *inner control* is to stabilize the helicopter attitude and the vertical dynamics by using a feedback loop. As previously described, the RMAX helicopter has a built-in stabilization system (YACS) which is used by the PFCM. If the helicopter is flown manually using a RC radio transmitter, the positions of the sticks are used as the input to the inner control loop. The stabilization of the helicopter makes it easier for the human pilot to operate.

The *outer control* is a feedback loop which based on the reference parameters provides control signals to the inner control loop. The reference parameters are the desired helicopter position, velocity and heading.

In the classical approach to the path following problem, a trajectory (path in space and time) is generated directly, taking into account the dynamic constraints of the system. In our approach, however, we split the problem into two parts. First, a geometrical description of a 3D path is

[10]DIAPM Homepage: `http://www.aero.polimi.it/`

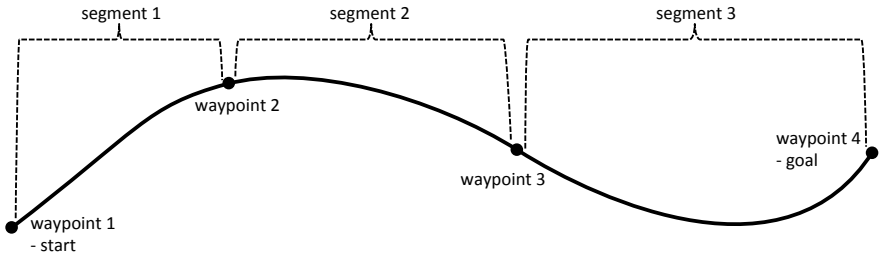


Figure 2.6: Example path consisting of three segments.

generated (e.g. by a path planner, Figure 2.7, part A). Then, the dynamic constraints are enforced by the path following control mode (Figure 2.7, part B) which uses the kinematic model of the platform and additional constraints (e.g. maximum acceleration). For example, if the path has a tight curve the controller will adapt the helicopter velocity in order to satisfy the platform dynamic constraints.

Using such a separation has two advantages: efficiency and robustness of the tracking performance. Not all of the dynamic constraints have to be imposed on the path planning level. This decreases the time required to generate a collision-free path because the state-space in which the search is performed has a lower dimension. It is relatively easy to enforce the dynamic constraints on the controller level, where a feedback loop is used. The controller will not only impose the constraints of the platform but will also take care of the external disturbances such as the wind.

In order to navigate safely in urban environments with dense obstacles, the PFCM mode has been designed to minimize the tracking error during path execution, that is, to follow the generated path as closely as possible. This is especially important because paths generated by the planner are collision-free (relative to the static obstacles present in the model), and staying closer to the geometric path assures safer navigation in the environment.

A path, for example generated by the path planner, 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 geometry of the current segment and the desired cruise (V_c) and final (V_f) velocities. Its output consists of four control signals (pitch, roll, yaw and the vertical channel – throttle). Additionally, the function returns a set of status flags which are used to coordinate the path segment switching mechanism and safety braking procedure.

A safety braking procedure is activated in case the next segment is not provided by the high-level system (e.g. due to communication failure) before a specific point in time. This time point is calculated using the minimum distance necessary to stop the helicopter at the end of the current segment

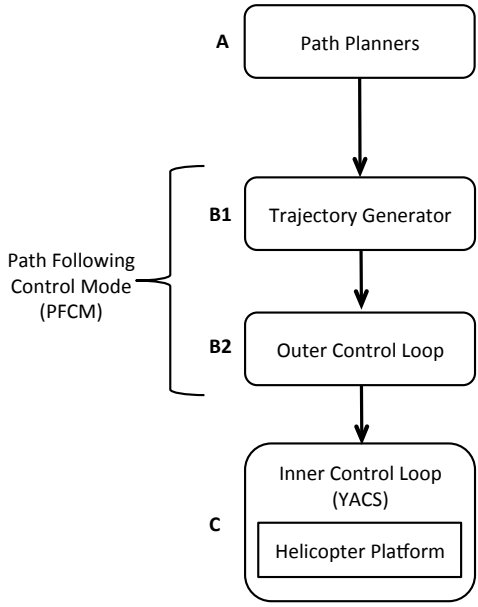


Figure 2.7: The path following control mode and its components.

without overshooting, thereby ensuring that even if a segment never arrives, the helicopter never leaves the designated path.

The path tracking error is also available and 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 be able to stay on the desired path.

The path following control mode is conceptually divided into two parts: (a) the *Trajectory Generator* (Figure 2.7, part B1) which calculates the reference trajectory used by (b) the *Outer Control Loop* (Figure 2.7, part B2) to calculate the control signals (i.e. pitch, roll, yaw and the vertical channel – throttle). We consider each part in the next two subsections.

Trajectory Generator

As previously described, the input to the path following controller is a set of parameters describing the geometry of a segment as well as the cruise and final velocities. The role of the trajectory generator is to calculate the reference position, velocity and heading which is fed to the outer control. Those parameters are calculated in the following way. First, based on the geometrical segment description an analytical expression of the 3D path is calculated. Second, the feedback algorithm calculates the control point on the path. A similar approach is used by Egerstedt et al. [27]. The control

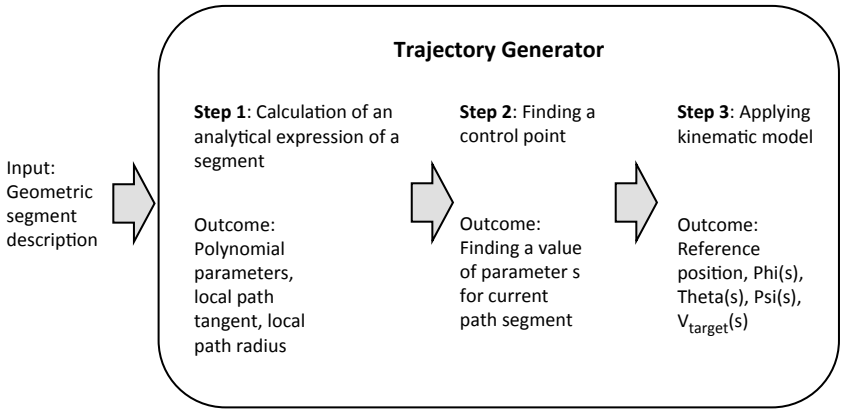


Figure 2.8: Calculations performed by the trajectory generator.

point is the point laying on the path where the helicopter ideally should be. Then, based on the kinematic model of the helicopter, the reference input to the outer loop is calculated. Figure 2.8 presents the three steps of calculation performed by the trajectory generator.

Step 1: Calculation of an analytical expression of a segment.

The geometric path segment (Figure 2.9) is represented by a parameterized 3D cubic polynomial. The choice of this type of path representation is motivated by several good characteristics of cubic curves, for example, flexibility, possibility of obtaining C^2-continuity at end-points and analytical solvability (which is used for collision checking by the path planning algorithms presented in chapter 4) [74].The following equation in vector form is used:

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

where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{D}$ are 3D vectors defined by the boundary conditions, the path segment parameter s, and $\vec{p} = [x, y, z]$. The boundary conditions are defined as 12 parameters: the starting point coordinates $\vec{p}(0)$, the end point coordinates $\vec{p}(1)$ and two vectors $\dot{\vec{p}}(0)$ and $\dot{\vec{p}}(1)$. The vectors represent the direction of the segment tangent at the starting $\dot{\vec{p}}(0)$ and the end point $\dot{\vec{p}}(1)$. The value of the parameter s ranges from 0 to 1, where $s = 0$ corresponds to the $\vec{p}(0)$ starting position and $s = 1$ corresponds to the $\vec{p}(1)$ end point of the same segment.

The following matrix notation is used for a description of a path segment $\vec{\boldsymbol{p}}(s)$:

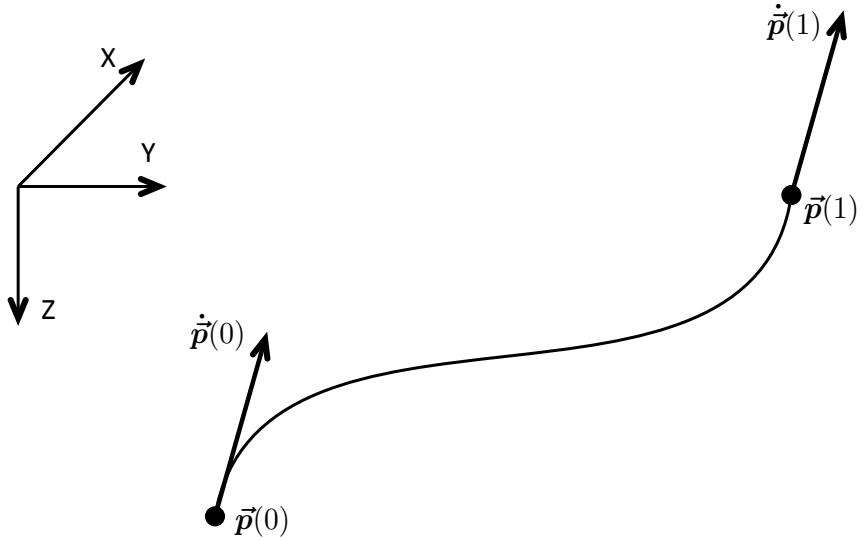


Figure 2.9: Boundary conditions for a path segment.

$$\vec{p}(s) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{p}(0) \\ \vec{p}(1) \\ \dot{\vec{p}}(0) \\ \dot{\vec{p}}(1) \end{bmatrix}$$

Additionally, for control purposes, the local path tangent $\vec{t}(s)$ and the path curvature radius $\vec{r}$ need to be calculated.

The local path tangent $\vec{t}(s)$ is defined as:

$$\vec{t}(s) = \begin{bmatrix} 3s^2 & 2s & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{p}(0) \\ \vec{p}(1) \\ \dot{\vec{p}}(0) \\ \dot{\vec{p}}(1) \end{bmatrix}$$

The curvature radius is defined as $\vec{r} = 1/\vec{k}$, where $\vec{k}$ is the path curvature defined as:

$$\vec{q}(s) = \begin{bmatrix} 6s & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{p}(0) \\ \vec{p}(1) \\ \dot{\vec{p}}(0) \\ \dot{\vec{p}}(1) \end{bmatrix}$$

$$\vec{k}(s) = \frac{\vec{t}(s) \times \vec{q}(s) \times \vec{t}(s)}{|\vec{t}(s)|^4}$$

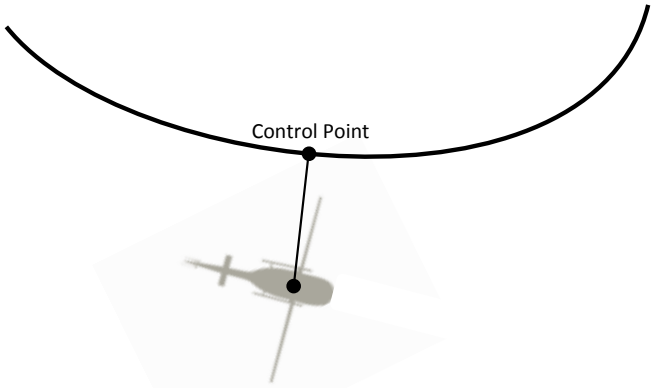


Figure 2.10: Control point on the reference path.

Once the parameter s is found the position coordinates (x, y, z), the local path tangent, and the path curvature radius can be calculated.

Step 2: Finding a control point.

In the previous step a set of parameters describing the current path segment has been calculated, thus defining the reference path. Note that the reference path is not time dependent and is parametrized by the value of the parameter s. In this case the control is realized by slowing down or accelerating the control point (the position on the reference path where the helicopter ideally should be, Figure 2.10).

The control point is defined as the closest point on the reference path to the current helicopter position. Although, a simple geometrical method of orthogonal projection of the helicopter position on the path could be used, it could result in multiple solutions. Instead a feedback method is used [20]. It finds the control point incrementally by performing a search for orthogonality condition only locally based on the previous value of s.

Step 3: Applying a kinematic model.

When the value of the parameter s is found a set of parameters needed by the outer control loop can be calculated. First, the reference position, the local path tangent and curvature radius are calculated. Second, a kinematic model of the helicopter is used to calculate roll ($\phi(s)$), pitch ($\theta(s)$) and yaw ($\psi(s)$) target angle values, thus applying dynamic constraints of the system. For safety, additional limits are used for each of the control channels (e.g. roll angle limit). Finally, the target helicopter velocity, that is the desired helicopter velocity along the segment ($V_{tar}(s)$), is derived.

The value of $V_{tar}(s)$ is based on the helicopter kinematic constraints

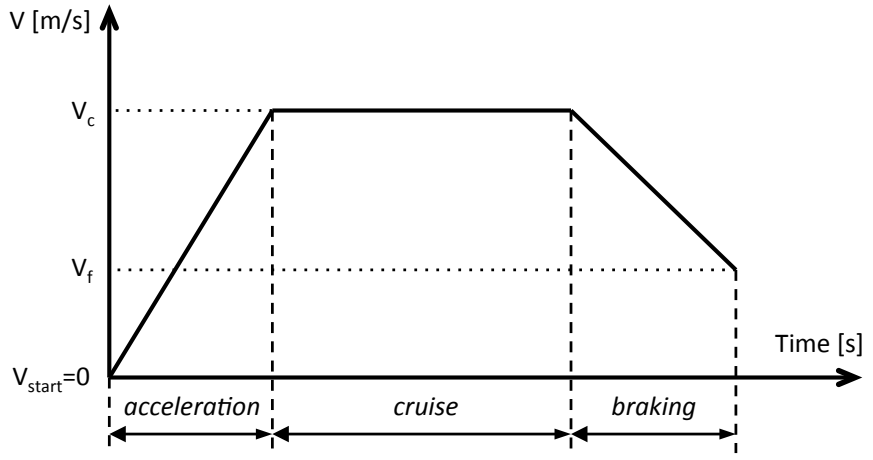


Figure 2.11: Example of an ideal velocity profile for a straight line path.

safety limits and the two velocity input parameters, i.e. 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).

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 the 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 activated when the remaining path length $d_{end}(s)$ is equal to the distance required to brake the helicopter from V_c to V_f for a 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 segments. Figure 2.11 presents an example of an ideal velocity profile for a straight line path.

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

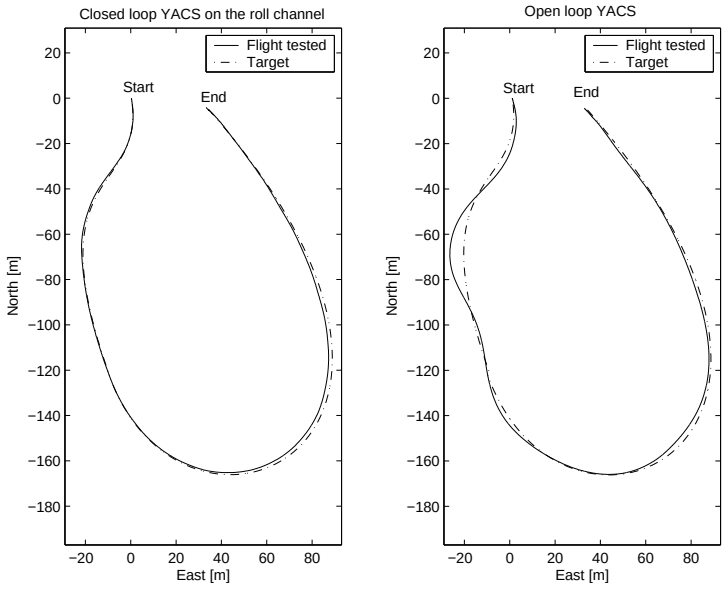


Figure 2.12: Comparison of path tracking performance using two different roll controllers performed at 36 km/h constant velocity.

must compensate the centripetal acceleration with a certain amount of roll angle. For safety reasons, the maximum roll angle (ϕ_{max}) and yaw rate (ω_{max}) are limited. Therefore, the maximum velocity during a turn maneuver is also restricted. The two velocity limits are calculated as follows:

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

where $R_{xy}(s)$ is the projection of the curvature radius vector ($\boldsymbol{r}^n$) on the XY horizontal plane.

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

Outer Control Loop

The outer control loop is implemented as four PID loops. Two versions of the outer loop, with and without a lead compensator in the roll control loop, have been developed and tested in real flights. A detailed description of the outer control loop variants developed are provided by Conte [20].

Summary

The path following control mode (including two versions of the outer 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 Figure 2.12 where the same path was tested with both versions of the outer control loop. Note that at the beginning of the path when the dynamic response of the controller is more important because of the changing curvature, the outer controller with a lead compensator in the roll loop (*extended controller*) performs much better than the other one. The error for the *extended controller* in this part is below 1 meter, while for the other it is around 6 meters. The results obtained during flight-tests show that the loop on the roll channel reduces the path tracking error, which makes the controller more suitable for obstacle-cluttered environments.

In summary, this section presented the path following control mode used for the execution of 3D paths consisting of a set of segments.

Chapter 3

Hierarchical Concurrent State Machines

In this chapter we provide a description of the Hierarchical Concurrent State Machine (HCSM) framework used for modelling hybrid control systems. The description includes the HCMS syntax and semantics, and provides a set of practical examples of HCSM use in the UASTechLab RMAX UAV platform. The chapter finishes with a summary referencing the Extended State Machines (ESM) framework which is a further development of the HCSM framework.

3.1 Introduction

In simple system designs, a procedural or object-oriented programming language can be used to solve the problem of control flow. This is done by using control flow statements included in a specific language (e.g. loops and conditional statements in the C programming language) or by directly calling functions in an appropriate sequence. However, as the complexity of the system grows it quickly becomes difficult to manage the necessary interactions between the different functions or objects. Thus, a more flexible mechanism for developing software systems for a robotic platform is required.

Using the abstraction of *state machines*, specifically Finite State Machines (FSM), for building systems is a well known and established approach. They help to control the complexity of the design and to provide a simple understandable means for solving complex modelling problems.

FSMs are a mathematical abstraction used to create a behavior model of a system. They are composed of a finite number of states, transitions between those states, and actions. Additionally, transitions can be guarded by conditions. For example a condition can be an evaluation of a sensor value. The input to a state machine can be realised by using a string of

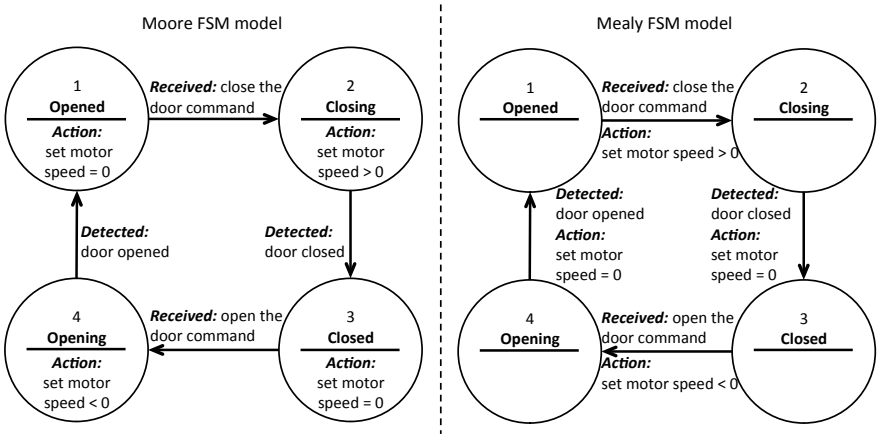


Figure 3.1: An example of a simple four-state system for an elevator door using Moore and Mealy FSMs.

symbols, events or commands depending on the implementation of the FSM system.

There is a wide variety of visual formalisms that have been proposed in the literature which have contributed to the popularity of state machines. The ability to visually model complex systems eases the development process.

An example of a simple four-state system for an elevator door using state diagrams [9] is presented in Figure 3.1. Two FSM paradigms are used in the example, Moore [64] and Mealy [70] machines.

In the Moore FSM, actions (outputs) depend only on the current state of the machine, as opposed to the Mealy FSM where actions are associated with both the input and the current state and are executed in transitions. In the example system, the door can be in one of the four states *opened*, *closed*, *opening* or *closing*. Depending on the current state and the received input the machine reacts with an appropriate action to either close or open the door passing through the intermediate states of opening or closing. Actions control the speed of the motor responsible for door motion.

Standard FSMs (e.g. Moore and Mealy machines) have been successfully used in the past for modeling systems for various purposes. However several extensions are necessary to make them useful for modelling complex real-time systems.

One of the major problems of FSMs is their limited expressivity mainly caused by their flat or single-level structure. There is no easy way of partitioning the problem into sub-problems that can be dealt with separately. Instead, each state has to be modelled explicitly leading in the worst case to a combinatorial explosion of the number of states. Additionally, it makes the system hard to extend. Because of these reasons, even moderately complex

systems tend to become large and unmanageable.

In practice it would be much easier to define a system gradually with different levels of granularity or with the help of various levels of abstractions. An obvious additional advantage would be the reusability of already modelled parts of the system in different contexts.

All of this can be achieved by HCSMs which extend FSMs with a support for hierarchy and concurrency. Before describing the HCSM framework we will briefly mention some of the related work which influenced our design.

The related work can be found in many fields: robotics, control theory, embedded system design, and software engineering. We looked at existing methods and included them in the framework if they turned out to be useful in practice for building robotic systems. There are many specification languages, programming languages, software frameworks, and design tools which are used in control or embedded system design, for example Ptolemy II [28], Esterel [5], Stateflow[1], Simulink[2], UML 2 [8], among others, but none of these is optimized for building control systems for autonomous robots.

Some of the important features required for the design and development of hybrid control systems for autonomous robots include [65, 67]: easy modelling of control flow, support for communication, real-time execution, tight integration between modelling formalism and execution mechanism, ability to reconfigure a system at run time, a well defined specification language, support for all the stages of development.

The specification language and the computation model we propose is mainly influenced by Harel's Statecharts formalism [38]. State machine based approaches have already been used successfully in many robotic systems. Brooks [10] for instance uses state machines to build reactive systems and Kleinehagenbrock et. al. [45] include them in a deliberative/reactive system. Albus et. al. [1] propose an architecture for intelligent hybrid control systems which has some similarities with our framework. It also includes state machines, defines a hierarchy of functional modules and includes a communication system, but it lacks some of the features mentioned above. Our framework supports the component-based design methodology. In [11] a component-based framework is proposed which aims for similar goals but it also does not provide some of the features mentioned above (e.g. real-time aspects). Cremer et al. [21] propose a HCSM framework for virtual environment applications. Although some similarities in the modeling formalism exist it is not associated with the framework presented in this thesis.

As previously described, the HCSM approach is based on the concept of a deterministic finite state machine (FSM) with several extensions. The following list is a compressed description of the extensions which are discussed later in this section:

- Supports hierarchy and concurrency. It is a combination of Mealy [64]

[1] MathWorks, Inc. Homepage: `http://www.mathworks.com/products/stateflow/`
[2] MathWorks, Inc. Homepage: `http://www.mathworks.com/products/simulink/`

and Moore [70] machines with vertical and horizontal (AND/OR) decomposition, where Mealy machines have actions (outputs) associated with transitions and Moore machines have actions associated with states. Hierarchy is realized by vertical state machine decomposition and concurrency by horizontal.

- Realized as a real-time interpreter of a textual state machine description.
- Semantics similar to Statecharts/MATLAB Stateflow but:
  - no history, i.e. when entering a state machine at a particular level the machine always starts in the init state.
  - no hierarchy crossing transitions; Transitions between different levels are realized by exiting to a super state. Transitions between different regions are realized using events.
  - no actions associated with states except those of a nested state machine.
- Time step event support (realized by a so called *pulse event*).
- Macro step/micro step execution, where external events are considered in the macro step and internal events in the micro step.

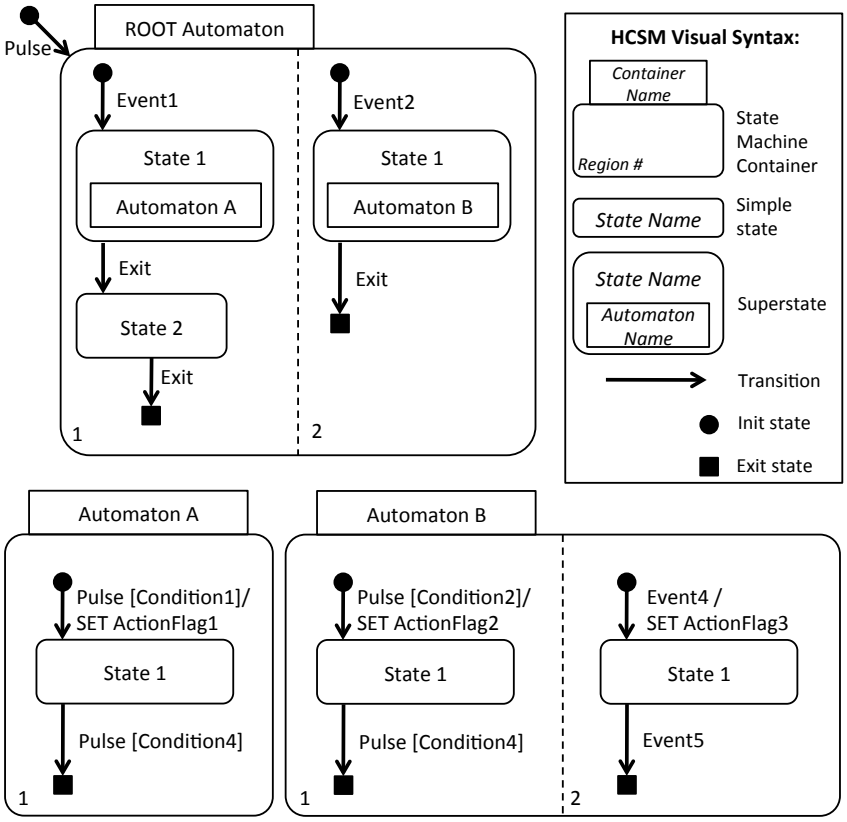
3.2 HCSM Framework

This section provides a detailed description of the HCSM framework. First, a set of basic concepts is described using the visual syntax of the HCSM state diagrams (Figure 3.2). Second, a description of the HCSM execution mechanism in a robotic platform is provided.

State machine, hierarchy and concurrency

A *state machine* (also referred to as an automaton) consists of states and transitions. A state represents any activity of a system at any level of abstraction. We define two main types of states: *simple states* and *superstates*. A superstate represents a nested state machine, thus a hierarchy of state machines can be created. Two special types of simple states are defined: *init* and *exit*. The init state of an automaton is the starting state and entering the exit state terminates its execution. An example of a simple and a superstate is presented in Figure 3.2, i.e. *State 2* and *State 1* of the root automaton, respectively.

We define a *state machine container* as a collection of one or more concurrent (child) state machines. Each of these machines is contained in a *region*. Regions are ordered by a consecutive number (RegionNumber) and



Syntax of a transition in EBNF notation:

Transition = EventID ["[" Guard "]"] ["/" ActionList]
ActionList = Action {"," Action}
Action = SET FlagID | UNSET FlagID | GET MemorySlotID | PUT MemorySlotID |
SEND EventID MemorySlotID DestinationID
Guard = ["¬"] FlagID | Guard Op Guard | "(" Guard ")"
Op = "∧" | "∨"
(* all *ID* symbols are ANSI C id-tokens *)
(* all *Name* symbols are sequences of ASCII characters *)

Figure 3.2: The HCSM visual syntax with an example of three state machines.

are separated by a dashed line in the visual syntax, e.g. *Automaton B* containing two regions in Figure 3.2.

The hierarchy provides a powerful mechanism for encapsulating specific behaviours and the framework permits reuse of state machine containers, i.e. it provides all the necessary means to execute multiple instances of a state machine, thus providing reusability. Additionally, such an encapsulation provides a practical way of aborting a particular behaviour.

Concurrency prevents combinatorial explosion of states, which would occur in FSMs, and allows for easy handling of asynchronous state machines.

One example which takes advantage of concurrency and hierarchy in the UASTechLab RMAX UAV is the modelling of complex behaviors such as vision-based landing [66]. The landing mode consists of several lower level behaviors that are controlled by the main superstate of the landing mode. It includes for instance control laws steering the helicopter and coordinates the camera system and image processing functionalities. When the landing behavior is activated, several state machines modelling the necessary activities are executed. Those include searching for a pre-defined pattern with the camera system, feeding a Kalman filter with image processing results which fuses them with the helicopter's inertial measurements. Once the pattern is found another state machine controls the camera in order to keep the pattern in the center of the image. This tracking behavior increases the robustness of the image processing results when the helicopter is close to the ground or in the presence of strong wind gusts. The state machine sends appropriate feedback when the landing procedure is finished or it has been aborted.

State Transitions, Events and Guards

A transition between two states is triggered by an *event*. Events in the HCSM framework can be generated internally by the state machine itself or externally. The HCSM supports both asynchronous (or sporadic) and periodic events. The sporadic events can contain data.

The HCSM implements two types of special events, namely the *pulse* and *exit* events. The pulse event is a periodic event generated before each iteration of the HCSM algorithm, like a clock pulse (discussed later). This event can be used for example to trigger a transition without a specific asynchronous event. The exit event is created when a state machine is in its exit state and it is only sent to the parent superstate informing it that a child automaton has finished its execution.

State transitions are guarded by *conditions* in the form of Boolean expressions. If an event for a particular state transition has been generated and the condition guarding the transition is TRUE, then the state transition takes place. It is optional to have conditions in transitions.

Activities vs Actions

As in Harel [38] we distinguish between actions and activities. Actions have no duration (zero-time assumption) and are executed in transitions (as in a Mealy machine), while activities take time and are associated with states (as in a Moore machine). Each transition can include a set of *actions* executed during the transition. Supported actions include setting binary flags (SET *flag name*), sending events (SEND *event-name data target-computer*), retrieving (GET *source*) and storing (PUT *destination*) data. Handling of data is realised using a predefined memory bank with labeled slots. If a received event carries data it is automatically stored in the *cache* memory slot. A GET action copies data from a source slot to the cache slot. A PUT action copies data from the cache slot to the destination slot. It is optional to have actions in transitions.

Activities are defined in terms of regularly executed functions. They are coordinated by a set of binary flags which are changed by actions. Functions are executed outside the state machine algorithm and their execution is discussed in the next section.

HCSM Design and Execution

The visual language of the HCSM provides a very powerful development tool for modelling system behavior. The design of a system in the HCSM framework starts with a visual description of the state machines using the presented visual state diagrams (Figure 3.3). 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 on the robotic platform. Thanks to its compact and efficient implementation, the interpreter can run in the real-time part of the system as a periodic task with high execution rate.

The definition of HCSMs can be done with the aid of graphical tools or for simpler designs it is possible to write the text files directly.

Conditions used in the state machine design are implemented as C-code. They are usually generated from sensor data or control functions. The HCSM supports AND/OR/NOT operators in condition statements directly in the visual description and HCSM text files.

Although conditions and activities (in the form of function calls) have to be pre-compiled in the state machine executable program, in practice it does not pose a major problem. From our experience with our UAV systems and other robotic platforms developed in the UASTechLab there is a fixed number of necessary conditions and function calls that have to be written in the system in the beginning of the development. The modelling of the system behavior itself can then take full advantage of the flexibility of the interpreted state machine language, i.e. necessary changes are only made in the text file and can be applied without recompilation of the whole system.

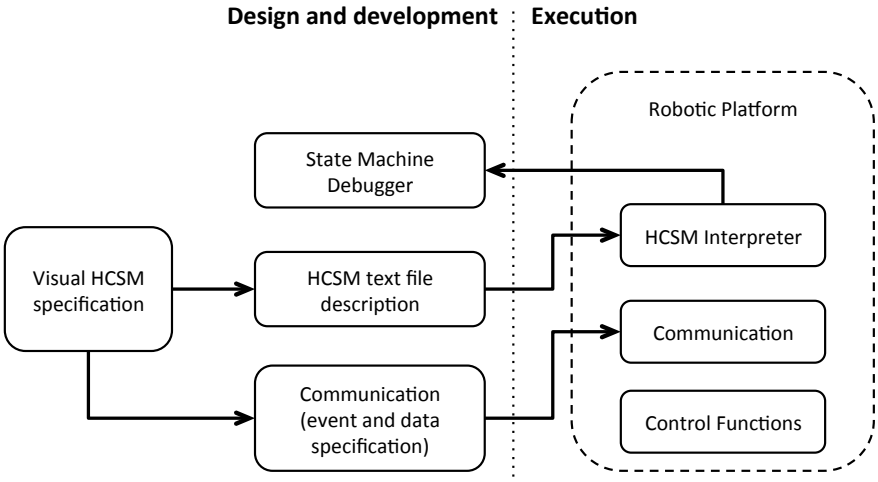


Figure 3.3: Overview of the HCSM design process and execution.

In our framework three forms of communication exist: (1) between states of the HCSM language processed by the same interpreter, (2) between computer systems of the same robot, and (3) between different robots or robots and operators. The first form is realized by an *internal event queue*, and the remaining two forms by transmitting external events in packets with predefined sizes. Received external events are put in an *external event queue*.

The framework provides the necessary means for sending packages with a predefined number of bytes to remote machines, receiving packages, and checking their integrity. It permits building real-time communication applications if the underlying network is suitable. Based on the description of the user defined events the necessary C-code implementation is automatically generated during the design process (Figure 3.3). Those functions are then used in the system in order to receive and send events. We successfully built real-time communication links based on serial line standards (RS232C and similar) and Ethernet.

Before describing the algorithm for execution of HCSMs a short example describing the outer execution loop is provided. The skeleton of a procedure for execution of a HCSM-based system on a single computer is presented in Algorithm 3.2.1.

As described previously, functions used for communication are generated automatically and are called in the *Communicate* function. Events received by the system are parsed, checked for integrity and put in the external event queue. The HCSM is executed (*ExecuteStateMachine*). Binary flags which are set by the state machine transitions (using actions) coordinate the execution of the control functions (*runControlFunctions*). The user has to define C-code for conditions and implement the *runControlFunctions* procedure.

Algorithm 3.2.1 Skeleton of a procedure for execution of a HCSM-based system on a single computer.

– *Main Execution Loop*

```
1: while system is running do
2:     ...
3:     Communicate();{Send and receive data packets containing debug
       information and external events. Events are put in the external event
       queue.}
4:     ExecuteStateMachine();{Execute algorithm 3.2.2}
5:     runControlFunctions();{Run the appropriate control functions
       based on the binary flags set by actions in the HCSM.}
6:     ...
7: end while
```

Figure 3.4 presents an example of a simple HCSM modelling a driving behaviour of a ground robot platform. An example of the necessary C-code implementation is presented on the right side of the figure. The platform is equipped with a proximity sensor mounted in the front which provides distance measurements to obstacles. When the *Simple Drive Automaton* is started, the robot is in the *init* state (solid dot) until it receives a *Drive Forward Event*. The event includes the desired forward speed value for the robot. Two actions are executed when the state machine switches from the init to the *Drive Forward* state, PUT drive-speed and SET start-motor-action-flag, respectively. The first action, copies data from the cache memory slot to the drive-speed slot. The second one, sets a binary flag that is used by the *runControlFunctions()* procedure. After the first iteration of the algorithm the robot starts to execute the driving behaviour. In the following iterations the state machine remains in the Drive Forward state while the *obstacle-detected-condition* is evaluated to FALSE. The C-code example presented checks if the measured distance to the obstacle is less than 1 meter. When the distance is below 1 meter the automaton switches to the *Stop* state and executes two actions switching off the driving activity and stopping the robot's motors.

The procedure used to execute a HCSM is presented in Algorithm 3.2.2. Transitions in HCSMs are divided into *macro steps* and *micro steps*. A macro step (Algorithm 3.2.2, *ExecuteStateMachine* function lines 4 to 10) begins by processing the first event from an external event queue. The events are processed in the order they were received (First-In-First-Out fashion). This event can trigger one or more transitions. These transitions may generate both internal and external events, which in turn trigger more transitions. The macro step is finished when no more transitions are made, i.e. the external event queue is empty. Micro step (Algorithm 3.2.2, *ExecuteStateMachine* function lines 6 to 9) are the steps within a macro step.

In the case the external event queue is empty only the micro steps are

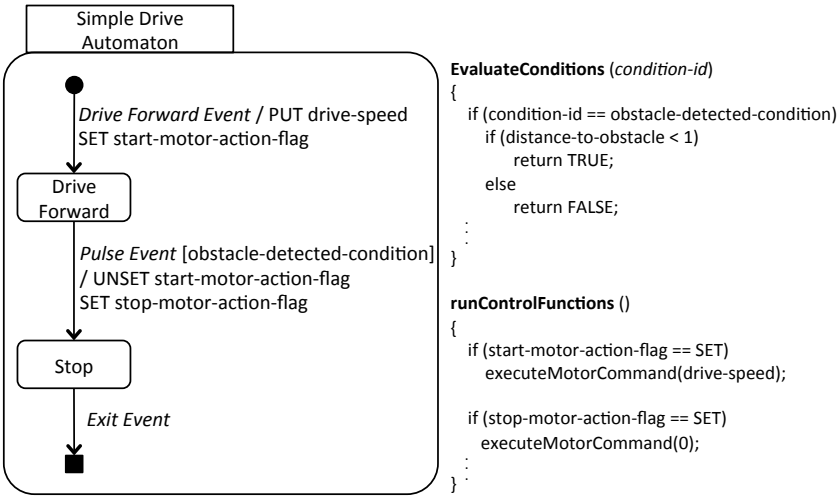


Figure 3.4: An example of a simple HCSM modelling a driving behaviour of a ground robot platform.

executed, processing all of the internal events (at the beginning one pulse event is in the internal event queue).

The system assumes the "synchrony hypothesis" i.e. during a macro step, inputs do not change and external events are not received. In practice, external events are not processed as soon as they arrive but they are buffered until the state machine interpreter is called. In the case of our UASTechLab RMAX system, in the real-time environment it is called periodically every 20ms, and in user space the state machine interpreter is called from the main loop. The duration of one *macro step* is set to 200us which corresponds to worst case execution time. This time depends on the complexity of the state machine, i.e. the number of regions and the maximum number of *micro steps* (generated events). The periodic update time and the duration is user configurable and was empirically chosen for our system.

3.3 Practical HCSM Examples

In this section we provide some practical examples of the use of HCSMs in the UASTechLab RMAX platform. As previously described (section 2.1), our platform uses three computer systems. Each computer is responsible for a specific functionality. The PFC computer is running basic flight control modes, the IPC is responsible for image processing and control over the sensor platform and its payload (i.e. cameras, pan/tilt unit) and the DRC is responsible for running high-level deliberative services.

The HCSM framework is used as a low-level real-time mechanism for

Algorithm 3.2.2 The HCSM Algorithm.

– *ExecuteStateMachine*

```
1: lock memory
2: create empty internal event queue
3: append pulse event to internal event queue
4: repeat
5:    remove first event from external event queue and append to internal
      event queue
6:    while internal event queue not empty do
7:       remove first event e from internal event queue
8:       call MakeTransitions(1, e)
9:    end while
10: until external event queue empty
11: unlock memory
```

– *MakeTransitions(MachineLevel, Event)*

```
1: for all concurrent state machines M at MachineLevel do
2:    if Event is received by transition of current state in M and Guards
      are TRUE then
3:       call ExecuteActions(actions associated with the transition)
4:       make transition
5:    else if current state is superstate then
6:       call MakeTransitions(MachineLevel+1, Event)
7:    end if
8: end for
```

– *ExecuteActions(ActionList)*

```
1: for each action Action in ordered ActionList do
2:    if Action is send event action then
3:       if destinationID is external computer then
4:          append Event to external communication queue
5:       else
6:          append Event to internal event queue
7:       end if
8:    else
9:       execute action
10:   end if
11: end for
```

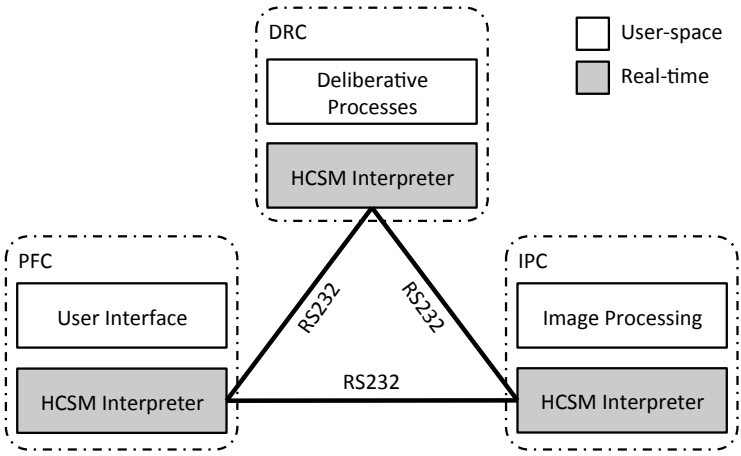


Figure 3.5: Overview of the HCSMs used in the UASTechLab RMAX UAV platform.

modelling of a system behaviour. On the PFC computer it is used for modelling and execution of the control system. On the IPC computer it is used for controlling a set of sensors and on the DRC computer it provides an interface to flight control modes accessible to the high-level deliberative and reactive algorithms. That is why HCSM interpreters are running on all three computers (Figure 3.5). Because the HCSMs running on different computers communicate with each other, all of the individual events used in the system are defined globally and have unique IDs.

Examples described in this section are focused on the state machines running on the PFC computer. An overview of all state machines running on the PFC system is provided, followed by a detailed description of two use cases: handling the switch between manual and autonomous flight mode and the execution of the path following control mode (PFCM, section 2.2.3).

Overview of the PFC HCSMs

Figure 3.6 presents a hierarchical view of all of the 15 automata that are running on the PFC computer. The whole UASTechLab RMAX system uses 207 events in total. Note that the two use cases presented in this section are examples from the deployed system and various extensions and different way of modelling to achieve the same functionality of the system is possible.

The software-controlled power system (section 2.1) is managed by the PFC computer which has to be switched on manually by the user. The rest of the devices can be switched on and off by software using the provided interface coordinated by the *Power Switch* automaton. At the system start-up, the *Root* automaton makes sure that at least the DRC computer is

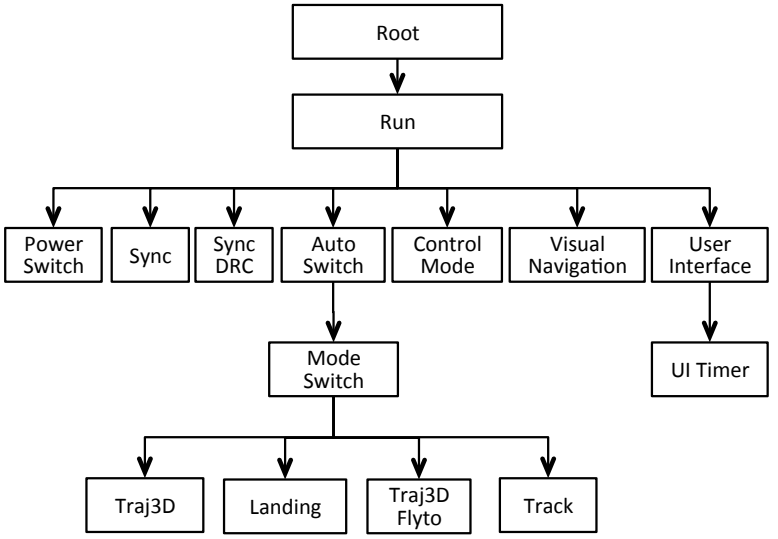


Figure 3.6: The hierarchical view of the HCSM automata running on the PFC computer.

switched on (in case it has not been manually switched on by the user). The *Root* automaton also contains a superstate for the *Run* automaton.

The *Run* automaton contains only one superstate for all of the following automata: *Power Switch, Sync, Sync DRC, Auto Switch, Control Mode, Visual Navigation, and User Interface*.

The *Sync* and *Sync DRC* automata are responsible for achieving a common state between the different computers through synchronization. The common state includes, for example the time and altitude offset. The *Sync DRC* automaton handles the synchronization between the IPC and DRC computers. The *Sync* automaton sends the common state from the PFC to both IPC and DRC computers.

The *User Interface* automaton handles commands received from a ground control user interface (UI) or a miniaturized keyboard. A number of UIs have been developed for the UASTechLab RMAX system. One of the interfaces used during every flight test is a low-level telemetry monitoring interface that is handled by the PFC computer. A ground operator monitors the telemetry data and in case of any abnormal status the mission can be aborted either by executing an emergency brake and default hovering or by the backup pilot who can take over the control of the helicopter and perform a manual landing.

The *UI Timer* automaton implements timeouts for accepting commands by the *User Interface* state machine when using a miniaturized keyboard. The keyboard has been constructed to provide a basic set of utility commands (e.g. starting of logging of the onboard data). Although, the same

functionality is available through a standard UI, it is often convenient (e.g. during a flight test) to be able to use it without the need for running an interface on the ground station computer. The keyboard is mounted in the helicopter avionics box. Selection of a specific utility command is performed by using a digital knob (selecting a number from 1 to 15) and a push button for acknowledging the selected command. Feedback from the system is provided on a small display mounted outside of the helicopter's avionics box.

The *Visual Navigation* automaton handles a non-GPS vision-based navigation module [20] which is running on the IPC computer.

Handling of the helicopter initialization and operational modes is done by the *Auto Switch* automaton. This includes the following aspects:

- Initializing the RMAX helicopter system before it is ready to accept commands. This includes monitoring the engine-on status and the RC radio transmitter status.
- Monitoring sensor data and initializing the state estimation based on a Kalman filter.
- Handling the simulation mode. A hardware-in-the-loop simulator has been developed for the UASTechLab RMAX UAV. It uses a dynamic helicopter model [17, 26] and all of the software components used during a real flight are run onboard the UAV. This enables testing of the developed functionalities before a real flight test is performed.
- Handling the RC radio transmitter switch responsible for selecting between the operational modes: manual/autonomous flight mode.

The *Auto Switch* automaton starts up the execution of the *Mode Switch* state machine after a successful initialization of the system and if the autonomous flight mode switch is selected.

The *Mode Switch* and *Control Mode* automata handle the switching between the flight control modes. This includes initializing, running and handling of error conditions of particular control functions.

Selected control modes also have their own mode-specific state machines:

- *Traj3D*: the path following control mode.
- *Landing*: vision-based landing control mode [66].
- *Traj3D Flyto*: a simplified path following mode which only uses straight line paths.
- *Track*: reactive car tracking.

Other control modes such as take-off and hovering are modelled explicitly by the *Mode Switch* and *Control Mode* automata without any additional state machines.

Each control function, including the path following control mode, that runs on the UASTechLab RMAX UAV is executed periodically with a 50Hz update rate. The outer loop is implemented similarly to Algorithm 3.2.1 with one exception. After the execution of the function responsible for communication and the HSCM algorithm, the function for handling SET action flags is called. The function maps the action flags into a set of local variables (also referred to as internal flags) used in the execution of the actual control function. Additionally, it automatically unsets a particular action flag in C-code in order to avoid the inclusion of explicit UNSET actions in the state machine specification, as opposed to the example presented in Figure 3.4. Using this approach, complex state diagrams become simpler to understand.

The next two sections describe the following two use cases: *Engaging Default Autonomous Hovering Mode* and *Path Execution*. For clarity, diagrams presenting the *Control Mode* (Figure 3.7), the *Mode Switch* (Figure 3.8), and the *Traj3D* (Figure 3.11) automata use the following notation:

- For fast identification, each automaton has a letter label, included in the state machine name. Additionally, all state transitions are numbered.
- Names of states end with "-ST".
- Names of events end with "-EV".
- Names of conditions end with "-CO".
- Names of SET action flags end with "-AC".
- Names of labeled memory slots end with "-DO".

3.3.1 Use Case 1: Engaging Default Autonomous Hovering Mode

The use case described in this section relates to the following situation. A backup pilot performs a manual take-off procedure. When the maneuver is finished, the UAV is switched to autonomous mode by using the auto switch button on the RC radio transmitter. A default hovering function is engaged for the current helicopter position.

Two state machines are mainly handling this use case, namely the *Control Mode* (Figure 3.7) and the *Mode Switch* (Figure 3.8).

Before describing the interaction between the two automata in detail, an overview of some of the design decisions that were made during the development of the system is provided.

As previously described, a specific control function in the UASTechLab RMAX system is called periodically with an update rate of 50Hz. In each iteration, the system checks an internal flag and based on its value an appropriate function is executed. The particular flag is set by a state machine (using a SET action), e.g. for hovering Set Hover-Mode-AC (B.2.).

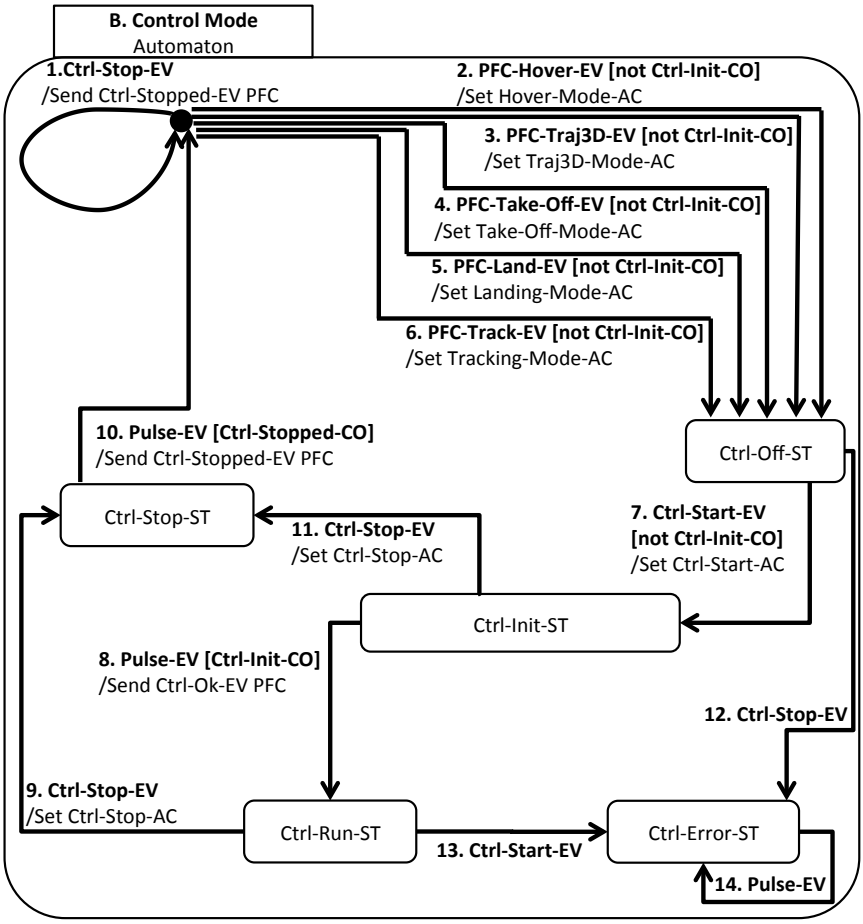


Figure 3.7: The *Control Mode* automaton.

Each control function implemented in the UASTechLab RMAX system follows a predefined template. Functions have three internal states and support two internal flags: an input and a status flag. The input flag is used to switch between the internal function states, i.e. initializing, running and stopping the execution of the function. The status flag is used for error handling, for example, when the initialization of the function fails. Both flags are used by the *Control Mode* automaton which models the internal states of the function, i.e. Ctrl-Init-ST, Ctrl-Run-ST and Ctrl-Stop-ST states. Additional states present in the automaton (i.e. Ctrl-Off-ST and Ctrl-Error-ST) are used to make sure the previous function has been properly stopped and no other functions are initialized.

The *Control Mode* automaton sets the appropriate input flag and keeps track of the control function status. Because each control function follows

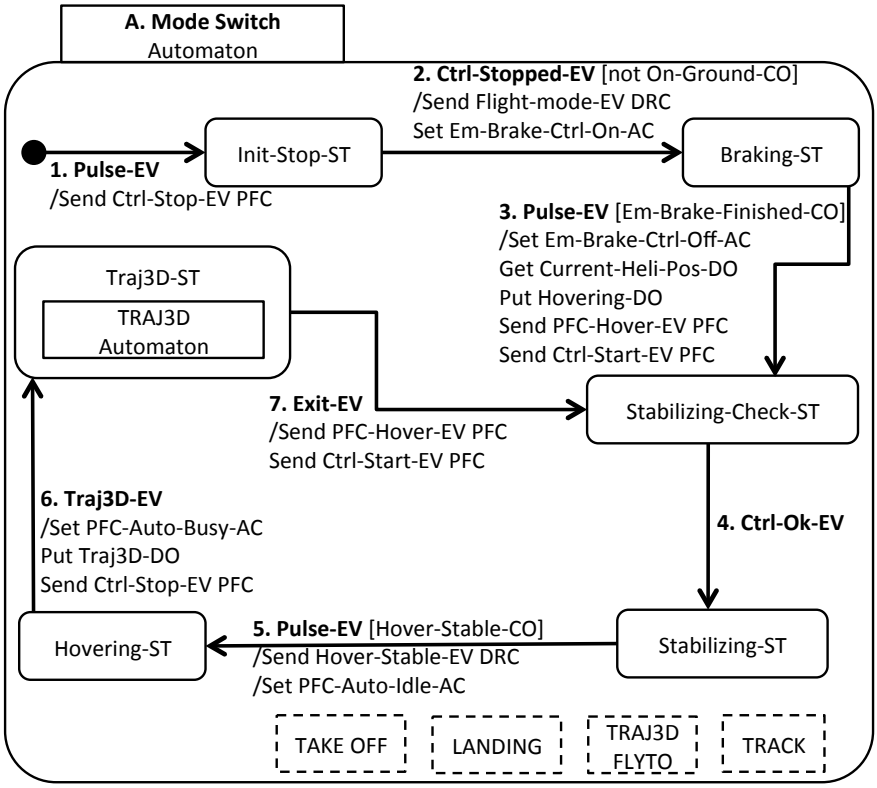


Figure 3.8: The *Mode Switch* automaton. Simplified view focused on a path following control mode execution.

the predefined template, the *Control Mode* state machine handles all modes transparently, i.e. there is no mode-specific state. When a particular control function should be executed, a state machine (e.g. the *Mode Switch*) sends a mode specific start event (e.g. PFC-Hover-EV) to the *Mode Control* automaton. This triggers one of the transitions B.2., B.3., B.4., B.5., or B.6. during which appropriate internal flags are set by executing SET actions (e.g. Set Hover-Mode-AC). In the next iteration of the outer loop (similarly to Algorithm 3.2.1) the control function starts its execution following, of course, the initialization and error handling states. If the initialization is successful, the *Control Mode* automaton switches its state to Ctrl-Run-ST and until the mode is stopped it is executed periodically in each iteration of the outer loop.

The *Control Mode* automaton (Figure 3.8) is mainly interacting with the *Mode Switch* state machine which implements the flight mode switching. This includes sequentialisation and coordination of control function execution. For example, the automaton ensures that after the path following

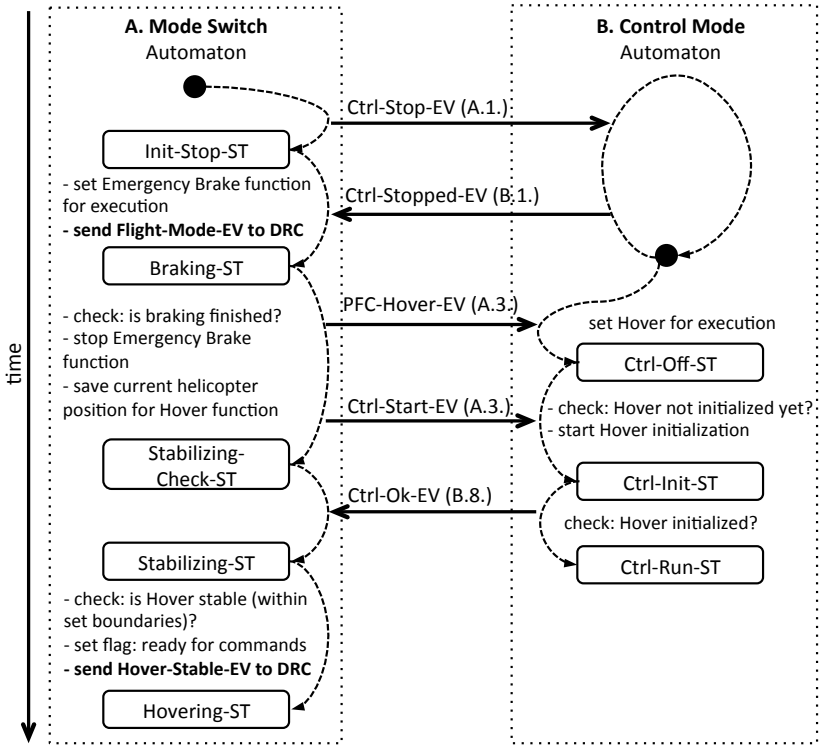


Figure 3.9: An execution trace for use case 1 showing the interaction between the *Control Mode* and the *Mode Switch* automata.

execution a default hovering mode is switched on.

The *Mode Switch* state machine, additionally, generates events that are sent to the high-level system (DRC), e.g. in the A.2. transition. It also reacts to events sent from the DRC, e.g. the A.6. transition.

Figure 3.9 presents the interaction between the *Control Mode* and the *Mode Switch* automata for use case 1. The time-line shows state transitions and events exchanged between the two state machines. For easy reference to the two state machine diagrams, the transition that generates a particular event is given in brackets.

The execution starts with an exchange of two events, Ctrl-Stop-EV (A.1.) and Ctrl-Stopped-EV (B.1.), to make sure no other control function is active. The *Mode Switch* automaton executes a braking procedure (Braking-ST). This procedure, also called emergency braking, is designed to stop the helicopter before a default hovering mode is engaged. The procedure uses the path following control mode with a predefined straight line path and a zero target velocity value as an input.

When the braking procedure is finished the current helicopter position

is saved to be used for the hovering function (i.e. Hovering-DO memory slot). Two events are also sent to the *Control Mode* automaton. The first event (PFC-Hover-EV (A.3.)) makes the *Control Mode* state machine set the hovering function for execution (i.e. Set Hover-Mode-AC). The second one (Ctrl-Start-EV (A.3.)) is starting the initialization procedure of the function. After a sucessfull initialization the Ctrl-Ok-EV (B.8.) is generated and the *Mode Switch* automaton changes its state to Stabilizing-ST. The automaton stays in this state until a Hover-Stable-CO condition is satisfied. The condition checks if the position, altitude, heading and velocity are within hovering tolerance bounds. The tolerances used in the UASTech system are set to: 5m for position, 2m for altitude, 5 degrees for heading, and 1 m/s for vertical and horizontal velocities.

When the Hover-Stable-CO condition is satisfied, a Hover-Stable-EV event is sent to the DRC computer and an internal system flag is set to indicate that the system is in the autonomous mode and ready to accept new commands (Set PFC-Auto-Idle-AC). The *Mode Switch* automaton changes its state to Hovering-ST.

3.3.2 Use Case 2: Path Execution

The use case described in this section assumes the UAV is already in the autonomous flight mode and the default hovering function is active (for example after executing the use case presented in the previous section). The description is focused on the execution of the path at the lowest control level running on the PFC computer. Before describing the details, an overview of the execution process from the perspective of a mission and all services involved is provided. The standard path execution scheme in the UASTechLab RMAX UAV for static operational environments is depicted in Figure 3.10.

A UAV mission is specified via a task procedure (TP) in the reactive layer of our architecture, perhaps after calling a task-based planner. A TP is a high-level procedural execution component which provides a computational mechanism for achieving different robotic behaviors (section 2.2).

For the case of flying to a waypoint, an instance of a navigation TP is created. It first calls the path planner service (step 1, Figure 3.10) 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, *Traj3D-EV*) of the trajectory via the control system interface and waits for the *Request-Segment-EV* event that is generated by the controller.

At the control level, the path is executed using a Path Following Control Mode (PFCM, section 2.2.3). When a *Request-Segment-EV* event arrives (step 4) the TP sends the next segment. This procedure is repeated (step

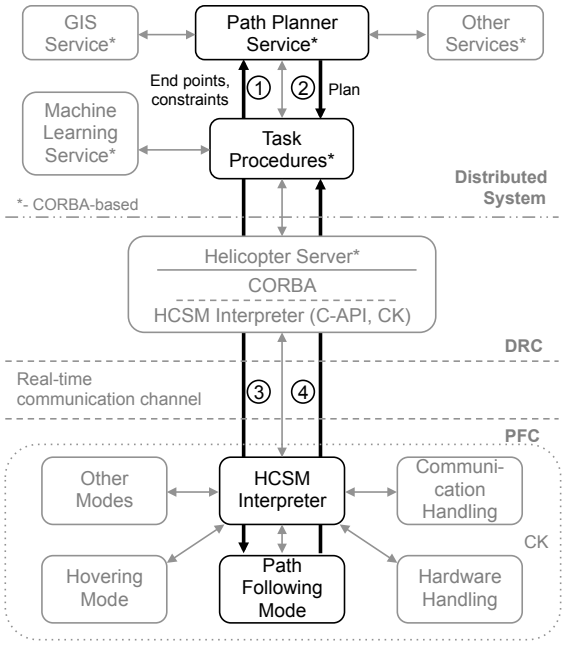


Figure 3.10: Path execution scheme.

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 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 presented path execution scheme shows the interaction between various components of the system providing an overview of the execution process. In the remainder of the section a detailed description of the execution from the perspective of the control system running on the PFC computer is provided.

As described, the path supplied by the navigation task procedure (generated by the path planner) is executed using the path following control mode. The execution on the PFC computer is coordinated by three automata, namely the *Control Mode* (Figure 3.7), the *Mode Switch* (Figure 3.8), and the *Traj3D* (Figure 3.11).

As in the case of hovering mode described in the previous section, the PFCM function implementation follows a predefined design template. The function itself is executed by setting an appropriate internal flag using a SET action (i.e. Set Traj3D-Mode-AC, B.3.) by the *Control Mode* automaton.

The *Mode Switch* state machine makes sure the default hovering function is properly terminated before the PFCM function can be activated. Additionally, when the path execution is finished it engages the default hovering

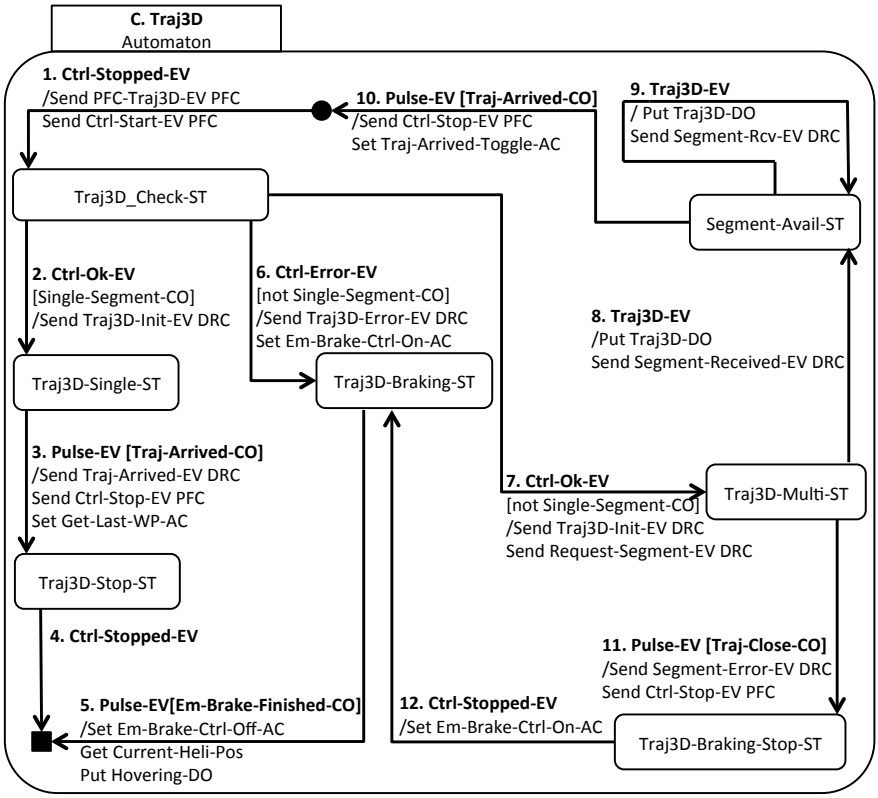


Figure 3.11: The *Traj3D* automaton.

function, similarly to the example shown in the previous section.

The role of the *Traj3D* automaton is to coordinate an exchange of events with other services (i.e. TP) and to assure that the appropriate segment data is available to the PFCM control function when needed. For example, if the necessary events containing a description of the next segment do not arrive on time, an emergency braking procedure is executed.

An example time-line for a path execution showing the interaction between the three automata is presented in Figure 3.12. The example focuses on a single segment execution.

The *Mode Switch* automaton is in the Hovering-ST state. After receiving a Traj3D-EV event from the DRC computer, the data describing a segment for execution is saved in the memory slot used by the PFCM function (i.e. Traj3D-DO). Additionally, the Ctrl-Stop-EV event (A.6.) is sent to the *Control Mode* automaton in order to stop the execution of the default hovering function. At this time point, the *Mode Switch* automaton transitions to the TRAJ3D-ST (Figure 3.11) state and remains in it until the PFCM execution is finished. Then it takes care of the switching to the default hovering mode.

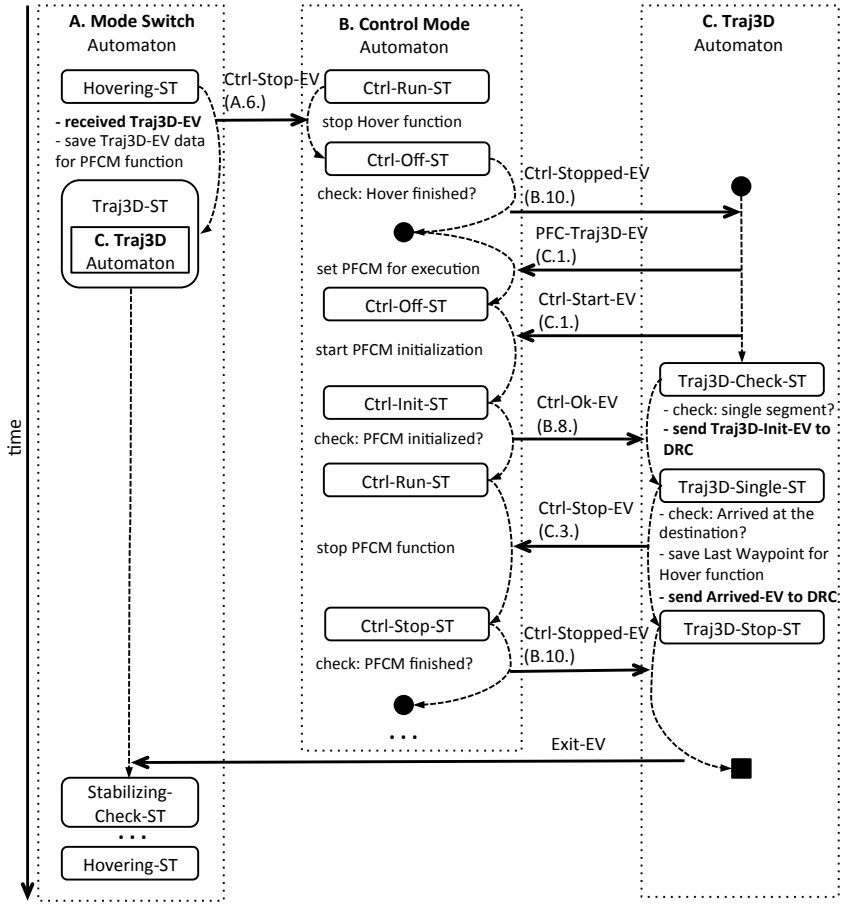


Figure 3.12: An execution trace for use case 2 showing the interaction between the *Control Mode*, the *Mode Switch* and *Traj3D* automata.

The TRAJ3D-ST state is a superstate for the *Traj3D* automaton effectively starting its execution in its init state.

The *Traj3D* automaton waits for the confirmation from the *Control Mode* state machine that the hovering function has been terminated (Ctrl-Stopped-EV (B.10.)) and sends two events back to initialize and start the execution of the PFCM control function (i.e. PFC-Traj3D-EV (C.1.) and Ctrl-Start-EV (C.1.)). It transitions to the Traj3D-Check-ST state waiting for the confirmation that the PFCM function has been initialized (Ctrl-Ok-EV (B.8.)). When the event arrives the condition if a single segment has been received for the execution is checked. A single segment is defined by the path parameters, i.e. the end velocity for the segment is set to zero. If the condition is satisfied the event informing the DRC computer that the

segment has been accepted for execution is sent.

At this time point the PFCM starts the execution of the segment and the *Traj3D* automaton is in the Traj3D-Single-ST state and remains in it until the Traj-Arrived-CO condition is not satisfied. In the UASTechLab RMAX UAV the segment has been successfully executed when the distance to the final waypoint is smaller than 3 meters. When the condition is satisfied the last waypoint of the path is saved for the hovering function and the Traj-Arrived-EV event is sent to the DRC computer informing it that the path execution is finished. Additionally the *Traj3D* automaton stops the execution of the PFCM function by sending the Ctrl-Stop-EV event (C.3.) to the *Control Mode* state machine. When the termination of the execution is confirmed by receiving the Ctrl-Stopped-EV event (B.10.) the *Traj3D* automaton transitions to its exit state and the *Mode Switch* state machine takes care of engaging the default hover mode, similarly to the example shown in the previous section.

3.4 Summary

In summary, this chapter presented a development framework for hybrid control systems used in autonomous robots. The HCSM framework uses hierarchical concurrent state machines as a modelling abstraction and allows the specification of the control flow of a software system in an easy visual formalism. Additionally, the visual state diagrams used during the design are transformed into an intermediate text format and executed directly on the robotic platform using a HCSM interpreter. Thanks to its compact and efficient implementation, the interpreter runs in the real-time part of the system. During the design process the user has to provide a minimal amount of C-code implementation for the control functions and conditions based on sensor data. The necessary functions for communication are generated automatically by the framework based on the user specification. The behaviour of the system can then be easily changed at a run time by editing the state diagrams and uploading the new text file to the HCSM interpreter. Examples of HCSMs used on the UASTechLab RMAX UAV were presented. The system has proven to be robust, reliable and easy to extend and we have used it in a number of autonomous missions through a period of several years.

The HCSM framework has been further developed and renamed to Extended State Machines (ESMs) [67]. The new framework adds several useful modelling features. The main changes include: explicit modelling of task states, data flow, control and system flags, an event filtering mechanism and no explicit external events. It also includes a visual tool for designing and debugging state machines, which makes the development of new and existing systems easier.

The ESM uses three types of states: *simple states*, *superstates* (as in the HCSMs), and *task states*. Control and other functions are modelled

explicitly in the task states. A schedule for function execution is provided by a scheduler included in the framework. The data used as input/output to the task state functions (data flow) is also explicitly modelled in the ESM formalism by *Data paths*. Asynchronous external events are modelled by a combination of *pulse event* and guard conditions, thus only one internal event queue is used in the ESM. Additionally the ESM introduces an event filtering mechanism which limits the scope of internal events. The full details are available in [67].

Chapter 4

Dynamic Path Replanning

This chapter presents a dynamic path replanning mechanism developed for the UASTechLab RMAX system. The first section provides background information on path planning algorithms. The basic concepts needed for understanding the path planning problem and algorithms are provided, followed by a description of the two sample-based planners used in the system.

The remainder of the chapter provides a detailed description of the replanning method together with experimental data.

4.1 Background

Path planning and motion planning algorithms deal with the problem of generating collision free paths or motions for a robot in order to navigate in an environment. Before defining a path planning problem, a description of some basic concepts is provided.

Three types of representations can be found in the literature for describing and solving path planning problems, namely workspace, configuration space and state space.

Workspace representation

The physical space in which the robot navigates is called the workspace $\mathcal{W}$. It is most often modelled as $\mathcal{R}^3$ but can be restricted to $\mathcal{R}^2$ - for example for robots navigating in a single plane (e.g. car-like robots in a 2D flat environment). This type of representation is particularly well-suited for collision checking since the robot and the obstacles are represented in the same space. However in many practical applications the workspace is not sufficient to describe the planning problem and a more expressive representation is necessary. In the case of our example car-like robot it is not sufficient to only consider the car position (p_x, p_y) but also its orientation ψ (Figure 4.1).

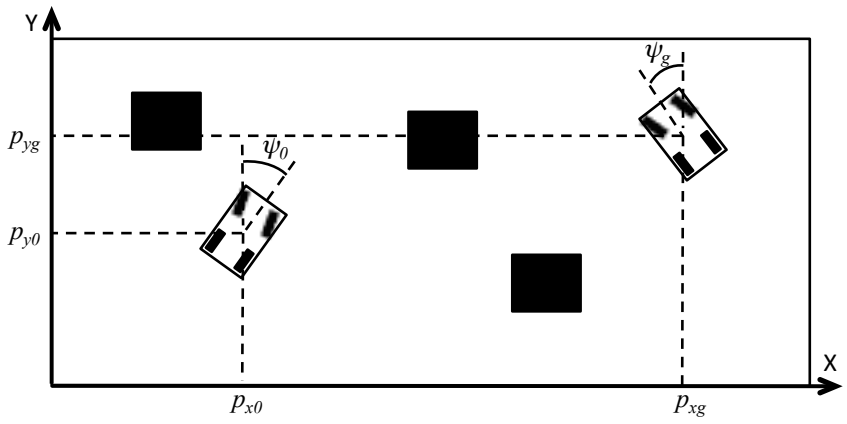


Figure 4.1: An example of a car-like robot path planning problem.

Configuration space representation

A more generic representation commonly used in path planning is a configuration space (C or *C-space*) which is defined as a vector space or manifold of robot configurations q. A set of parameters that uniquely defines the location of all points of the robot in the workspace $\mathcal{W}$ is defined as a robot configuration. In the case of our car-like robot example $q = (p_x, p_y, \psi)$. Not all robot configurations are attainable due to obstacle constraints. The *free space* or C_{free} is a subset of the C-space of a robot that is free from collisions with obstacles.

State space representation

In order to deal with robotic systems in motion, the configuration of the robot is insufficient to describe the problem. Additionally, the dynamic state of a robot (i.e. velocity) has to be accounted for. The state space representation extends the configuration space by adding first order derivatives $\dot{q}$ of the robot configuration q. It means that for a robot configuration $q = (q_0, \ldots, q_n)$, the state, x, is defined as:

$$x = \langle q, \dot{q} \rangle$$
$$\dot{q} = (\dot{q}_0, \ldots, \dot{q}_n)^T$$

As an implication, for an n-dimensional C-space, the state space X is a 2n-dimensional vector space. In the case of our car-like robot example a single configuration in the state-space is defined as $x = (p_x, p_y, \psi, \dot{p}_x, \dot{p}_y, \dot{\psi})$.

Constraints

As previously described, one basic class of constraints used for path planning is a set of collision constraints. However additionally two classes have to be considered in practice, namely the kinematic and dynamic constraints. Both of these types of constraints belong to a class of non-holonomic constraints (also called motion constraints) and are common for many types of robots.

A robot is non-holonomic if the controllable degrees of freedom are less than the total degrees of freedom. Our car-like robot is an example of a non-holonomic robot. It can only drive forwards or backwards and it cannot drive sideways. Further, its turning radius will depend on its velocity.

The kinematic constraints are constraints where only first-order derivatives of the configuration parameters are allowed. Acceleration (i.e. second-order derivatives) are allowed in the dynamic constraints.

A helicopter platform also belongs to a non-holonomic class of robots. Although, it can move freely in any direction its freedom of movement depends on its speed. When a helicopter is hovering or flying with a small speed it could be considered to be holonomic, but in this case its usage would be very limited. The algorithms presented later in this section handle the kinematic and dynamic constraints of the UASTechLab RMAX platform.

Path Planning

The path planning problem is defined as finding a path in C_{free} that connects the start (q_0) and the goal (q_g) configuration. A simple example of a path planning problem for a car-like robot without taking into account motion constraints is presented in Figure 4.1 where $q_0 = (p_{x0}, p_{y0}, \psi_0)$ and $q_g = (p_{xg}, p_{yg}, \psi_g)$.

There are two main classes of motion planning algorithms: combinatorial and sample-based [51]. The problem of finding optimal paths between two robot configurations in a high-dimensional configuration space is intractable in general. Canny and Reif [14] prove that even a simple problem of finding the optimal path for a point-like robot in three-dimensional space with polyhedral obstacles is NP-hard. Another example is presented by Reif and Wang [75] where additionally non-holonomic constraints on the curvature radius are added (e.g. car-like robot). In this case the problem of finding an optimal path is proven to be NP-hard even for two-dimensional problems.

Combinatorial motion planning uses an exact representation of the original problem (often referred as *exact* algorithms). The algorithms in this class are complete and optimal, but most often computationally impractical for solving real-world problems (i.e. more than 2 dimensions). Sample-based methods use an approximation of the C_{free} continuous space (in configuration space or state-space) in order to deal with the complexity of high-dimensional problems. The discrete representation of the original continuous

space (typically represented in the form of a graph) sacrifices strict completeness for a weaker definition such as *resolution completeness* or *probabilistic completeness* [51].

An algorithm that for all problem instances correctly reports whether there is a solution in a finite amount of time is said to be *complete*. In sample-based planning resolution completeness is related to the denseness of the approximation of the C_{free} continuous space. As the number of iterations of the sample-based algorithm goes to infinity the samples come arbitrarily close to any configuration. An algorithm that samples the C_{free} space deterministically is said to be resolution complete. Such an algorithm will find a solution in finite time, if one exists. Otherwise, if there is no solution to the problem it may run forever. Probabilistic completeness is related to random sampling which is used in many sample-based planners. An algorithm is said to be probabilistically complete if the probability that it finds an existing solution converges to one when the number of samples goes to infinity.

4.1.1 Probabilistic Roadmaps

The original probabilistic roadmap (PRM) algorithm [44] works in two phases, one offline and the other online. In the offline phase a discrete roadmap representing a free configuration space is generated using a 3D world model. First, it randomly generates a number of configurations and checks for collisions with the world 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. This results in a roadmap approximating the configuration free space. Figure 4.2 presents an example of the PRM offline phase for a simple 2D environment.

In the online 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. An example of the PRM online phase for a simple 2D environment is presented in Figure 4.3.

Figure 4.4 provides a schema of the PRM path planner used in the UASTechLab RMAX system. The planner uses an OBBTree-algorithm [34] for collision checking and an A* algorithm for graph searching. 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.

- *Multi-level roadmap planning*
  The standard probabilistic roadmap algorithm is formulated for fully controllable systems only (i.e. holonomic). This assumption is true for a helicopter flying at low speed with the capability to stop and

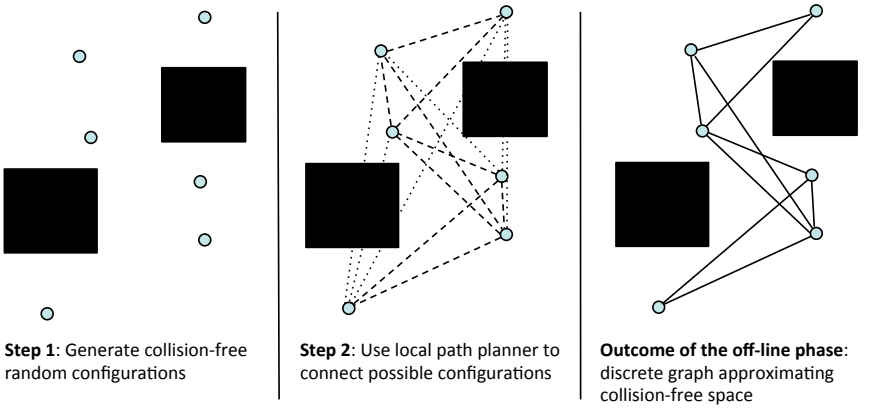


Figure 4.2: Example PRM roadmap generation (offline phase) for a simple 2D environment.

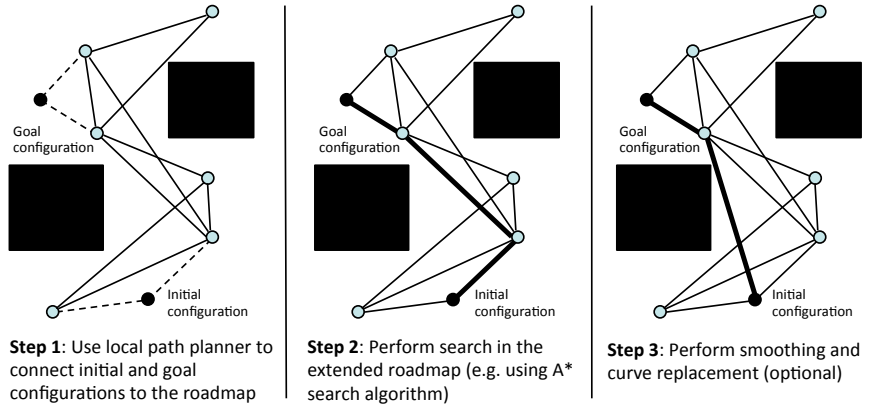


Figure 4.3: An example of the PRM online phase for a simple 2D environment.

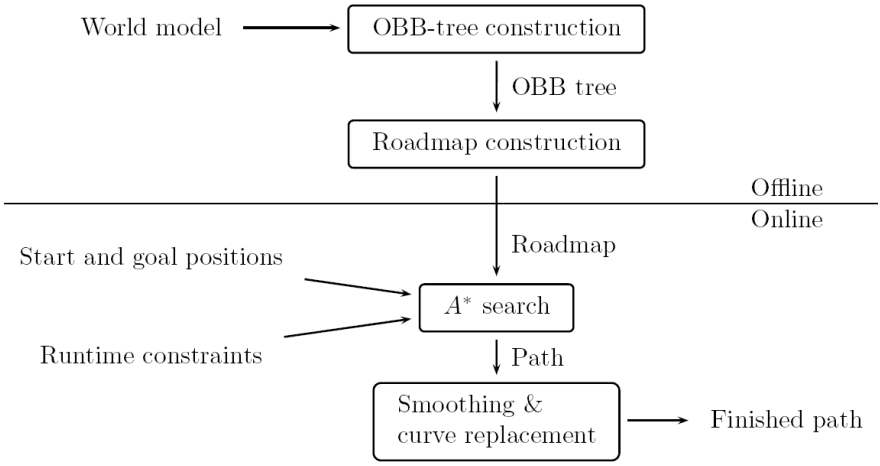


Figure 4.4: PRM path plan generation.

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. In 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 limiting 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 running on the current helicopter hardware and for a typical flight environment is below 1000ms and the use of runtime constraints does not noticeably influence the mean. A more detailed description of the modified PRM planner is provided by Pettersson [74].

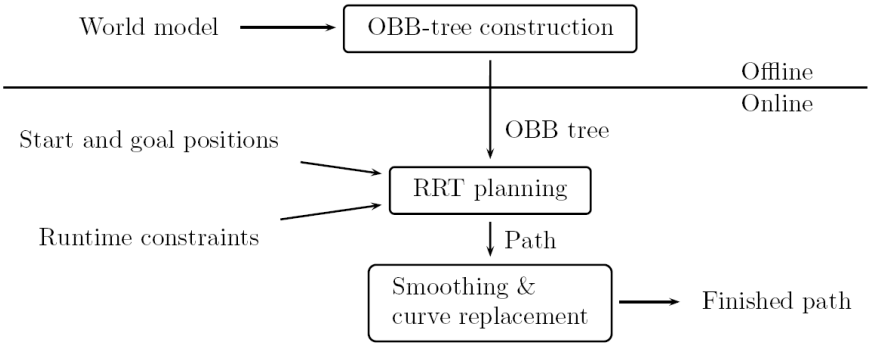


Figure 4.5: RRT path plan generation.

4.1.2 Rapidly Exploring Random Trees

The rapidly exploring random trees (RRT) [49, 50] is a variant of the sample-based algorithm that does not use a precompiled roadmap as opposed to the PRM planner. Instead, it uses a specialized search strategy to construct a roadmap online rather than offline to find solutions quickly during runtime. This is a strong advantage of the RRT algorithm since it does not require knowing a 3D model of the environment before hand. It makes it applicable for dynamic and unknown environments.

The algorithm generates two trees rooted in the start and end configurations respectively, by exploring the configuration space randomly from both directions. While the trees are being generated, attempts are 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 (Figure 4.5). In comparison with the PRM planner, the mean planning time with RRT is also below 1000ms, but in this case, the success rate is much lower and the generated plans are not optimal which may sometimes cause anomalous detours [74].

Figure 4.6 presents an example of RRT execution in a simple 2D environment. Note that the resulting plan is not optimal or as optimal as in the case of the PRM planner. It can happen that even after applying a post processing path optimization step (e.g. removal of redundant configurations) the result can contain detours. In the case of our simple 2D example (Figure 4.6), the tree expansion from the goal node can grow either left or right of the top-right obstacle since the exploration is random. Additionally, each time the planner is executed with the same init and goal configurations, generated plans will most certainly be different, as opposed to the PRM with a fixed precompiled roadmap.

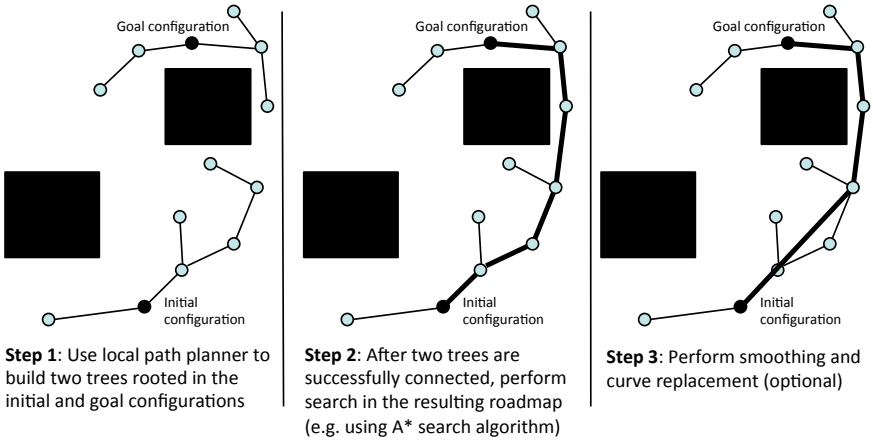


Figure 4.6: Example execution of RRT in a simple 2D environment.

4.2 Dynamic Replanning of the Path

The obstacle avoidance problem in the unmanned aircraft domain is most commonly handled using a reactive control component. Such solutions unfortunately suffer from problems with local minima. For example, model predictive control (MPC, [84]) solves the control problem for a certain time horizon, but it does not preserve global plan optimality.

Motion planners, on the other hand, have a global view of the problem and can generate plans that take all known (old and new) obstacles into account. Our objective is therefore to use motion planners to the maximum extent possible. Each time a new obstacle or no-fly zone obstructing the current flight path is detected, a *Strategy Selector* determines which replanning strategy can be expected to yield the best plan within the available time.

The amount of time available depends on several factors. The most obvious ones are the range at which the new obstacle is detected and the UAV's current velocity. Given these factors, the time remaining before the UAV reaches the obstacle can be calculated. However, we cannot spend all of this time calculating a new path, or we will finish just in time for a collision. We must reserve enough time to change to a new trajectory. This is subsumed by the time required to perform an emergency brake in case replanning takes longer than estimated. As soon as we detect a target at a given distance, we therefore subtract the required braking distance for our current velocity as well as a safety margin of 6 meters, the minimum safe distance between the helicopter and an obstacle. Dividing this with the UAV's current velocity gives us the time window in which replanning can be performed.

As described in section 3.3.2, paths generated by the path planners are executed using a reactive Task Procedure (TP). The TP sends sequentially a set of path segments to the Path Following Control Mode (PFCM) for

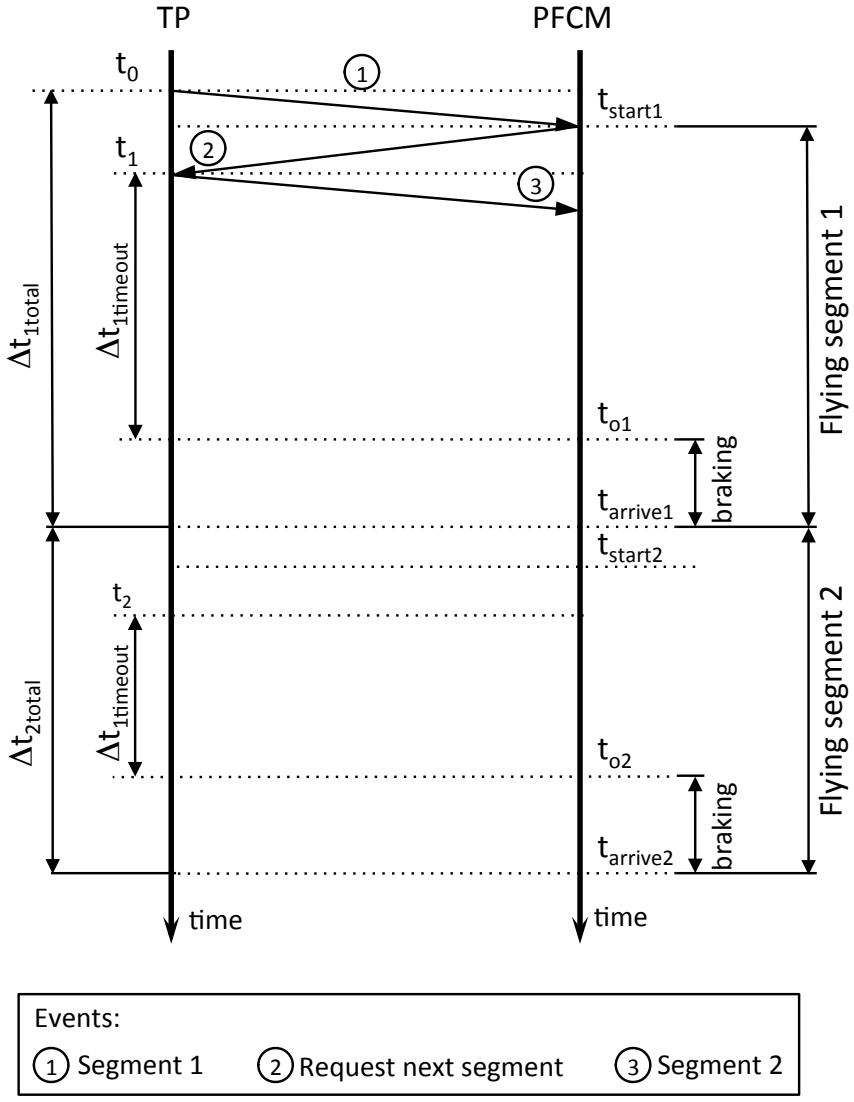


Figure 4.7: Execution time-line for a path consisting of 2 segments.

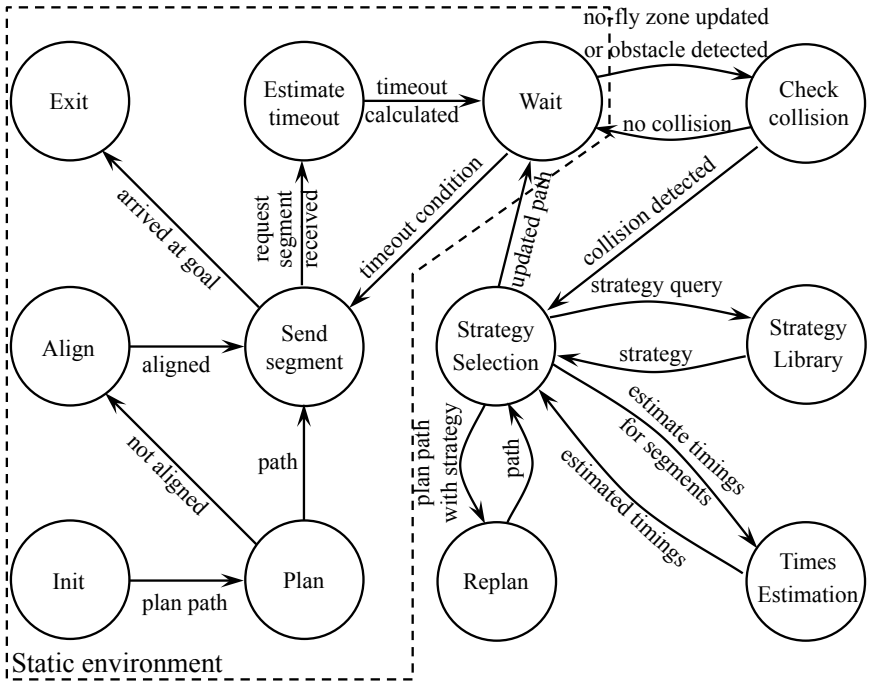


Figure 4.8: The dynamic path replanning automaton.

execution. Figure 4.7 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 PFCM controller and waits for a *Request segment* event which arrives immediately (t_1) after the helicopter starts to fly (t_{start1}). Typical delays 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 $\Delta_{t_1 timeout}$ ($t_o 1 - t_1$) time window to send the next segment to the PFCM before it initiates the safety braking procedure. If the segment is sent after t_{o1}, the helicopter will start braking. In the current implementation, segments that are sent after the timeout are ignored. This will be changed in a future implementation. In practice the $\Delta_{t_1 timeout}$ time window is almost always large enough to use one of the standard path planners for the path repair. The updated segments are then sent to the PFCM controller transparently.

Note that a new obstacle may be detected at any time point, thus not always the full $\Delta_{t_1 timeout}$ time window can be used for the path repair. Choosing a particular scheme for the path repair is discussed later in this chapter and in chapter 5.

There are several services that are used during 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 the path is depicted in Figure 4.8. The TP takes the start and the end points and the 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 PFCM 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 the timeout condition is true and then sends the next segment to the PFCM 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 the available replanning strategies that may be used to replan when calling the path planner (*Replan* state).

The TP terminates when the UAV arrives at the goal position. More details on the Strategy Selector service, the Strategy library and the Prediction service will follow in the next subsections.

4.2.1 Prediction Service

All time estimations that have to do with paths or part of the paths are handled by the Prediction service. It derives the velocity profile of the vehicle along the path using the path following control mode and the helicopter model (section 2.2.3). The profile is based on the path parameters, the cruise and final velocity and takes into account the control mode and platform dynamic parameters. By applying an integration of the velocity profile over the time, specific timings can be derived, i.e. timeouts, total times, and combinations of those. For instance, in the case of flying a two-segment trajectory (see the execution timeline in Figure 4.7) 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).

4.2.2 Strategy Library

When part of the path is no longer valid, 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 for this step which can give different results depending on the situation.

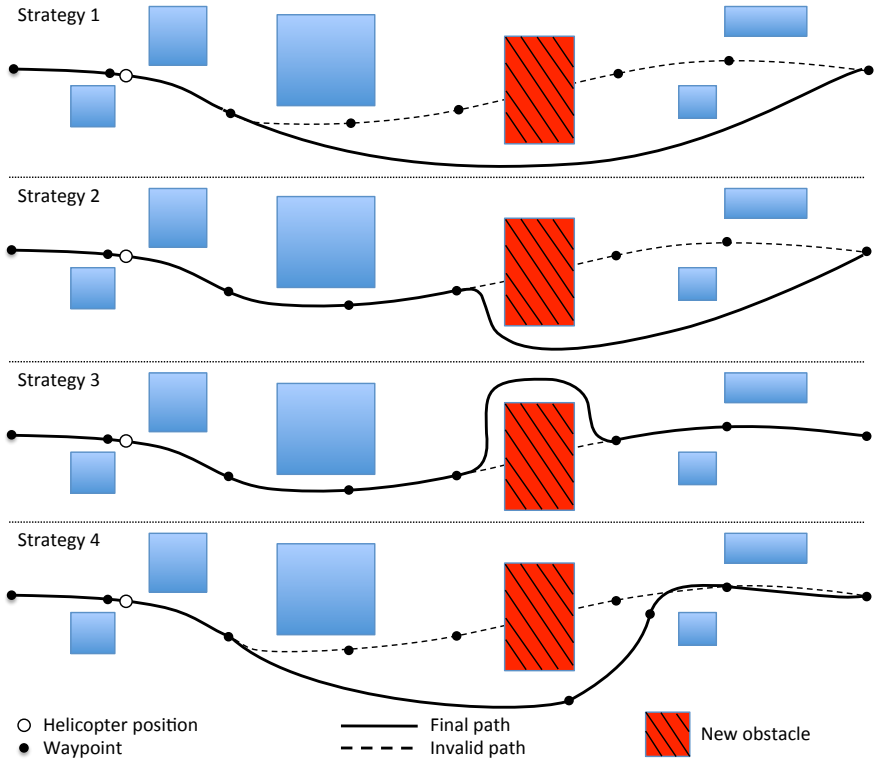


Figure 4.9: Examples of replanning strategies.

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 Figure 4.9.

- *Strategy 1*
  Replanning 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 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 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 can be 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 described in the next subsection.

4.2.3 Strategy Selector Service

The Strategy selector service is responsible for choosing the strategy or strategies to execute in the event of path occlusion. It keeps track of the time that it uses, so that a valid path is always available when the timeout condition becomes true. The Strategy Selector holds information as to which segments of the path were invalidated and it can use the Prediction service to get estimated timings for the path or parts of the path. Based on that and the available strategies (from the Strategy Library) it can make a decision which strategy or strategies to use for replanning at the current time. If many strategies are applied and more new plans are generated, it also evaluates them according to a given optimization criterion that is declared by the user or another service. For instance, if the time window for making a decision about the next segment is short then the fastest strategy is used in order to produce a valid plan on time.

The Strategy Selector is also responsible for updating information about strategies in the Strategy Library, in particular estimated execution times. The same strategies in different environments might require less or more time for execution. This information is fed back to the library, so the next time the Strategy Selector has more accurate information about the execution time and can make a better decision which strategy to apply.

4.3 Time Analysis of Replanning Strategies

In order to check the feasibility of using the proposed replanning technique a set of necessary experiments were conducted. The main objective was to

check if any of the proposed strategies can be executed in time during typical mission execution.

We have used both the path planners available in the UASTechLab RMAX system (i.e. PRM and RRT, section 4.1). We included the first three strategies from the Figure 4.9 in the Strategy Library. During the flight forbidden regions were randomly added by the ground operator. In order to compare the performance of different strategies only one strategy was used per experiment.

Typical values of parameters related to the execution and the planning phases are presented in Tables 4.1 and 4.2, for *Strategy 1* and *Strategy 3*, respectively. *Strategy 2* is omitted from the comparison because typical results of applying this strategy fall in between *Strategy 1* and *Strategy 3* (plan repair vs full replan).

Table 4.1: Results of the experiments using Strategy 1

Planner	path length (m)	number of segments	added forbidden regions	min. segment length(m)	max. replanning time (ms)	minimum Δ_t (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

Observe that in the case of *Strategy 1* (Table 4.1), Δ_t (time window for replanning, last column) is generally greater than four times the amount of time required to generate full plans using either the PRM or RRT planners.

The difference is even greater (up to 20 times) in the case of *Strategy 3* (Table 4.2). This is as expected, the more the existing plan is reused the less time is needed to repair it. Although, replanning times for applying *Strategy 3* are much smaller, the paths have much more segments (up to 15). Such paths usually imply a smaller average velocity which typically results in a longer flight time.

Values of the Δ_t presented in the results are calculated under the assumption that the obstacle is detected as early on as possible. In practice,

Table 4.2: Results of the experiments using Strategy 3

Planner	path length (m)	number of segments	added forbidden regions	min. segment length (m)	max. replannig time (ms)	minimum Δ_t (ms)
PRM	524.47	12	4	28.23	196	2938
	514.51	10	4	41.42	185	2892
	607.72	11	4	33.98	163	2928
	594.59	14	5	13.02	160	1080
	586.74	12	5	20.53	163	1005
	546.15	12	5	16.60	153	2607
	575.15	13	6	29.38	202	2907
RRT	495.07	10	4	24.06	104	2088
	527.95	11	4	12.24	240	1249
	558.45	10	4	23.79	160	2096
	562.07	12	5	22.14	132	1529
	586.70	15	5	15.83	156	2686
	604.90	13	5	21.97	251	2556
	576.27	15	6	16.12	206	2696

however, the detection can occur during any moment in the time window, thus the choice of which strategy to apply becomes not as trivial as choosing, for example *Strategy 1* (because it most often yields best results and from the presented experiments it would be applicable in all of the cases). The problem of choosing a specific strategy is discussed later in chapter 5.

4.4 Experimentation

This section presents an example of a short flight where the dynamic replanning mechanism was tested. Figure 4.10 shows the logged flight-test data superimposed on the map of the area. Gray polygonal area marks the no-fly zone added during the flight by the ground operator. Black dotted line shows the invalidated part of the path (between the WP2 and WP3 waypoints).

The flight started with an autonomous take off, and the helicopter began executing the planned path towards the designated waypoints. After arriving at the first one (WP1), the direction of flight changed to south and the execution of the planned path (from WP1 through WP2 to WP3) began. A ground operator added a no-fly zone intersecting the flight path. An approximate position of the helicopter at the time of adding the no-fly zone is marked by the white arrow. 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 (WP3), it was commanded to return

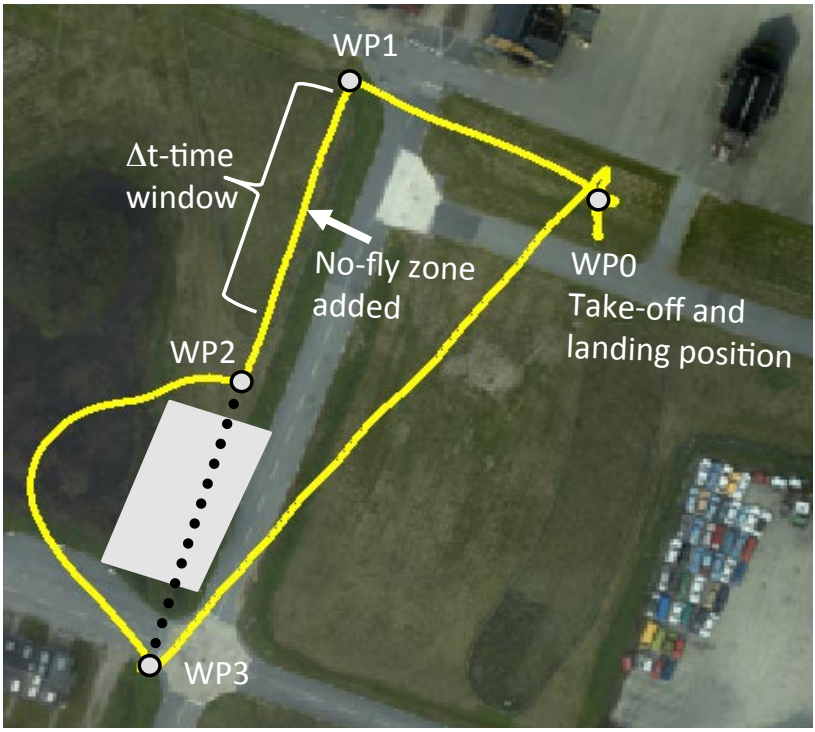


Figure 4.10: Use of the dynamic replanning in a real mission.

to home base and land (WP0). The replanning of the path was performed using the PRM planner and the *Strategy 3*. Its execution took 187 ms.

4.5 Summary

This chapter discussed the problem of the dynamic path replanning. Two sample-based path planning methods available on the UASTechLab RMAX helicopter have been described. As shown in the analysis of the path execution, when a new obstacle is detected there can be up to few seconds available for the system to repair the current flight path.

The proposed dynamic replanning framework uses a combination of a path planning algorithm together with a set of replanning strategies to ensure the executed path is always collision-free. The timing analysis of the proposed strategies proves the feasibility of using such a solution. An important component of the framework is the *Strategy Selector* service responsible for choosing the planner/strategy tuple that will yield the best possible solution in the available time before the potential collision. The solution to this selection problem is discussed in the next chapter.

Chapter 5

Choosing Replanning Strategies

In the previous chapter a mechanism for dynamic replanning was introduced. As described, the basic idea is to use the path planning algorithms to the maximum possible extent when new obstacles or no-fly zones obstructing the current flight path are detected.

We proposed a set of example strategies that can be used during the replanning phase (section 4.2.2, page 61). For most strategies, calculating the expected time requirements for replanning is considerably less straightforward than calculating the available time to collision. The timing depends on numerous features of the current plan, the obstructed segment and the relevant areas of the map. Without taking such information into account, we would have to fall back on always using a simple approach such as *Strategy 3*. As this strategy only makes local repairs, it is generally faster and its time requirements vary less. But in many cases we do have enough time to use better strategies – if we have the ability to *predict* that this is the case for the current environment and path.

Many path planning methods are also parameterized in various ways. For example, PRM planners (section 4.1) generate a roadmap graph in a pre-processing phase and search this graph whenever a plan is required. Increasing the number of nodes in the graph will increase the plan quality, but again, this will also affect the time required for plan generation.

The selection of a particular replanning strategy and a parameterized path planning algorithm is performed by a *Strategy Selector* (section 4.2.3). This service should determine which choice can be expected to yield the best plan within the available time. The problem is presented in Figure 5.1. Given the domain information (e.g. map), current context (e.g. initial path) and available time to collision, what is the expected replanning time and flight time for each of the possible strategy/planner choices? Although the flight time was used as the plan quality metric other criteria can be used,

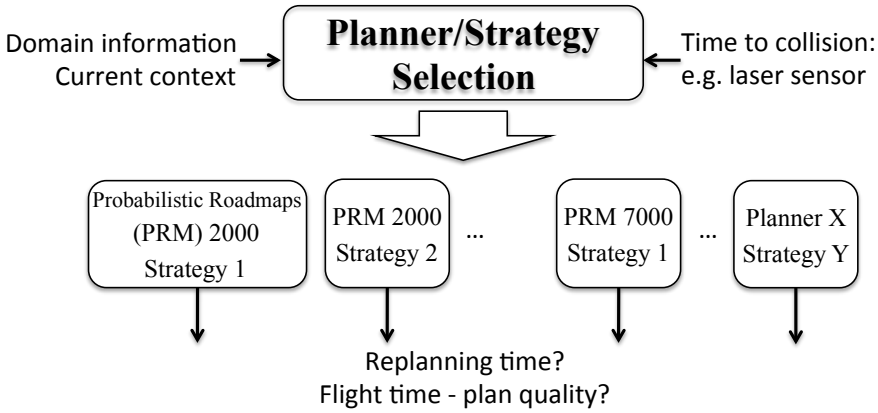


Figure 5.1: The basic idea of choosing replanning strategies.

for example fuel consumption.

The problem presented is a classical prediction problem. Many applications of prediction can be found in different fields, for example in signal processing, control theory, robotics, and artificial intelligence. The foundation of any prediction is a model which encapsulates the knowledge, or assumptions, concerning the predicted system. Without the model the prediction mechanism would not be able to conclude anything about the future. Thus any prediction would be a guess.

The model building/acquisition techniques can be divided into three categories: modelling from first (or physical) principles, estimation (or learning) from data, and encoding of "expert knowledge". Those techniques are often combined, for example in system identification applications [55]:

- White-box identification: a dynamical system is described by differential equations and the estimation of their parameters is based on experimental data. For example, an aircraft flight model.
- Gray-box or semi-physical identification: a generic model structure is known and the estimation of parameters is derived from data. For example, a water tank system [31].
- Black-box identification: no knowledge of either the model structure or physical principles is known. The model is solely derived from data. For example, stock market predictions.

In the case of our problem we do not have any knowledge of the structure of the model that could be applicable. Neither do we know how the context and domain information influences planning time and planning quality. We do have a general feeling that when the domain becomes more complex the required time will increase, but what is the physical relation which could

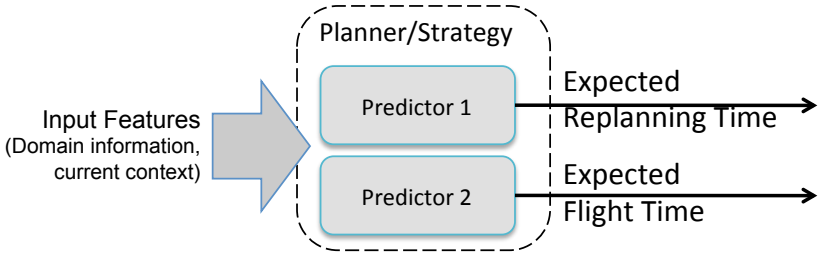


Figure 5.2: The concept of building predictors using machine learning.

describe that dependency? Therefore, our prediction model has to be derived using the black-box approach.

Another important issue is the computational complexity of building and using the model. In our case the predictive model should be as fast as possible in order to leave as much of the available time to the selected planner/strategy repair process. It should be at least an order of magnitude faster than running an actual strategy/planner tuple. Otherwise, it may become impossible to execute any of the available repair strategies since the time spent on making the prediction was too long.

We therefore use machine learning techniques to generate a suitable set of predictors for each particular flight environment and aircraft type, where each motion planner is viewed as a black-box function 5.2. We assume a stationary distribution, where the relevant properties of the map do not change over time. If significant changes are detected, for example because large numbers of new obstacles have been detected, the prediction model can be recomputed for use in future missions.

In the following subsections, we will describe the algorithms we have used, the features that were selected and the results of empirical experimentation.

5.1 Support Vector Machines

Several machine learning techniques were tested and compared, including support vector machines (SVM) [93, 94], least median squared linear regression, gaussian processes for regression, isotonic regression, and normalized Gaussian radial basis function networks.

With their high generalization performance and an ability to model non-linear relationships, support vector machines have been shown to outperform other alternatives in many applications. They are applicable to many real-world problems such as document classification and pattern recognition [94], face detection and recognition [53] and vehicle detection [87]. As it turns out, SVMs also yield the smallest prediction errors for our domain.

The idea underlying (non-linear) support vector machines is that n-

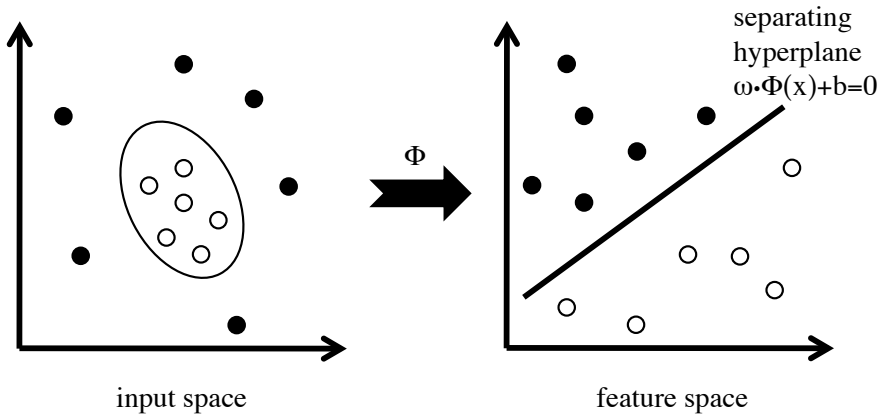


Figure 5.3: Mapping and separating hyperplane.

dimensional input training data can be mapped by a non-linear function Φ into a high-dimensional feature space, where the resulting vectors are linearly separable (Figure 5.3). One then constructs a separating hyperplane with maximum margin in the feature space.

Consider a classification problem where $x_i \in \mathbb{R}^n$ for $i = 1, ..., l$ is a training set of size l and $y_i = \pm 1$ are class labels. Given a suitable Φ, the SVM method finds an optimal approximation $f(x) = \omega \cdot \Phi(x) + b$ such that $f(x) > 0$ for positive examples and $f(x) < 0$ for negative examples, where $\omega \in \mathbb{R}^n$ is a vector perpendicular to the separating hyperplane and $b \in \mathbb{R}$ is an offset scalar. This is referred to as Support Vector Classification (SVC).

The SVM approach can also be used for solving regression problems (Support Vector Regression, SVR), where each x_i in the training set is associated with a target value $y_i \in \mathbb{R}$. The SVR tries to find a function $f(x)$ that can be used for accurate approximation of future values. The generic SVR function can be written as

$$f(x) = \omega \cdot \Phi(x) + b$$

and can be solved by maximizing $W(\alpha^*, \alpha) =$

$$-\frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\Phi(x_i) \cdot \Phi(x_j)$$

$$-\varepsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*)$$

where α_i and α_i^* are Lagrange multipliers, subject to

$$\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0 \text{ and } \alpha_i, \alpha_i^* \in [0, C]$$

which provides the solution

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)(\Phi(x) \cdot \Phi(x_i)) + b$$

As expressed above, dot products are calculated in a high-dimensional or possibly infinite-dimensional space. This can often be avoided by replacing $\Phi(x_i) \cdot \Phi(x_j)$ with a suitable kernel function $K(x_i, x_j)$ satisfying the conditions of Mercer's theorem. Examples of commonly used kernel functions include:

- linear: $K(x_i, x_j) = x_i \cdot x_j$
- polynomial: $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$
- RBF: $K(x_i, x_j) = exp\left(-\gamma \parallel x_i - x_j \parallel^2\right), \gamma > 0$
- PUK: $K(x_i, x_j) = \frac{1}{\left[1+\left(\frac{2\sqrt{\|x_i - x_j\|^2}\sqrt{2^{1/\omega}-1}}{\sigma}\right)^2\right]^{\omega}}$

We have used the Pearson VII Universal Kernel (PUK) [92]. The PUK provides equal or stronger mapping power compared to several standard kernels, and can be used as a generic alternative to the common linear, polynomial and Radial Basis Function (RBF) kernels. We use iterative Sequential Minimal Optimization (SMO) [83, 85] to solve the regression problem, which has minimal computational requirements [98].

5.2 Prediction Features

Many parameters influencing planning time and plan quality were considered as potential inputs to the machine learning algorithm. After empirical testing, a set of features were selected which produced good results. The following input features were used for building the prediction models (all normalized to the range of [-1,1]):

- Information about the initial plan: number of segments, path length, estimated flight time, time required for initial plan generation, and the target velocity. Information about static obstacles, such as buildings, trees and static no-fly zones, is implicit in these measures.
- Information about dynamically added obstacles including no-fly zones: total area and number of all new obstacles in region 0 , region 1, region 2, region 3, and the entire map (see Figure 5.4).
- Information about the obstructed segment: number of segments from the current point to the obstructed segment, and Euclidean distance between the start and the end of the obstructed segment.

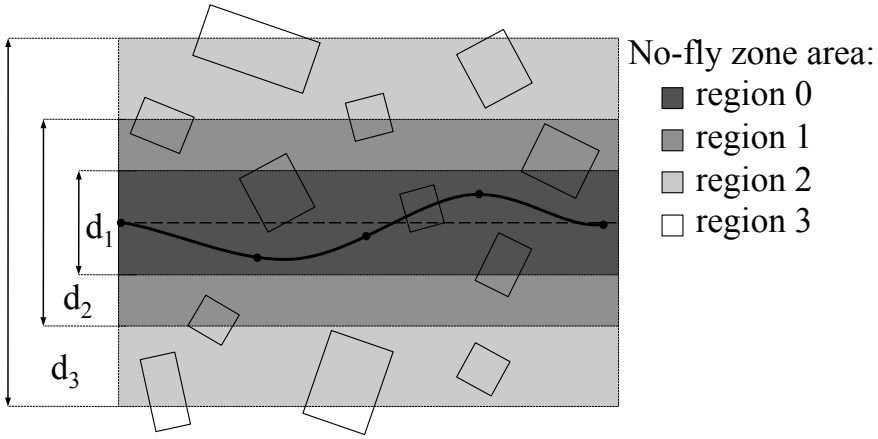


Figure 5.4: No-fly zone area calculation.

Correlation-based feature selection [37] was used to assess the relevance of these features relative to the six replanning strategies (defined in the next subsection) and the two quantities to be predicted (time requirements and plan quality). The results differed considerably across the twelve cases and may also be dependent on the map being used. As support vector machines are quite robust against the inclusion of irrelevant features, we decided to use the full set of features for all prediction models.

We currently use 2D area information for dynamically added obstacles, but may augment this with 3D obstacle volumes in the future. The parameters d_1, d_2 and d_3 in Figure 5.4 were chosen empirically and for our test models were equal to 20, 40 and 60 meters, respectively. The choice of parameters can be automated by building a set of models using the training set for different values of d_i and comparing the resulting prediction accuracy on the test set.

5.3 Experimental Results

The method has been evaluated on two environments of different complexity. The first environment is a 3D model of a real flight test venue, an urban area of approximately 1 km^2. It consists of 205 buildings and other structures (e.g. trees) constructed by around 20000 polygons. The model has a simplified flat ground elevation representation. The second environment extends the first by adding ground elevation data, increasing the number of polygons in the 3D model to 120000. Maps were generated using manned aircraft with a laser sensor, with an accuracy of 10 cm. Experiments take place in hardware-in-the-loop simulation with all the necessary services running as in a real flight.

Environment	Strategy	Nodes	Model Evaluation Results: Replanning time prediction [%]	Model Evaluation Results: Flight time prediction [%]
1	1a	2000	-0.06 ± 4.17	-0.85 ± 4.35
	2a	2000	-1.49 ± 7.85	0.06 ± 2.82
	3a	2000	-1.25 ± 21.38	0.38 ± 2.17
	1b	7000	-0.23 ± 1.72	0.49 ± 4.46
	2b	7000	-0.86 ± 7.51	-0.17 ± 2.70
	3b	7000	-0.14 ± 8.69	0.35 ± 1.80
2	1a	2000	-0.38 ± 6.13	0.84 ± 4.40
	2a	2000	-1.05 ± 13.60	0.35 ± 3.00
	3a	2000	-0.73 ± 23.94	0.31 ± 3.31
	1b	7000	-0.20 ± 4.48	0.56 ± 3.95
	2b	7000	-2.55 ± 11.23	0.09 ± 2.42
	3b	7000	0.30 ± 10.08	0.23 ± 2.16

Table 5.1: Relative mean error of prediction and standard deviation for the PRM planner.

We began the learning process using the PRM path planner with a 5-dimensional configuration space (3-dimensional position plus 2-dimensional direction of flight). A set of 1500 training samples was used for each of the two environments.

Each sample was generated in the simulation environment as follows. First, a random number of no-fly zones in the range from 1 to 15 were added to the environment. Then, an initial path was generated by the planner, with start and goal positions chosen randomly within the environment. A single no-fly zone was used to randomly obstruct one of the initial path segments, corresponding to an obstacle newly detected by, for example, a laser range finder sensor. Finally, six plan repair strategies were applied and the resulting timing and path quality values were logged. Strategies 1a, 2a and 3a were configured as shown in Figure 4.9 using a sparse 2000-node roadmap. Strategies 1b, 2b and 3b are similar, but use a denser 7000-node roadmap.

All the experiments presented in this section were performed with a fixed target velocity of 10 m/s and distances between start and goal configurations greater than 700 m. This setup allows us to present the results in a clear way and compare optimal and worst case scenarios over all 500 test cases that were generated for the evaluation. The performance of the machine-learned models is similar in the cases where these assumptions do not hold.

SVR parameter tuning was performed using exhaustive grid search over the kernel parameters σ and ω. Other parameters (i.e. the C and ϵ constants for support vector regression) were chosen manually.

Prediction quality. 500 samples were used for the evaluation. For each sample, we calculated the relative error of prediction for both plan quality and replanning time, defined as $(y_i - \hat{y}_i)/y_i$, where $\hat{y}_i$ is the prediction and y_i is the measured value. Table 5.1 presents the mean of the relative errors (the Relative Mean Error of Prediction, RMEP) and their standard deviation for each environment and strategy.

As seen in the table, the quality of a repaired path (expressed as the required flight time for the path) can be predicted with high accuracy for each of the six strategies and in each of the environments. More importantly the standard deviation of the error is also quite small, ranging from 1.80% to 4.46%, demonstrating that the prediction rarely deviates greatly from the true value. The deviation is greater for strategies 1a/b, where a larger part of the path is replanned.

We can also see that it is somewhat more difficult to predict the time required for replanning. However, using a greater number of nodes decreases variability considerably. Part of the remaining variation is due to unavoidable factors such as processor load and Java garbage collection.

For each environment, we have analyzed what these prediction properties mean in terms of enabling us to satisfy our main objective: choosing the highest quality replanning strategy that is possible given the available time.

To make this choice for a particular path and environment, we first predict the expected replanning time and the expected resulting plan quality for each of the six strategies. We then use the highest-quality strategy among those whose predicted replanning time is sufficiently low. This leads to the question of what "sufficiently low" means. The simplest criterion would be that the predicted replanning time does not exceed our current time window. However, the predicted time is an expected value, not an upper bound. Using this criterion, there may be a significant risk that the time window is exceeded. An alternative would be to add one or two standard deviations to the predicted time, and choose among those strategies where this estimate does not exceed the time window. The results of using these three decision criteria are presented in several graphs.

Plan quality. Figure 5.5 is generated from testing in environment 1, and presents the mean flight time (over 500 samples) as a function of the available decision time window.

With a time window of only 50 ms, strategy 3a is chosen for all samples and all three time-dependent decision criteria (*direct*, *1-sigma* and *2-sigma*). This leads to a mean flight time of around 118 seconds. When the time window increases, higher quality strategies are predicted to succeed for some of the samples, and the mean flight time steadily decreases. As expected, flight times decrease more quickly for the *direct* criterion, where no safety margins in terms of plan generation time are added.

As stated before, the best option available *without* predictive abilities is to use a strategy that is very fast regardless of the environment or the properties of the original path. Strategies 1a/b and 2a/b regenerate a large

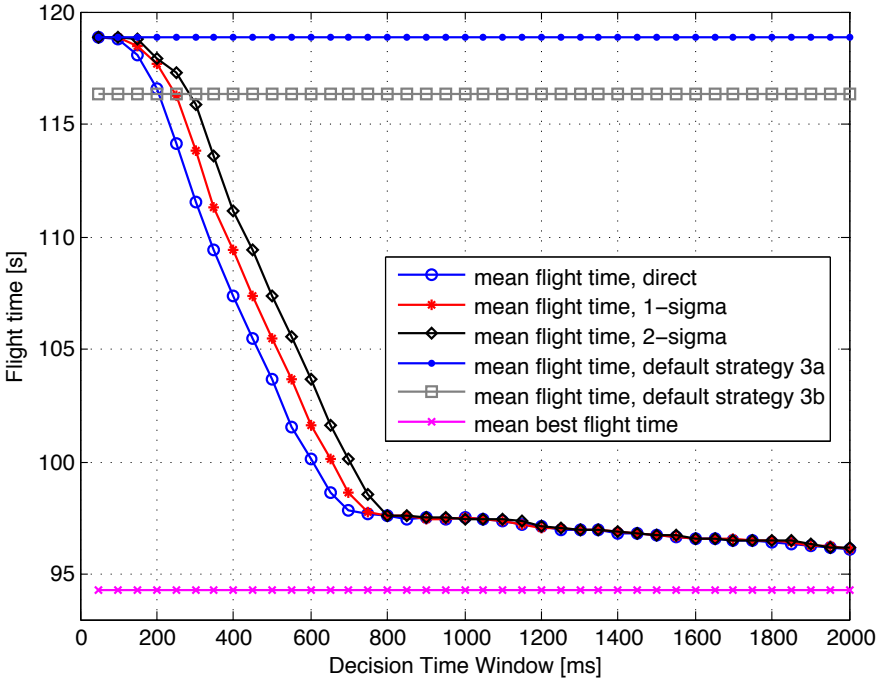


Figure 5.5: Plan quality (flight time) as a function of the available decision time window for environment 1.

part of the path, and therefore require more time than strategies 3a/b. They also vary more depending on the environment. Thus, strategies 3a/b are more suitable as baselines with which our results are compared. As these baselines do not take time windows into account, we see them as straight lines at approximately 118 and 116 seconds of flight time, respectively.

For comparison, we also show the mean flight time that would result from always using the *best* possible strategy: Slightly less than 95 seconds for the average sample. Realizing this in practice would require a very long time window, with sufficient time to run all strategies and choose the best result. As seen in the figure, the three decision criteria based on machine learning tend to reach a quality level quite close to this optimum given a sufficiently large time window.

Success rates. Figure 5.6 shows the success rates for each decision criterion. Here we see the flip side of the improved flight times for the *direct* criterion: With time windows up to around 700 ms, this criterion fails to deliver plans on time in up to 5% of the cases. Whether this is acceptable depends on the application at hand and the penalty associated with having to slow down or stop. The *1-* and *2-sigma* criteria are considerably better in this respect, and can hardly be discerned from each other in the graph. Always using

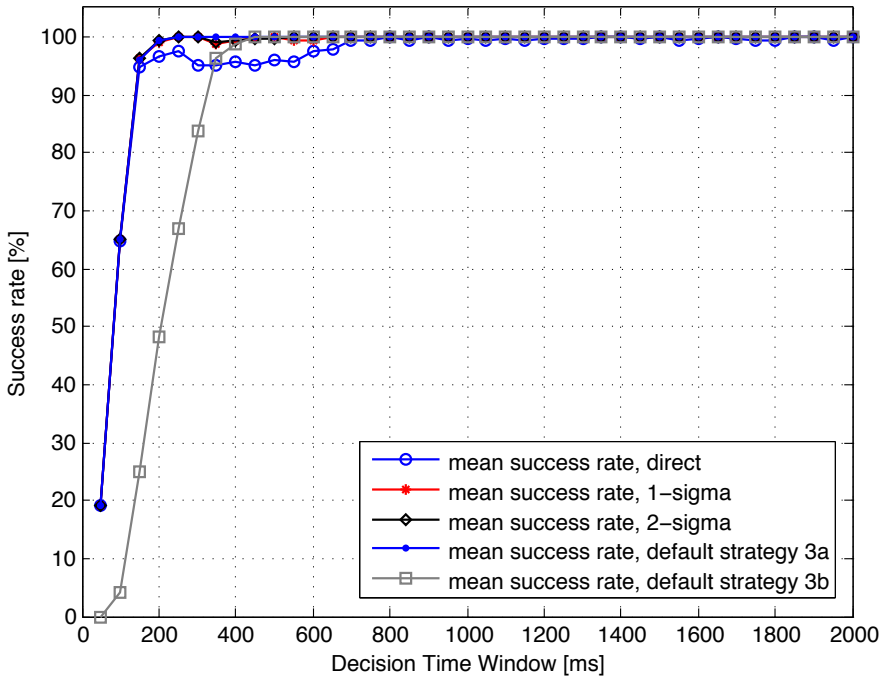


Figure 5.6: Success rate of execution of the chosen strategy in the available decision time window for environment 1.

strategy 3a yields the highest success rate (but the lowest quality). Strategy 3b often requires several hundred ms and thus yields a very low success rate for shorter time windows.

Second environment. Figures 5.7 and 5.8 show the corresponding results for environment 2. Due to the slightly larger prediction errors in this environment, the mean success rate for the *direct* criterion is somewhat worse. However, the *1-sigma* and *2-sigma* criteria still yield considerably better plans than either of the fixed strategies (3a/b) for time windows of around 400 ms and up.

Predictions for RRT. A similar predictive model has been built for the RRT planner. As could be expected, prediction is generally not as accurate for this planner, with a higher relative mean error of prediction. This is mostly due to the more random nature of the RRT algorithm: Instead of using a single sampled roadmap for all queries, the RRT randomly explores the environment from the start and goal position for each single planning query. Time requirements and plan quality still depend on the selected features, such as the area of the newly detected obstacle or obstacles, but also have a significant random component, making the construction of a prediction model with high accuracy considerably harder.

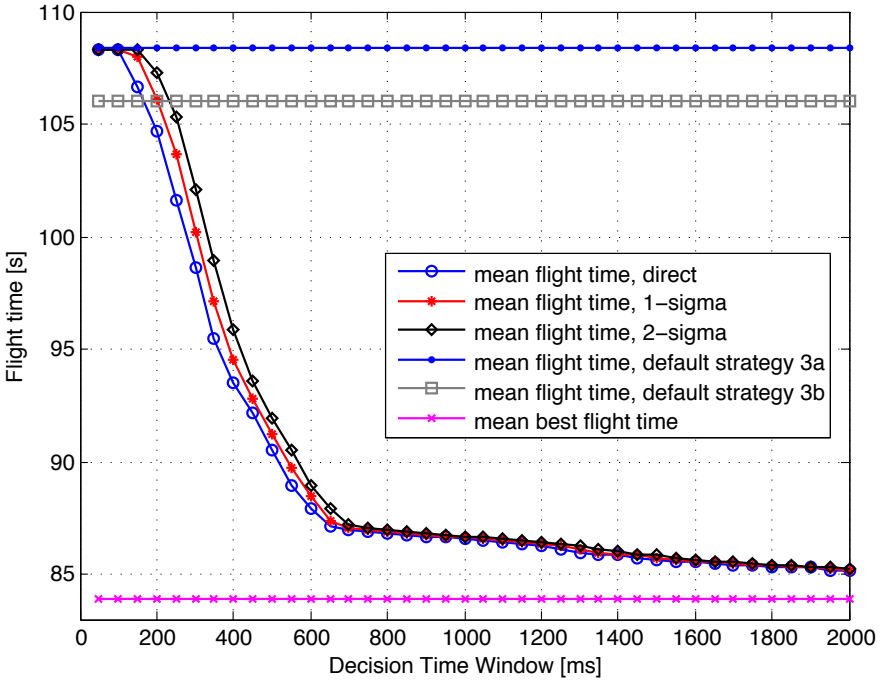


Figure 5.7: Plan quality (flight time) as a function of the available decision time window for environment 2.

5.4 Related Work

In the framework proposed by Morales et al. [71], a set of planners can cooperate to generate a roadmap covering a given environment. Machine learning is used to divide the environment into regions that are homogeneous with respect to certain features, such as whether obstacles are dense or sparse, after which a suitable planner is chosen for each region. Region-specific roadmaps are then created and eventually merged. This approach shows promising results, but is explicitly limited to roadmap-based planning and does not handle the choice of replanning strategy.

A similar approach is presented by Rodriguez et al. [77], where the strategies used by the RESAMPL motion planner are guided by the entropy of each region.

Burns and Brock [13] propose a model-based motion planning technique, where an approximate model of the configuration space is built using locally weighted regression in order to increase planner performance and make predictions about unexplored regions. Although the technique can be used for problems involving motion planning in dynamic environments it does not explicitly consider time constraints. Machine learning is used to provide faster solutions, but there is no attempt at providing the best possible solution for

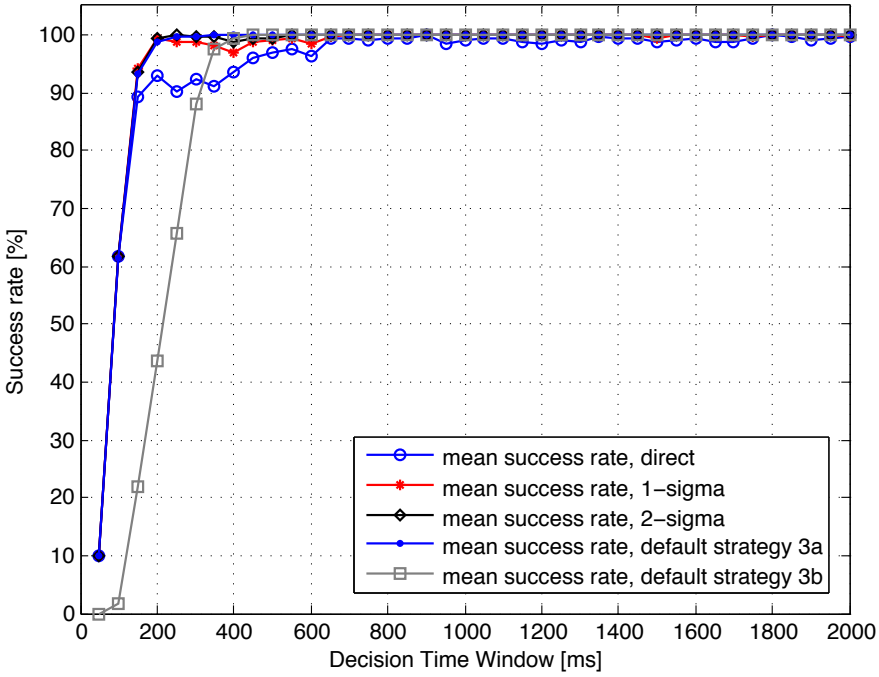


Figure 5.8: Success rate of execution of the chosen strategy in the available decision time window for environment 2.

a given time window, which is required for our problem.

Hrabar [41] presents a UAV system using a stereo-vision sensor for obstacle avoidance. A* search is used within the PRM planner to calculate the initial path. When an obstacle is detected, D* search is used [86]. In this approach there is no consideration for how much time replanning may require. It is assumed that the aircraft can stop and hover, potentially excluding the use of this technique for platforms such as fixed-wing aircraft. The experiments presented use a flight velocity of 0.5 m/s.

A number of motion replanning algorithms have been proposed in the literature, including DRRT [30] and ADRRT [29]. These algorithms incrementally generate improved solutions, thereby spending part of their time on generating solutions that will not be used. In contrast, our framework uses machine learning to determine suitable bounds for replanning in advance, spending almost all available time generating the final solution.

Chapter 6

Map Building Using A Laser Range Finder

An accurate model of the environment is a prerequisite for sucessfull navigation. As described in chapter 4, the path planning algorithms use a geometrical description of the environment to generate collision-free paths. The safety of a UAS operation therefor depends on having an accurate 3D model. Maps may become inaccurate or outdated over time because of the environment changes (e.g. new building structures, vegetation growth etc.). Thus, adequate sensors and techniques for updating or acquiring new 3D models of the environment are necessary.

Among the many sensors available for providing 3D information about an operational environment, laser range finders provide high accuracy data at a reasonable weight and power consumption. One of the reasons for the innovation in this particular sensor technology is its wide use in many industries. Example applications include automation tasks [52], volume or stockpile measuring [15], forest inventory [3], and surveillance [95]. Laser range finders have also received a great deal of interest from the robotics community. Their main usage is in navigation and mapping tasks for ground robotic systems. Examples include: localization [12], 2D Simultaneous Localisation and Mapping (SLAM [68]), 3D SLAM (includes 3D position [16]), and 6D SLAM (includes 3D position and attitude [72]).

In this chapter we present techniques for acquiring 3D models of the environment using a UAS platform equipped with a laser range finder sensor. First, a description of the sensor integrated with the UASTechLab RMAX platform is presented. Second, a method consisting of several steps required for obtaining a world model is described. The steps include the necessary transformations of acquired sensor data and methods for improving a map quality when several measurements are merged together. Next, the initial results of some acquired 3D models during a flight test session are presented. Finally, a short feasibility analysis of using such a sensor in the context of the

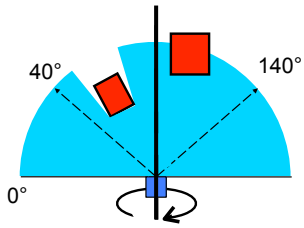


Figure 6.1: Top view of the SICK LMS-291 scanning field and the axis of rotation when using the rotation mechanism.

proposed dynamic replanning framework (chapter 4) concludes the chapter.

This chapter is work in progress and should be considered tentative. But it is useful as a framework for future research.

6.1 Integration of the Laser Range Finder

The device integrated with the UASTechLab RMAX system is the popular LMS-291 from SICK[1]. The laser has been mounted on an in-house developed rotation mechanism which allows for obtaining half-sphere 3D point clouds even when the vehicle is stationary. Similar approach to the integration of the laser range finder with a RMAX UAV platform is used by Whalley et al. [97].

Technical specification of the laser range finder system

LMS-291 is a non-contact optical distance measurement system and it does not require any reflectors or markers on the targets nor scene illumination to provide real-time measurements. It performs very good both in indoor and outdoor environments. The system is equipped with a rotating mirror which allows for obtaining distance information in one plane in front of the sensor with a selectable field of view of 100 or 180 degrees (Figure 6.1). The detailed specification of the system is presented in Table 6.1.

The laser unit used with the UASTechLab RMAX system has been modified to reduce its weight from 4.5 to 1.8 kg to make it more applicable for use on-board a UAV. The general system schematics and a photograph of the unit is presented in Figure 6.2. The LMS-291 is attached to a rotating mount which allows for continuous rotation of the sensor around the middle laser beam (solid line in Figure 6.1). This enables obtaining data not only in one plane but in a half-sphere in front of the sensor.

[1]SICK AG. Homepage: `http://www.sick.com`

Range	0-80m with 1cm distance resolution 0-8m with 1mm distance resolution
Angular resolutions	0.25, 0.5, 1 degree
Response times	53ms, 26ms ,13 ms
Data interface	RS232, RS422
Data rates	9.6, 19.2, 38.4, 500 kbaud
Power	approx. 20W
Weight	approx. 4.5 kg (factory) approx. 1.8 kg (after modification)

Table 6.1: SICK LMS-291 parameters.

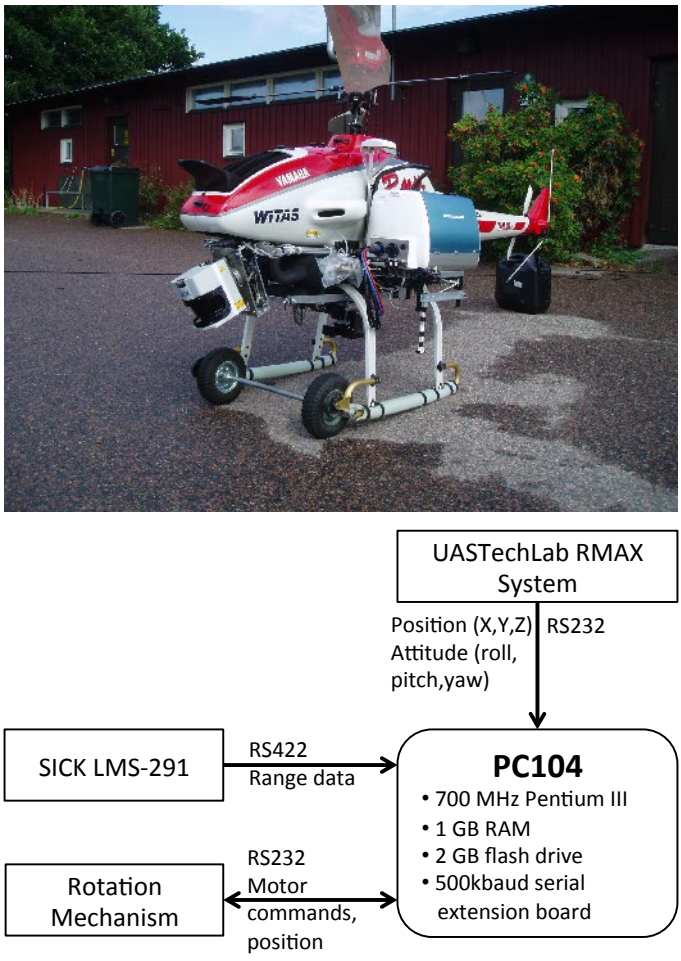


Figure 6.2: A photograph and schematic of the integration of the rotating laser range finder sensor with the UASTechLab RMAX UAV.

Speed	Regulated up to 120 rotations per minute
Angular resolution	0.028 degrees
Range	360 degrees continuous
Data interface	RS232, RS422
Power	Regulated, max approx. 10W
Weight	2 kg

Table 6.2: Rotating laser mount parameters.

The laser rotation mechanism

The in-house developed rotating mount for the laser range finder consists of an aluminum frame which holds the motor, rotating shaft, and a slip ring connecting the power and the serial interface to the laser. The properties of the rotating laser mount are summarized in Table 6.2.

The mount is attached to the body of the UAV through a set of dumpers. They were chosen based on payload vibration measurements during a flight test with a laser substitute of the same weight. The vibrations were measured using an acceleration sensor. Several sets of dumpers were tested and the ones providing the best isolation from influence of the UAV vibrations were chosen. The amount of vibrations transferred to the laser is within the specification of the LMS-291.

The motor of the mount is controlled through a serial interface from a dedicated PC104 computer serving the laser (Figure 6.2). Speed, acceleration and holding power can be adjusted to the needs of a mission. The range data received from the LMS-291, UAV state and the mount motor position are logged by the dedicated computer in a synchronized manner.

6.2 Scan Transformation

We define a *point cloud* (also referred to as *scan*) as a set of range measurements done in one 180 degree revolution of the rotation mechanism. A range measurement (*line-scan*) is a single reading from the laser range finder (Figure 6.1).

In order to compute a 3D environment model, the measurements from the laser range finder (LRF) have to be transformed into the world coordinate system. In short, the laser measurements are relative ranges in the sensor coordinate system (Figure 6.4). The helicopter position and orientation as well as the potential misalignment between the LRF and the rotation mechanism has to be taken into account when transforming the local measurements into the world coordinate system. This allows for merging of a set of line-scans taken at different attitudes and positions of the helicopter in order to transform them into the world coordinate system. All coordinate systems involved in the transformation are depicted in Figure 6.3. The four

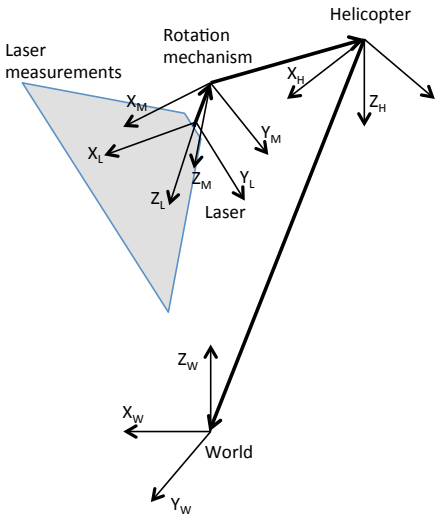


Figure 6.3: Coordinate systems used in the scan transformation in the UASTechLab RMAX platform.

steps of the transformation are presented next. A superscript is used to annotate a particular coordinate system in each of the equations (for example X^L denotes vector X in the LRF coordinate system).

Step 1: Polar to Cartesian coordinate system LRF transformation.

The laser range finder line-scan defined in polar coordinates is:

$$p_i^L = \left[r_i^L, \alpha_i^L, \beta_i^L\right]^T \tag{6.1}$$

where $r = r_0, \ldots, r_n$ is the vector of range data measured by the LRF (Figure 6.4), α is the horizontal angle, and β is the angle of rotation around the X_L axis.
The line-scan p_i^L in Cartesian coordinates of the laser range finder is:

$$p_i^L = \left[x_i^L, y_i^L, z_i^L\right]^T \tag{6.2}$$

The transformation of the polar to the Cartesian coordinate system is calculated by applying the following Cartesian transformation T_{P2C}:

$$\begin{bmatrix} x^L \\ y^L \\ z^L \end{bmatrix} = T_{P2C}(r, \alpha, \beta) = \begin{bmatrix} r\ sin(\beta)\ cos(\alpha) \\ r\ sin(\beta)\ sin(\alpha) \\ r\ cos(\beta) \end{bmatrix} \tag{6.3}$$

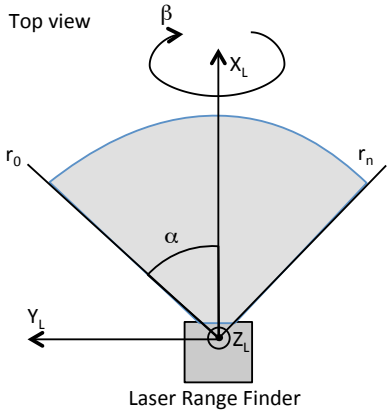


Figure 6.4: Polar and Cartesian coordinate systems of the laser range finder.

Step 2: Transformation of the LRF data to rotation mechanism coordinate system.

Figure 6.5 presents the principle (simplified in two dimensions) of the transformation between the LRF and the rotation mechanism coordinate systems.

In the following steps, a standard generic combined translation and rotation transformation matrix [40] which transforms between a and b coordinate systems is used:

$$T_{a2b} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

The upper left 3x3 matrix represents the rotation transformation, and the $[x\ y\ z]^T$ portion of the matrix represents the translation between the two coordinate systems.

The line-scan can be transformed into the rotation mechanism coordinate system ($p_i^M = \left[x_i^M, y_i^M, z_i^M\right]^T$) as follows:

$$\begin{bmatrix} x^M \\ y^M \\ z^M \\ 1 \end{bmatrix} = T_{L2M} \begin{bmatrix} x^L \\ y^L \\ z^L \\ 1 \end{bmatrix} \tag{6.5}$$

The rotation and translation parameters of the T_{L2M} matrix have been found using the following calibration procedure. First, a set of raw measurements (line-scans) for a predefined scene was collected. The scene included

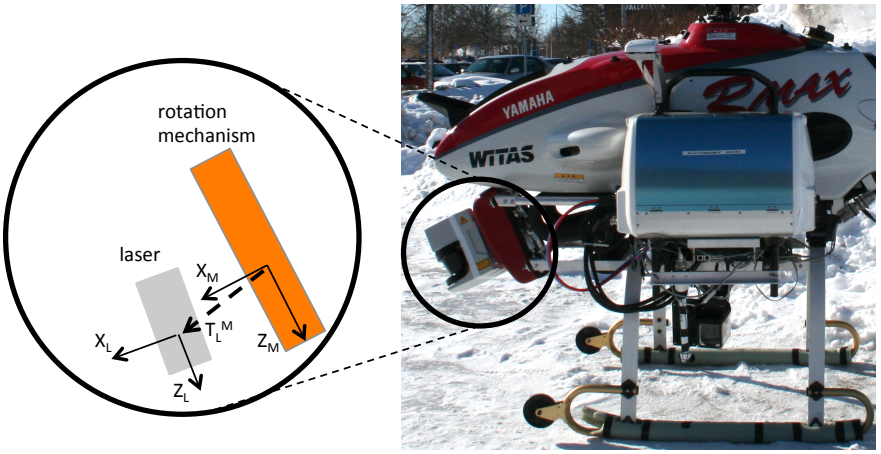


Figure 6.5: Displacement between the laser range finder and the rotation mechanism.

a flat surface with several round reflective markers placed at known positions. The laser beams hitting the markers would be dissipated making the measurements in those regions easy to filter out from the rest of the scan. Second, the transformation parameters were found by manual adjustment of the values and previewing of the resulting scene. Additional verification of the calibrated scene was performed by measuring of the diameter of the markers and the distances between them.

Step 3: Transformation to the helicopter coordinate system.

Similarly as in the previous step, a line-scan in the helicopter coordinate system ($p_i^H = \left[x_i^H, y_i^H, z_i^H\right]^T$) is derived as follows:

$$\begin{bmatrix} x^H \\ y^H \\ z^H \\ 1 \end{bmatrix} = T_{M2H} \begin{bmatrix} x^M \\ y^M \\ z^M \\ 1 \end{bmatrix} \tag{6.6}$$

The parameters of the T_{M2H} matrix have been found by measuring the rotation and translation from the center of the rotation mechanism to the center of the inertial measurement unit of the UASTechLab RMAX UAV.

Step 4: Transformation to the world coordinate system.

As described in chapter 2, the UASTechLab RMAX system is equipped with a GPS receiver and an inertial measurement unit. Data from both

sensors are fused using a Kalman filter. Pitch, roll and yaw angles, and position estimates are the parameters of the last transformation matrix (T_{H2W}) used to transform the line-scan to the world coordinate system ($p_i^W = \left[x_i^W, y_i^W, z_i^W\right]^T$):

$$\begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix} = T_{H2W} \begin{bmatrix} x^H \\ y^H \\ z^H \\ 1 \end{bmatrix} \tag{6.7}$$

To summarize, the measurement taken by the LRF is transformed to the world coordinate system by applying the described four steps. A set of point clouds (consisting of line-scans) acquired during the flight constitutes a 3D model. In the following section, techniques for improving the *raw* 3D map are discussed.

6.3 Scan Alignment

Scan alignment, also referred to as scan registration deals with the problem of finding a transformation between two point clouds. As discussed in the previous section, during the mapping process as the robot traverses the environment a set of collected scans is transformed to the common world coordinate system. Ideally, the mapping process would be finished at this step and a collection of point clouds would form a 3D map of the environment. In practice, however, these measurements are subject to several sources of error. These include, for example, sensor inaccuracy, measurement noise, environmental conditions (such as visibility and temperature), vibrations, the reflectance properties of the target object, and an uncertainty of the position and attitude when the measurements are taken. The degree of influence of a particular error source on the accuracy of the measurement varies. Nevertheless, one of the main contributors is the uncertainty of the robot position and attitude estimations. Comprehensive study of error budget for laser range finder mapping applications is presented by Alshawa et al. [2], Reshetyuk [76].

Figure 6.6 presents a simplified example of the influence of pitch angle uncertainty on the measurement error depending on the distance to an obstacle. The larger the distance to the obstacle, the larger the error of the measurement (e_0 vs e_1). At close distances, for example 20 meters, a 1 degree difference in the pitch estimation will introduce an error of around 0.6 meters in the range measurement. For the distance of 60 meters, the error increases to around 2 meters.

The state estimation in the UASTechLab RMAX platform is done by fusing a GPS position with an attitude estimate delivered by the inertial measurement unit (IMU) of the Yamaha RMAX helicopter (section 2.1).

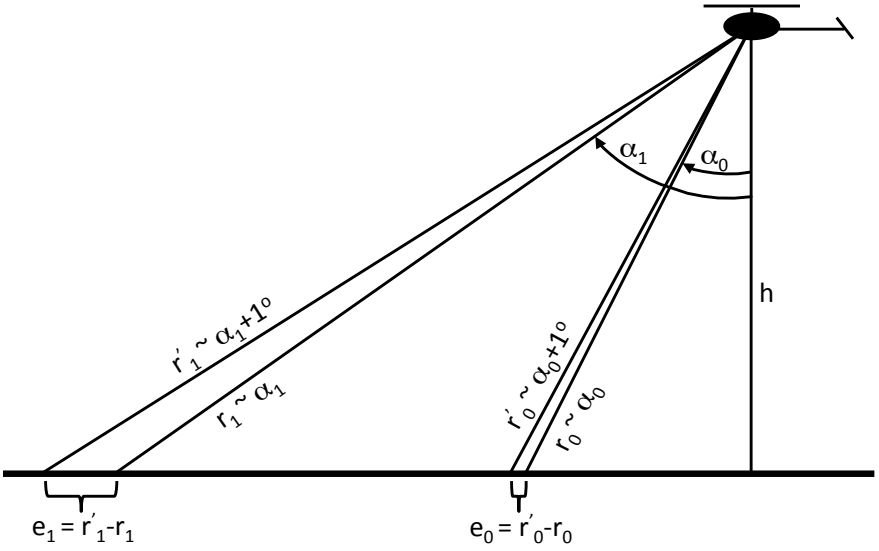


Figure 6.6: A simplified example of the influence of pitch angle uncertainty on the LRF measurement error.

Both of these measurements have a certain accuracy. In the case of the GPS receiver, its nominal accuracy is within 1 meter. The angles estimated by the Yamaha IMU have around 2 deg accuracy. The resulting accuracy of the fused data is of course smaller nevertheless it can introduce a significant error in the acquired 3D map as discussed in the presented example (Figure 6.6). In order to decrease this type of error a registration technique can be used where a pair-wise scan alignment is performed.

Various methods have been proposed in the literature for scan registration. Many of them are suitable for 2D/3D scan alignment often used on robotic platforms. Examples are Iterative Closest Point (ICP [6]), Iterative Dual Correspondences (IDC [57]), probabilistic Iterative Correspondence (pIC [69]), Normal Distributions Transform (NDT [7]), 3D-NDT [59], nonrigid ICP [36, 43] or Least-Squares Matching [58]. Implementations of a number of scan matching methods are also available in the Point Cloud Library (PCL [82]).

As an initial step we have used a variant of the ICP algorithm available in the Mobile Robot Programming Toolkit (MRPT)[2]. It is an implementation of the three-dimensional version of the algorithm presented by Besl and McKay [6].

The ICP algorithm is one of the most commonly used methods for registering a pair of point clouds and in fact many other algorithms (also some

[2]Developed at Machine Perception and Intelligent Robotics Research Group, University of Málaga. Homepage: `http://www.mrpt.org/`

Table 6.3: Results of applying the ICP algorithm on three example scan pairs presented in Figure 6.7.

Source scan ID (#points)	Target scan ID (# points)	Translation [meters]			Rotation [degrees]		
		x	y	z	roll	pitch	yaw
0(24281)	1(24259)	0.49	-0.06	0.39	1.12	-0.71	-0.84
5(17435)	6(18788)	-0.42	-0.077	0.063	-0.90	0.94	-0.20
11(22563)	12(23301)	0.12	0.05	-0.11	0.22	0.09	0.20

listed above) are based on similar intuitions. The ICP iteratively revises the transformation parameters by minimising the sum of squared distances between corresponding points in the two scans. The correspondence between points is assigned by using the nearest neighbor criterion. The algorithm terminates when the sum of square distances is below a certain threshold. The ICP, as any gradient descent method, is applicable when a relatively good starting transformation is known. Otherwise, the convergence to the global optimum cannot be guaranteed and the algorithm may end up in a local minimum.

In the case of data collected using the UASTechLab RMAX UAV, the point clouds are sufficiently close so such a problem does not occur. The state estimation provided by the Kalman filter is accurate enough to provide a good starting point for the registration algorithm.

The ICP was applied on a number of point cloud pairs with various results. It was possible to improve a number of scans with good visual result. Figure 6.7 presents three example scan pairs before (left side) and after the application of the ICP algorithm (right side). White arrows in examples 1 and 2 highlight the misalignment of the point clouds in the vertical planes representing building walls. The white arrow in example 3 emphasizes the offset between the horizontal lines representing a building rooftop. Results of the ICP registration for the three examples are presented in Table 6.3. A *source scan* denotes the initial scan to which a *target scan* is matched.

The maximum translation error between two scans in the presented examples is 49 cm. In 3D maps based on a collection of scans the accumulative error may be even larger. Such errors, if not corrected, can narrow down the operational environment of the UAV. For example, a narrow passage between two buildings can be excluded from the map although, in reality, there is enough space for the UAV to fly through.

Although, the ICP algorithm was successfully applied to a number of point cloud pairs, in some cases it failed due to a large noise in the range measurements. Figure 6.8 shows an example of measurement errors in a single point cloud. White arrows point to the misalignment of range measurements within a scan shown from top and side views. Such errors are most probably related to reflectance properties of the scanned object sur-

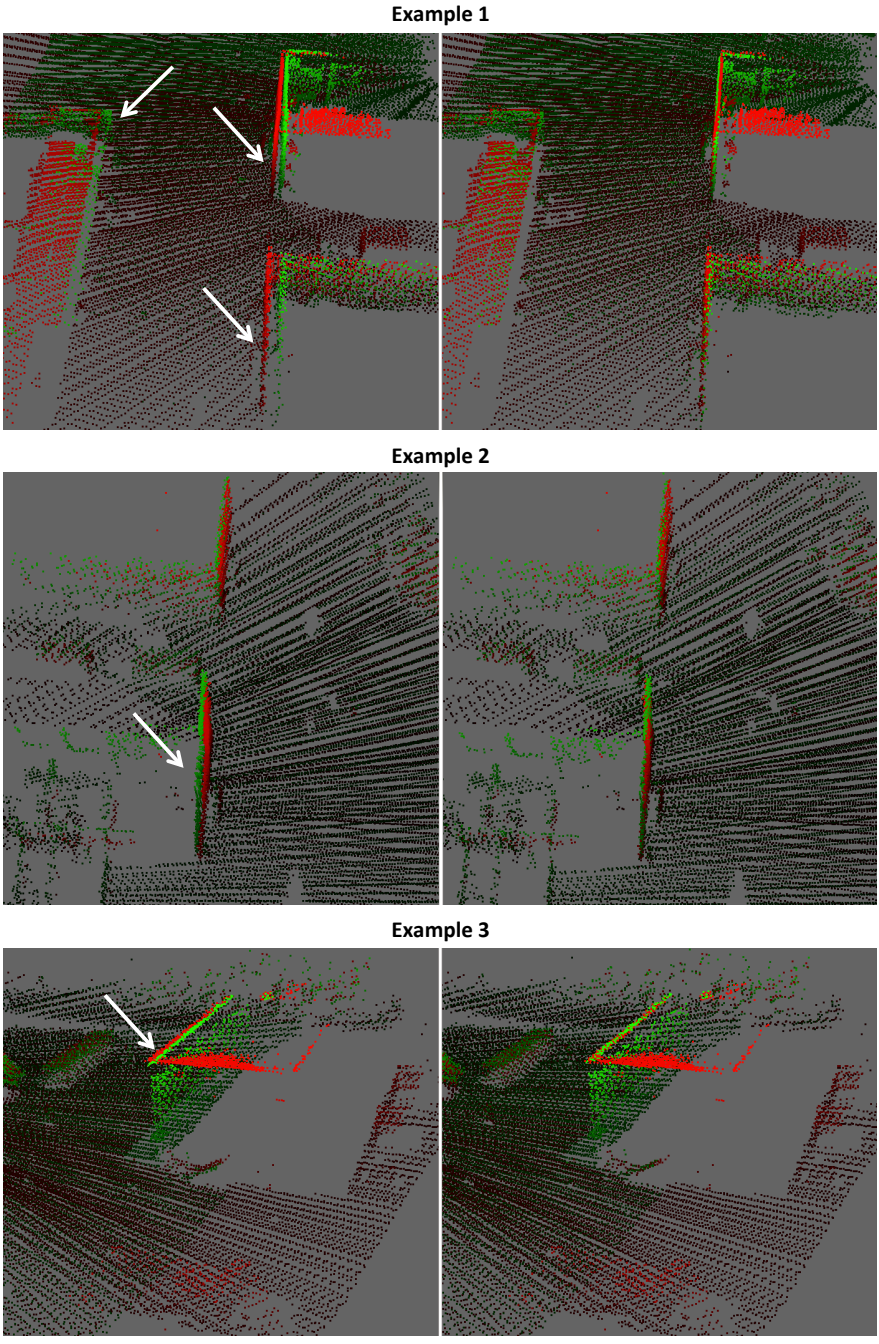


Figure 6.7: Examples of the ICP application.

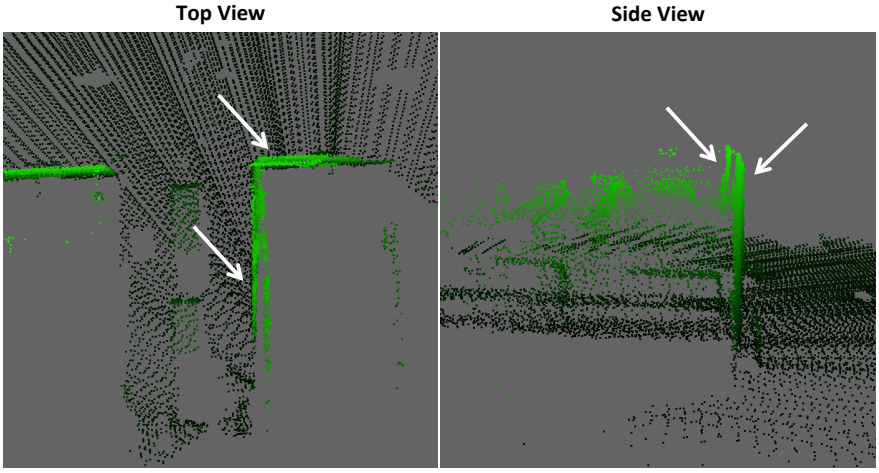


Figure 6.8: Examples of measurement errors in a single point cloud.

face due to several factors as material properties, polarization, and surface colour, moisture, roughness and temperature.

Using the ICP algorithm on the point clouds with large measurement errors prevents the algorithm from finding an appropriate alignment (i.e. converging to the global minimum). In such cases results are often worse than before the application of the alignment procedure. Further investigation of other suitable scan registration methods (previously listed) has to be conducted. Two promising techniques that are more robust towards measurement errors are the 3D-NDT [59] and nonrigid ICP algorithm [36, 43].

6.4 Using 3D Maps for Navigation

The following section presents initial results of the map building procedure using a laser range finder sensor. In the first section, models built during a flight test are presented and used with the existing path planning algorithms. The second section provides a short analysis of the use of the laser range finder for collision avoidance in the context of proposed dynamic replanning framework. Models presented are based on *raw* measurements without applying scan registration techniques.

6.4.1 Static Environment

Several flights were performed during which both the laser range finder data and UAV state estimates were collected. Figure 6.9 presents a reconstructed elevation map of the Revinge flight test area (the left side) focusing on two building structures. A photo of corresponding buildings is presented on the

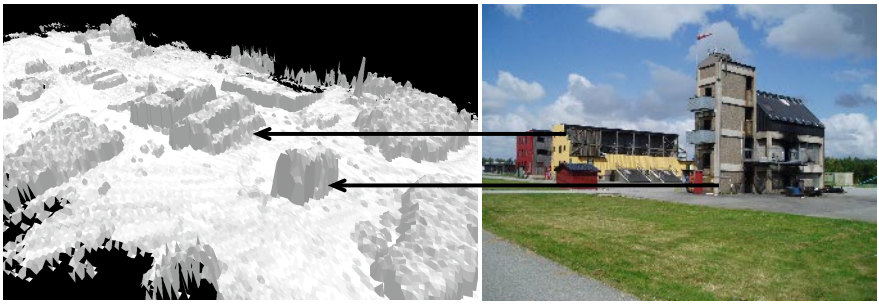


Figure 6.9: Overview of the reconstructed elevation map of the Revinge flight test area based on the laser range finder data (left) and a photo of corresponding building structures (right).

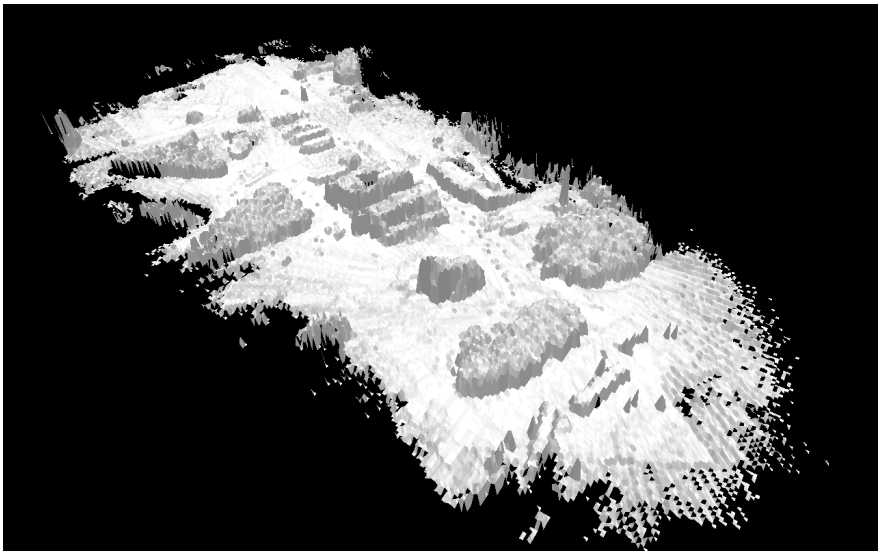


Figure 6.10: Reconstructed elevation map of the Revinge flight test area based on the laser range finder data.

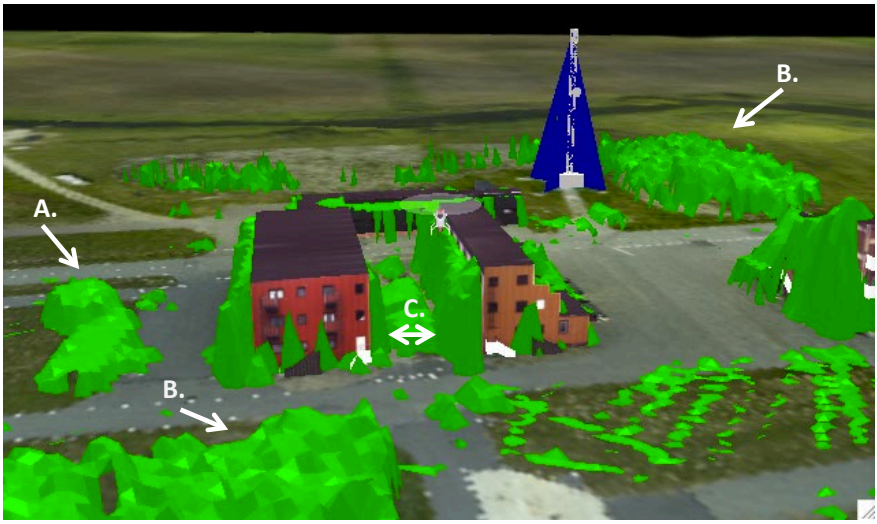


Figure 6.11: Overlay of the new elevation map with the existing Revinge flight test area model.

right side of the figure. The elevation map is built by sampling the LRF data with 1 meter resolution and constructing a set of triangles in order to represent the elevation.

A complete reconstructed elevation map of the area is presented in Figure 6.10. In order to assess the fidelity of the newly generated model an overlay with the existing model was generated and the result is presented in Figure 6.11. The new map includes changes in the environment, i.e. a metal container on the left in the figure (A.) and new vegetation (B.) not present in the existing model.

The new models can be used by the UAV platform for path planning in order to generate collision-free paths. Since generated models are based on noisy measurements a safety margin during the planning is used. In the UASTechLab RMAX UAV, the safety margin of 6 meters is used. An example of path generated by the path planner using the new model is presented in Figure 6.12.

The accuracy of models built with the raw LRF point clouds (without applying the scan matching algorithms) is sufficient for navigation purposes if the necessary safety margins are used. The inaccuracies introduced by the measurement errors and the uncertainty of the UAV state estimate might result in narrowing down the operational environment of the UAV. For example, in Figure 6.11 a narrow passage (C.) can be excluded from the collision-free space, although the corridor between the two buildings is wide enough to fly through. Thus, further investigation of methods for improving the model quality has to be conducted, as outlined in the previous section.

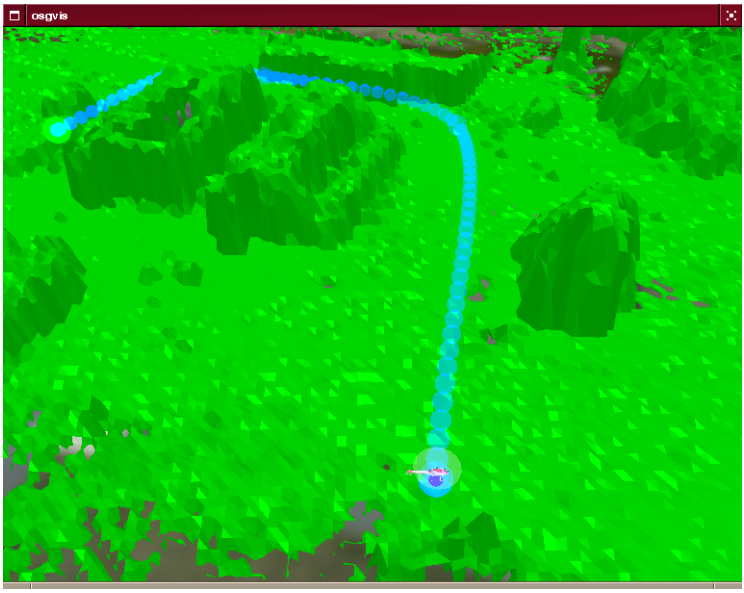


Figure 6.12: Example of path planner use in the reconstructed map of the Revinge area.

6.4.2 Collision Avoidance

This section provides an analysis of using the integrated laser range finder in the context of the dynamic replanning framework presented in chapter 4.

As described previously, the proposed framework uses a planner/strategy tuple in order to repair the path when a new obstacle obstructing the current flight path is detected. In the presented experiments no-fly zones have been used as obstacles. The integrated laser range finder can potentially be used for obstacle detection if its detection range and time will allow for the execution of the proposed technique.

In order to assess the feasibility of using a LRF sensor for this application a timing analysis based on the sensor range and the UASTechLab RMAX UAV dynamic model is presented (Figure 6.13).

The laser has a maximum detection range of 80 meters. Based on experimental evaluation, the *optimal* detection range is 40 meters. In other words, objects at a range between 40 and 80 meters may not always be detected, but if they are, the range measurement is quite precise ($\pm$ 1 meter). Thus, the distance to an obstacle will be known as soon as it is detected.

The time required to fly any given part of a path can be calculated using a dynamic model of the helicopter and the path following control mode (section 2.2.3) with the intended velocity for each segment of the path. Additionally, the required distance for an emergency break at each possible velocity can be calculated. It is shown as a black solid line in Figure 6.13.

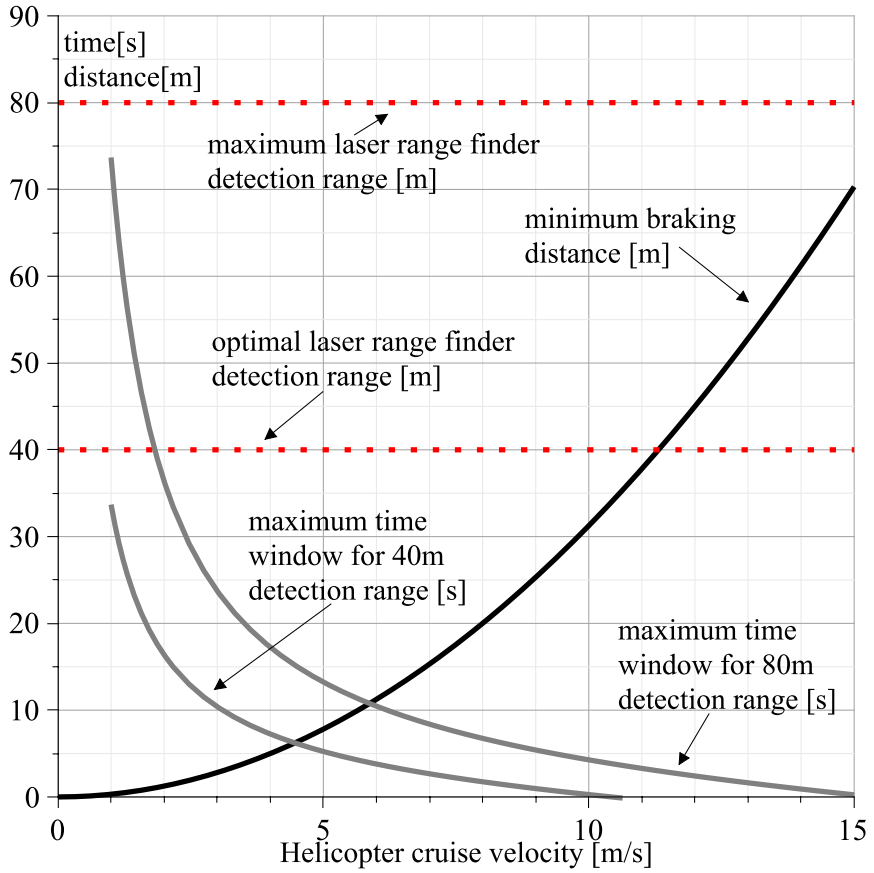


Figure 6.13: The minimal braking distance and time windows for the UASTechLab RMAX UAV as a function of the cruise velocity.

The figure also depicts the resulting time windows available for decision making for the UASTechLab RMAX UAV system as a function of the current velocity. As we are mainly interested in flying at speeds between 10 and 15 m/s, these time windows are quite narrow. For example, assuming a flight at only 10 m/s and detecting an obstacle at a range of 80 meters, the braking distance is 31.5 meters. This gives $80 - 31.5 - 6 = 42.5$ meters available for replanning. This corresponds to 4.25 seconds. If the obstacle is detected at 40 meters, we have only 0.25 seconds available. Nevertheless a sufficient amount of time for applying the collision avoidance techniques exists for wide range of UAV velocities.

The analysis presented shows the feasibility of using the previously described dynamic replanning framework (chapter 5) with the LRF sensor and further integration can be pursued.

Chapter 7

Conclusions

This thesis presents a number of solutions to the navigation and path planning problems. The task of navigation in an airborne system is a complex problem and various techniques are required at different levels of abstraction as shown in each of the chapters in the thesis. The goal of navigation in the UAV domain is to safely fly between different positions in the environment. The process involves executing an appropriate control mode, planning as to which route to take in order to avoid collisions with static obstacles and reacting to any changes in the environment (i.e. new or moving obstacles) while the vehicle is in motion. Additionally, a UAV system has to be able to perceive the changes in the environment in order to react to them. In this chapter a short summary of the techniques presented is provided together with a discussion topics of possible extensions and directions for future work.

Starting at the *control level* a modeling framework for hybrid control systems was presented in chapter 3. It is based on an abstraction of hierarchical and concurrent state machines (HCSM). The framework is used to specify reactive behaviors in the system and to sequentialize the execution of control modes. The HCSM uses a visual formalism of state diagrams which eases the design process. The diagrams are translated into an equivalent textual format and passed to an HCSM interpreter for execution, running on a robotic system. The interpreter is executed in the real-time part of the system which allows dynamic reconfiguration of the system behaviour.

The HCSM framework has been successfully deployed on the UASTechLab RMAX UAV and examples of its uses were presented in chapter 3. In addition, we mention a new extended version (the Extended State Machine, ESM) of the framework. The main difference is the extension of the HCSM to include modelling of the data flow and an explicit task state representation in the formalism. A natural step for future work with respect to design and modelling of control systems is an evaluation of the new ESM framework in the UASTechLab RMAX UAV control system.

In chapter 4 a problem of generating collision-free paths was discussed.

When a UAV mission involving flying between two positions is executed a path planing algorithm is used to generate a collision-free path. Algorithms, PRM and RRT, used in the UASTechLab RMAX UAV were presented. Previously, our UAV platforms could only operate in static environments. In this thesis a dynamic replanning framework implementing a collision avoidance mechanism was presented.

The framework estimates the amount of time available before a collision and chooses one of the proposed replanning *strategies* in order to improve the overall path quality and avoid the need of using non-optimal reactive sense-and-avoid procedures. The replanning *strategy* represents a specific choice of which parts of a path are replanned and which parameters are given to the motion planning algorithm. Timing analysis of the proposed replanning technique together with an experimental flight was presented in chapter 4.

The problem of selecting a particular strategy/planner tuple used during the replanning phase is discussed in chapter 5. The objective is to always choose the strategy/planner tuple that yields the highest quality possible within the available time. As discussed, the planning time and the resulting plan quality (i.e. after the repair) depends on many factors. For example on features of the current plan, the obstructed segment position and size and the relevant areas of the map. A novel method based on machine learning for the selection problem has been presented. For each strategy/planner tuple a set of two prediction models is created. One for estimating the required time for applying the path repair and the other for predicting expected plan quality. The flight time was used as the quality measure. The selection mechanism has been tested in two operational environments with different complexity. The empirical tests presented show promising results: In each test environment, flight times could be improved up to 25% compared to the use of a fixed replanning strategy, resulting in times close to the best achievable with the available planning algorithms.

In future versions of the dynamic replanning framework we plan to include a reactive sense and avoid technique (e.g. model predictive control) in case the time available for replanning is too short and the use of path planners is not possible. Additionally, we plan to continue to explore the possibility of using machine learning techniques in the context of path planning.

An accurate model of the environment is a prerequisite for successful navigation. As described, the path planning algorithms use a geometrical description of the environment to generate collision-free paths. The safety of a UAS operation therefore depends on having an accurate 3D model. Maps may become inaccurate or outdated over time due to changes in the environment (e.g. new building structures, vegetation growth etc.). Thus, adequate sensors and techniques for updating or acquiring new 3D models of the environment are necessary. In chapter 6 initial results of integrating a laser range finder with the UASTechLab RMAX UAV were presented. We

showed how a 3D model of the environment is constructed and presented two applications of acquired models. In the first, a 3D map of the environment is built offline, after an exploratory flight over all building structures is performed and the data collected. The second application relates to the use of a laser range finder sensor for collision avoidance in the context of the proposed dynamic replanning framework. An analysis of potential uses of such a sensor in this context was discussed. Related work in the field of scan alignment and building consistent global 3D maps was presented outlining future research within this topic.

We believe that the combination of the techniques presented in this thesis provides another step towards building comprehensive and robust navigation and path planning frameworks for future UASs.

Bibliography

[1] James S Albus and Fred G Proctor. A Reference Model Architecture for Intelligent Hybrid Control Systems. *Proceedings of the International Federation of Automatic Control (IFAC)*, pages 1–7, June 2000.

[2] M. Alshawa, E. Smigiel, and P. Grussenmeyer. Integration of a Terrestrial Lidar on a Mobile Mapping Platform: first experiences. *Proceedings of the 5th International Symposium on Mobile Mapping Technology*, 2007.

[3] Gregory P Asner, Michael Palace, Michael Keller, Rodrigo Pereira, Jose N M Silva, and Johan C Zweede. Estimating Canopy Structure in an Amazon Forest from Laser Range Finder and IKONOS Satellite Observations. *Biotropica*, 34(4):483–492, dec 2002.

[4] G. Belloni, M. Feroli, A. Ficola, S. Pagnottelli, and P. Valigi. An autonomous aerial vehicle for unmanned security and surveillance operations: design and test. pages 1 –4, sep. 2007. doi: 10.1109/SSRR.2007.4381277.

[5] G erard Berry. The esterel synchronous programming language; design, semantics, implementation. *Science of computer programming*, 1992.

[6] P J Besl and H D McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14 (2):239–256, 1992.

[7] P. Biber and W. Straßer. The normal distributions transform: A new approach to laser scan matching. *Proceedings of the IEEE Inter- national Conference on Intelligent Robots and Systems (IROS)*, 2003.

[8] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The unified modeling language user guide*. Addison-Wesley Professional, 2005.

[9] Taylor L Booth. *Sequential Machines and Automata Theory*. John Wiley & Sons Inc, jan 1967.

[10] R. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. In *Proc. of the 1989 IEEE Int'l Conf. on Robotics and Automation (Vol. 2)*, pages 692–696, May (14-19) 1989.

[11] A. Brooks *et. al.* Towards component-based robotics. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, August (2-6) 2005.

[12] W. Burgard, D. Fox, and S. Thrun. Active mobile robot localization. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, San Mateo, CA, 1997. Morgan Kaufmann.

[13] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proc. IEEE International Conference on Robotics and Automation*, 2005.

[14] John Canny and John Reif. New Lower Bound Techniques for Robot Motion Planning Problems. In *28th Annual Symposium on Foundations of Computer Science*, pages 49–60. IEEE, jun 1987.

[15] Daofang Chang, Houjun Lu, and Weijian Mi. Bulk Terminal Stockpile Automatic Modeling Based on 3D Scanning Technology. In *2010 International Conference on Future Information Technology and Management Engineering (FITME)*, pages 67–70. IEEE, 2010.

[16] D.M Cole and P.M Newman. Using laser range data for 3D SLAM in outdoor environments. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1556–1563, 2006.

[17] G. Conte. *Navigation Functionalities for an Autonomous UAV Helicopter*. 1307, Dept. of Computer and Information Science, March 2007.

[18] G. Conte, S. Duranti, and T. Merz. Dynamic 3D path following for an autonomous helicopter. In *Proc. IFAC Symp. on Intelligent Autonomous Vehicles*, 2004.

[19] G. Conte, M. Hempel, P. Rudol, D. Lundström, S. Duranti, M. Wzorek, and P. Doherty. High accuracy ground target geo-location using autonomous micro aerial vehicle platforms. In *AIAA Guidance, Navigation, and Control Conference, 2008*, volume 26, Honolulu, Hawaii, 2008.

[20] Gianpaolo Conte. *Vision-Based Unmanned Aerial Vehicle Navigation Using Geo-Referenced Information*. PhD thesis, Linköping University, Linköping, 2009.

[21] J. Cremer, J. Kearney, and Y. Papelis. HCSM: a framework for behavior and scenario control in virtual environments. *ACM Transactions on Modeling and Computer Simulation*, 1995.

[22] Frank Dellaert, Steven M. Seitz, Charles E. Thorpe, and Sebastian Thrun. Structure from motion without correspondence. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 557–564, 2000.

[23] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. In *Proc. DARS*, Toulouse, France, 2004.

[24] P. Doherty, P. Haslum, F. Heintz, T. Merz, T. Persson, and B. Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. In *Proc. Int. Symp. on Distributed Autonomous Robotic Systems*, pages 221–230, 2004.

[25] S. Duranti, G. Conte, D. Lundström, P. Rudol, M. Wzorek, and P. Doherty. Linkmav, a prototype rotary wing micro aerial vehicle. In *Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace*, 2007.

[26] Simone Duranti and Gianpaolo Conte. In-flight identification of the augmented flight dynamics of the RMAX unmanned helicopter. *Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace. Toulouse, France.*, 2007.

[27] 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.

[28] J Eker, J.W Janneck, E.A Lee, Jie Liu, Xiaojun Liu, J Ludvig, S Neuendorffer, S Sachs, and Yuhong Xiong. Taming heterogeneity - the Ptolemy approach. In *Proceedings of the IEEE*, pages 127–144, 2003.

[29] David Ferguson and Anthony (Tony) Stentz. Anytime, dynamic planning in high-dimensional search spaces. In *Proc. ICRA*, 2007.

[30] David Ferguson, Nidhi Kalra, and Anthony (Tony) Stentz. Replanning with RRTs. In *Proc. ICRA*, 2006.

[31] U. Forssell and P. Lindskog. Combining Semi-Physical and Neural Network Modeling: An Example of Its Usefulness. *Proceedings of the 11th IFAC Symposium on System Identification (SYSID97)*, pages 795–798, 1997.

[32] E. Gat. On three-layer architectures. *Artificial intelligence and mobile robots*, 1997.

[33] Michael A. Goodrich, Bryan S. Morse, Damon Gerhardt, Joseph L. Cooper, Morgan Quigley, Julie A. Adams, and Curtis Humphrey. Supporting wilderness search and rescue using a camera-equipped mini uav: Research articles. *J. Field Robot.*, 25(1-2):89–110, 2008. ISSN 1556-4959. doi: http://dx.doi.org/10.1002/rob.v25:1/2.

[34] 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.

[35] C. Haddal and J. Gertler. Homeland security: Unmanned aerial vehicles and border surveillance. In *Congressional Research Report, Library of Congress*, 2010.

[36] D. Hahnel, S. Thrun, and W Burgard. An extension of the ICP algorithm for modeling nonrigid objects with mobile robots. *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[37] Mark Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proc. ICML*, 2000. ISBN 1-55860-707-2.

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

[39] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, pages 231–274, 8 1987.

[40] Donald Hearn and M Pauline Baker. *Computer graphics, C version*. Prentice Hall, second edition edition, 1997.

[41] Stefan Hrabar. *Vision-Based 3D Navigation for an Autonomous Helicopter*. PhD thesis, University of South California, 2006.

[42] D. Jones. Power line inspection - a uav concept. page 8 pp., nov. 2005.

[43] Ralf Kaestner, Sebastian Thrun, Michael Montemerlo, and Matt Whalley. A Non-rigid Approach to Scan Alignment and Change Detection Using Range Sensor Data. In Peter Corke and Salah Sukkariah, editors, *Field and Service Robotics*, pages 179–194. Springer Berlin / Heidelberg, Stanford University Robotics Laboratory Computer Science Department Stanford CA, 2006. 10.1007/978-3-540-33453-8_16.

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

[45] M. Kleinehagenbrock *et. al.* Supporting advanced interaction capabilities on a mobile robot with a flexible control system. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, September 28 – October 2 2004.

[46] A. Kleiner, C. Dornhege, R. Kümerle, M. Ruhnke, B. Steder, B. Nebel, P. Doherty, M. Wzorek, P. Rudol, G. Conte, S. Durante, , and D. Lundstr"om. Robocuprescue - robot league team rescuerobots freiburg (germany). In *RoboCup 2006 (CDROM Proceedings), Team Description Paper, Rescue Robot League*, 2006.

[47] Kurt Konolige, Motilal Agrawal, Robert C. Bolles, Cregg Cowan, Martin Fischler, and Brian Gerkey. Outdoor mapping and navigation using stereo vision. In *Proceedings of the International Symposium on Experimental Robotics*, 2006.

[48] T Koo, F Hoffmann, F Mann, and H Shim. Hybrid control of an autonomous helicopter. *IFAC Workshop on Motion Control*, 1998.

[49] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. ICRA*, 2000.

[50] S.M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Department, Iowa State University, 1998.

[51] Steven M LaValle. Planning algorithms, 2004.

[52] D Lecking, O Wulf, V Viereck, J Tödter, and Bernardo Wagner. The RTS-STILL Robotic Fork-Lift. *EURON Technology Transfer Award*.

[53] Yongmin Li. Support vector machine based multi-view face detection and recognition. *Image and Vision Computing*, 22(5):413–427, 2004. doi: 10.1016/.

[54] Zhengrong Li, Yuee Liu, Rodney Walker, Ross Hayward, and Jinglan Zhang. Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform. *Mach. Vision Appl.*, 21(5):677–686, 2010. ISSN 0932-8092. doi: http://dx.doi.org/10.1007/s00138-009-0206-y.

[55] Lennart Ljung. *System Identification: Theory for the User (2nd Edition)*. Prentice Hall, 2 edition, jan 1999.

[56] Market Research Media Ltd. URL `http://www.marketresearchmedia.com/2010/04/09/unmanned-aerial-vehicles-uav-market/`. http://www.marketresearchmedia.com/2010/04/09/unmanned-aerial-vehicles-uav-market/ (accessed August, 2010).

[57] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 1997.

[58] H.-G. Maas. Least-Squares Matching with airborne laserscanning data in a TIN structure. *International Archives of Photogrammetry and Remote Sensing*, 33:548–555, 2000.

[59] Martin Magnusson. *The Three-Dimensional Normal-Distributions Transform — an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* PhD thesis, Örebro University, 2009.

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

[61] Wzorek Mariusz and Doherty Patrick. Preliminary report : Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *24th Annual Workshop of the UK Planning and Scheduling Special Interest Group, PlanSIG,2005*, 2005.

[62] Wzorek Mariusz and Doherty Patrick. The witas uav ground system interface demonstration with a focus on motion and task planning. In *Software Demonstrations at the International Conference on Automated Planning Scheduling (ICAPS-SD)*, pages 36–37, 2006.

[63] Wzorek Mariusz and Doherty Patrick. Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *ICAPS 2006 - The International Conference on Automated Planning Scheduling,2006*, 2006.

[64] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[65] T. Merz. Building a system for autonomous aerial robotics research. In *Proc. IFAC Symp. on Intelligent Autonomous Vehicles*, 2004.

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

[67] T. Merz, P. Rudol, and M. Wzorek. Control system framework for autonomous robots based on extended state machines. *Autonomic and Autonomous Systems, International Conference on*, 0:14, 2006. doi: http://doi.ieeecomputersociety.org/10.1109/ICAS.2006.19.

[68] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. Submitted for publication, 2002.

[69] L. Montesano, J. Minguez, and L. Montano. Probabilistic scan matching for motion estimation in unstructured environments. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3499–3504, 2005.

[70] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[71] Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato. A machine learning approach for feature-sensitive motion planning. In *Proc. Int. Workshop on the Algorithmic Foundations of Robotics*, July 2004.

[72] Andreas Nüchter, Hartmunt Surmann, Kai Lingermann, Joachim Hertzberg, and Sebastian Thrun. 6D SLAM with an Application in autonomous mine mapping. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, pages 1998–2003, 2004.

[73] Per Nyblom. A language translator for robotic task procedure specifications. Master's thesis, Linköping University, 2003.

[74] P-O Pettersson. *Using Randomized Algorithms for Helicopter Path Planning*. Licentiate thesis, Linköping University, 2006.

[75] John H. Reif and H. Wang. The Complexity of the Two Dimensional Curvature-Constrained Shortest-Path Problem. In *Third International Workshop on Algorithmic Foundations of Robotics (WAFR98), Pub. by A. K. Peters Ltd*, pages 1–34, jun 1998.

[76] Y. Reshetyuk. *Investigation and calibration of pulsed time-of-flight terrestrial laser scanners*. Licentiate thesis, KTH Royal Institute of Technology, 2006.

[77] Samuel Rodriguez, Shawna Thomas, Roger Pearce, and Nancy M. Amato. RESAMPL: A region-sensitive adaptive motion planner. In *Proc. Int. Workshop on the Algorithmic Foundations of Robotics*, 2006. ISBN 978-3-540-68404-6.

[78] P. Rudol and P. Doherty. Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery. In *Proceedings of the IEEE Aerospace Conference*, 2008.

[79] P. Rudol, M. Wzorek, G. Conte, and P. Doherty. Micro unmanned aerial vehicle visual servoing for cooperative indoor exploration. In *Proceedings of the IEEE Aerospace Conference*, pages 1–10, March 2008.

[80] P. Rudol, M. Wzorek, R. Zalewski, and P. Doherty. Report on sense and avoid techniques and the prototype sensor suite. Technical report, National Aeronautics Research Program NFFP04-031, Autonomous flight control and decision making capabilities for Mini-UAVs, 2008.

[81] P. Rudol, M. Wzorek, and P. Doherty. Vision-based pose estimation for autonomous indoor navigation of micro-scale unmanned aircraft systems. In *IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 1913–1920, May 2010.

[82] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[83] S. K. Shevade. Improvements to the SMO algorithm for SVM regression. *IEEE Transactions on Neural Networks*, 11(5):1188–1193, 2000. doi: 10.1109/72.870050.

[84] DH Shim, Hoam Chung, and SS Sastry. Conflict-free navigation in unknown urban environments. *Robotics & Automation Magazine, IEEE;*, 13(3):27–33, 2006. doi: 10.1109/MRA.2006.1678136.

[85] Alex Smola. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199, 2004. ISSN 0960-3174. doi: 10.1023/.

[86] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10(3):89–100, 1995.

[87] Zehang Sun. On-road vehicle detection using evolutionary Gabor filter optimization. *IEEE Transactions on Intelligent Transportation Systems*, 6(2):125–137, 2005. doi: 10.1109/.

[88] Hartmut Surmann, Kai Lingemann, Andreas Nu chter, and Joachim Hertzberg. A 3D laser range finder for autonomous mobile robots. In *Proceedings of the 32nd ISR (International Symposium on Robotics)*, 2001.

[89] T.Cox, C. Nagy, M. Skoog, and I. Somers. Civil uav capability assessment. In *Technical report, NASA*, 2004.

[90] S Thrun, M Bennewitz, W Burgard, and Cremers. Minerva: a second-generation museum tour-guide robot. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 3:1999 – 2005 vol.3, 1999. doi: 10.1109/ROBOT.1999.770401.

[91] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont,

L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006. accepted for publication.

[92] B. Üstün, W.J. Melssen, and L.M.C. Buydens. Facilitating the application of support vector regression by using a universal Pearson VII function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81:29–40, 2006.

[93] Vladimir Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. ISBN 0471030031.

[94] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1999. ISBN 0387987800.

[95] G Vosselman, BGH Gorte, and G Sithole. Change detection for updating medium scale maps using laser altimetry. *Proceedings of International Society for Photogrametry and Remote Sensing Congress*, 2004.

[96] A Watson. OMG (Object Management Group) architecture and CORBA (common object request broker architecture) specification. *Distributed Object Management, IEE Colloquium on*, 1994.

[97] M Whalley, G Schulein, and C Theodore. Design and Flight Test Results for a Hemispherical LADAR Developed to Support Unmanned Rotorcraft Urban Operations Research. *American Helicopter Society 64th Annual Forum*, 2008.

[98] Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005. ISBN 0120884070.

[99] M. Wzorek and P. Doherty. Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *Proceedings of the IEEE International Conference on Hybrid Information Technology, ICHIT*, 2006.

[100] M. Wzorek, G. Conte, P. Rudol, T. Merz, S. Duranti, and P. Doherty. From motion planning to control - a navigation framework for an autonomous unmanned aerial vehicle. *Proc. 21st Bristol UAV Systems Conference*, 2006.

[101] M. Wzorek, David Landén, and P. Doherty. Gsm technology as a communication media for an autonomous unmanned aerial vehicle. *Proc. 21st Bristol UAV Systems Conference*, 2006.

[102] Mariusz Wzorek, Jonas Kvarnström, and Patrick Doherty. Choosing path replanning strategies for unmanned aircraft systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.

Avdelning, Institution
Division, Department

AIICS,
Department of Computer
and Information Science
581 83 Linköping

Datum
Date

2011-11-10

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

☒ Licentiatavhandling
☐ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
978-91-7393-037-6

ISRN
LiU-Tek-Lic–2011:48

Serietitel och serienummer **ISSN**
Title of series, numbering 0280–7971

Linköping Studies in Science and Technology

Thesis No. 1509

URL för elektronisk version
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-71147

Titel
Title

Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems

Författare
Author

Mariusz Wzorek

Sammanfattning
Abstract

Unmanned aircraft systems (UASs) are an important future technology with early generations already being used in many areas of application encompassing both military and civilian domains. This thesis proposes a number of integration techniques for combining control-based navigation with more abstract path planning functionality for UASs. These techniques are empirically tested and validated using an RMAX helicopter platform used in the UASTechLab at Linköping University. Although the thesis focuses on helicopter platforms, the techniques are generic in nature and can be used in other robotic systems.

At the *control level* a navigation task is executed by a set of control modes. A framework based on the abstraction of hierarchical concurrent state machines for the design and development of hybrid control systems is presented. The framework is used to specify reactive behaviors and for sequentialisation of control modes. Selected examples of control systems deployed on UASs are presented. Collision-free paths executed at the control level are generated by path planning algorithms. We propose a path replanning framework extending the existing path planners to allow dynamic repair of flight paths when new obstacles or no-fly zones obstructing the current flight path are detected. Additionally, a novel approach to selecting the best path repair strategy based on machine learning technique is presented. A prerequisite for a safe navigation in a real-world environment is an accurate geometrical model. As a step towards building accurate 3D models onboard UASs initial work on the integration of a laser range finder with a helicopter platform is also presented.

Combination of the techniques presented provides another step towards building comprehensive and robust navigation systems for future UASs.

Nyckelord
Keywords

Path planning, motion planning, autonomous Unmanned Aircraft Systems (UAS), Hierarchical Concurrent State Machines (HCSM), UAV.

Department of Computer and Information Science
Linköpings universitet

Licentiate Theses

Linköpings Studies in Science and Technology
Faculty of Arts and Sciences

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge- Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Systems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.

No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.

No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.

No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.

No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.

No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.

No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.

No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.

No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.

No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.

FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.

FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.

No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.

No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.

No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.

No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.

No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.

No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.

FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.

No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.

No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.

No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.

No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.

No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.

No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.

No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.

No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.

FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylprogram, 1995.

No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.

No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.

No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.

No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.

No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.

No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.

FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.

FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.

No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.

No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.

No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.

No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.

No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.

No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.

No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.

FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.

No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.

No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.

No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.

No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.

No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.

No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.

No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.
No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629 **Gunilla Ivefors:** Krigsspel och Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676 **Jan Håkegård**: Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16 **Marie-Therese Christiansson:** Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader, 1999.
No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.

No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800 **Anders Subotic:** Software Quality Inspection, 1999.
No 807 **Svein Bergum**: Managerial communication in telework, 2000.
No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823 **Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37 **Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40 **Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863 **Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management in Information-providing Dialogue systems, 2001.
FiF-a 47 **Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer: Informationssystemarkitektur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.
No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
No 906 **Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.
No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
FiF-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
FiF-a-51 **Per Oscarsson:** Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
No 938 **Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.
No 942 **Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
No 958 **Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
FiF-a 61 **Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
No 985 **Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
No 982 **Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
No 989 **Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
No 990 **Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.
No 991 **Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.

No 999 **Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002.
No 1000 **Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
No 1001 **Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
No 988 **Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
FiF-a 62 **Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
No 1003 **Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
No 1005 **Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
No 1008 **Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
No 1010 **Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
No 1015 **Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
No 1018 **Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
No 1022 **Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
FiF-a 65 **Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
No 1024 **Aleksandra Tešanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.
No 1034 **Arja Vainio-Larsson**: Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
No 1033 **Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
FiF-a 69 **Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
No 1049 **Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
No 1052 **Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
No 1054 **Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
FiF-a 71 **Emma Eliason:** Effektanalys av IT-systems handlingsutrymme, 2003.
No 1055 **Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
No 1058 **Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
FiF-a 73 **Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
No 1079 **Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
No 1084 **Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
FiF-a 74 **Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
No 1094 **Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
No 1095 **Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
No 1099 **Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
No 1110 **Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
No 1116 **Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
FiF-a 77 **Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
No 1126 **Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
No 1127 **Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
No 1132 **Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
No 1130 **Ioan Chisalita:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
No 1138 **Thomas Gustafsson:** Maintaining Data Consistency in Embedded Databases for Vehicular Systems, 2004.
No 1149 **Vaida Jakoniené:** A Study in Integrating Multiple Biological Data Sources, 2005.
No 1156 **Abdil Rashid Mohamed**: High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
No 1162 **Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
No 1165 **Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
FiF-a 84 **Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
No 1166 **Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
No 1167 **Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
No 1168 **Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
FiF-a 85 **Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
No 1171 **Yu-Hsing Huang:** A systemic traffic accident model, 2005.
FiF-a 86 **Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
No 1172 **Petter Ahlström**: Affärsstrategier för seniorbostadsmarknaden, 2005.
No 1183 **Mathias Cöster**: Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
No 1184 **Åsa Horzella**: Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
No 1185 **Maria Kollberg**: Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
No 1190 **David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.

No 1191 **Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.
No 1192 **Nicklas Bergfeldt:** Towards Detached Communication for Robot Cooperation, 2005.
No 1194 **Dennis Maciuszek:** Towards Dependable Virtual Companions for Later Life, 2005.
No 1204 **Beatrice Alenljung:** Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005.
No 1206 **Anders Larsson:** System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
No 1207 **John Wilander:** Policy and Implementation Assurance for Software Security, 2005.
No 1209 **Andreas Käll:** Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
No 1225 **He Tan:** Aligning and Merging Biomedical Ontologies, 2006.
No 1228 **Artur Wilk:** Descriptive Types for XML Query Language Xcerpt, 2006.
No 1229 **Per Olof Pettersson:** Sampling-based Path Planning for an Autonomous Helicopter, 2006.
No 1231 **Kalle Burbeck:** Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.
No 1233 **Daniela Mihailescu:** Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.
No 1244 **Jörgen Skågeby:** Public and Non-public gifting on the Internet, 2006.
No 1248 **Karolina Eliasson:** The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.
No 1263 **Misook Park-Westman:** Managing Competence Development Programs in a Cross-Cultural Organisation - What are the Barriers and Enablers, 2006.
FiF-a 90 **Amra Halilovic:** Ett praktikperspektiv på hantering av mjukvarukomponenter, 2006.
No 1272 **Raquel Flodström**: A Framework for the Strategic Management of Information Technology, 2006.
No 1277 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Embedded Systems, 2006.
No 1283 **Håkan Hasewinkel:** A Blueprint for Using Commercial Games off the Shelf in Defence Training, Education and Research Simulations, 2006.
FiF-a 91 **Hanna Broberg:** Verksamhetsanpassade IT-stöd - Designteori och metod, 2006.
No 1286 **Robert Kaminski:** Towards an XML Document Restructuring Framework, 2006.
No 1293 **Jiri Trnka:** Prerequisites for data sharing in emergency management, 2007.
No 1302 **Björn Hägglund:** A Framework for Designing Constraint Stores, 2007.
No 1303 **Daniel Andreasson**: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.
No 1305 **Magnus Ingmarsson:** Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.
No 1306 **Gustaf Svedjemo**: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.
No 1307 **Gianpaolo Conte**: Navigation Functionalities for an Autonomous UAV Helicopter, 2007.
No 1309 **Ola Leifler**: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.
No 1312 **Henrik Svensson**: Embodied simulation as off-line representation, 2007.
No 1313 **Zhiyuan He:** System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.
No 1317 **Jonas Elmqvist:** Components, Safety Interfaces and Compositional Analysis, 2007.
No 1320 **Håkan Sundblad:** Question Classification in Question Answering Systems, 2007.
No 1323 **Magnus Lundqvist**: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.
No 1329 **Martin Magnusson:** Deductive Planning and Composite Actions in Temporal Action Logic, 2007.
No 1331 **Mikael Asplund:** Restoring Consistency after Network Partitions, 2007.
No 1332 **Martin Fransson:** Towards Individualized Drug Dosage - General Methods and Case Studies, 2007.
No 1333 **Karin Camara:** A Visual Query Language Served by a Multi-sensor Environment, 2007.
No 1337 **David Broman:** Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.
No 1339 **Mikhail Chalabine:** Invasive Interactive Parallelization, 2007.
No 1351 **Susanna Nilsson:** A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.
No 1353 **Shanai Ardi:** A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.
No 1356 **Erik Kuiper:** Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.
No 1359 **Jana Rambusch**: Situated Play, 2008.
No 1361 **Martin Karresand:** Completing the Picture - Fragments and Back Again, 2008.
No 1363 **Per Nyblom:** Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.
No 1371 **Fredrik Lantz:**Terrain Object Recognition and Context Fusion for Decision Support, 2008.
No 1373 **Martin Östlund:** Assistance Plus: 3D-mediated Advice-giving on Pharmaceutical Products, 2008.
No 1381 **Håkan Lundvall:** Automatic Parallelization using Pipelining for Equation-Based Simulation Languages, 2008.
No 1386 **Mirko Thorstensson:** Using Observers for Model Based Data Collection in Distributed Tactical Operations, 2008.
No 1387 **Bahlol Rahimi:** Implementation of Health Information Systems, 2008.
No 1392 **Maria Holmqvist:** Word Alignment by Re-using Parallel Phrases, 2008.
No 1393 **Mattias Eriksson**: Integrated Software Pipelining, 2009.
No 1401 **Annika Öhgren:** Towards an Ontology Development Methodology for Small and Medium-sized Enterprises, 2009.
No 1410 **Rickard Holsmark:** Deadlock Free Routing in Mesh Networks on Chip with Regions, 2009.
No 1421 **Sara Stymne:** Compound Processing for Phrase-Based Statistical Machine Translation, 2009.
No 1427 **Tommy Ellqvist**: Supporting Scientific Collaboration through Workflows and Provenance, 2009.
No 1450 **Fabian Segelström:** Visualisations in Service Design, 2010.
No 1459 **Min Bao**: System Level Techniques for Temperature-Aware Energy Optimization, 2010.
No 1466 **Mohammad Saifullah**: Exploring Biologically Inspired Interactive Networks for Object Recognition, 2011

No 1468 **Qiang Liu**: Dealing with Missing Mappings and Structure in a Network of Ontologies, 2011.
No 1469 **Ruxandra Pop**: Mapping Concurrent Applications to Multiprocessor Systems with Multithreaded Processors and Network on Chip-Based Interconnections, 2011.
No 1476 **Per-Magnus Olsson**: Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles, 2011.
No 1481 **Anna Vapen**: Contributions to Web Authentication for Untrusted Computers, 2011.
No 1485 **Loove Broms:** Sustainable Interactions: Studies in the Design of Energy Awareness Artefacts, 2011.
FiF-a 101 **Johan Blomkvist:** Conceptualising Prototypes in Service Design, 2011.
No 1490 **Håkan Warnquist:** Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis, 2011.
No 1503 **Jakob Rosén:** Predictable Real-Time Applications on Multiprocessor Systems-on-Chip, 2011.
No 1504 **Usman Dastgeer:** Skeleton Programming for Heterogeneous GPU-based Systems, 2011.
No 1506 **David Landén:** Complex Task Allocation for Delegation: From Theory to Practice, 2011.
No 1509 **Mariusz Wzorek:** Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems, 2011.
No 1510 **Piotr Rudol:** Increasing Autonomy of Unmanned Aircraft Systems Through the Use of Imaging Sensors, 2011.