Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Helicopter

Per Olof Pettersson and Patrick Doherty
Linköping University
Department of Computer and Information Science
581 83 Linköping, Sweden
{peope, patdo}@ida.liu.se

Abstract— **The emerging area of intelligent unmanned aerial vehicle (UAV) research has shown rapid development in recent years and offers a great number of research challenges for artificial intelligence. For both military and civil applications, there is a desire to develop more sophisticated UAV platforms where the emphasis is placed on development of intelligent capabilities. Imagine a mission scenario where a UAV is supplied with a 3D model of a region containing buildings and road structures and is instructed to fly to an arbitrary number of building structures and collect video streams of each of the building's respective facades. In this article, we describe a fully operational UAV platform which can achieve such missions autonomously. We focus on the path planner integrated with the platform which can generate collision free paths autonomously during such missions. It is based on the use of probabilistic roadmaps. The path planner has been tested together with the UAV platform in an urban environment used for UAV experimentation.**

I. Introduction

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

Intelligent UAVs will play an equally important role in civil applications. For both military and civil applications, there is a desire to develop more sophisticated UAV platforms where the emphasis is placed on development of intelligent capabilities. Focus in research has moved from low-level control towards a combination of low-level and decision-level control integrated in sophisticated software architectures. These should also integrate well with larger net-centric based C^4I^2 systems. Such platforms are a prerequisite for supporting the capabilities required for the increasingly more complex mission tasks on the horizon and an ideal testbed for the development and integration of AI technologies.

The WITAS[1] Unmanned Aerial Vehicle Project [2] is a long-term basic research project whose main objectives are the development of an integrated hardware/software VTOL (Vertical Take-Off and Landing) platform for fully autonomous missions and its deployment in applications such as traffic monitoring and surveillance, emergency services assistance, photogrammetry and surveying.

Basic and applied research in the project covers a wide range of topics which include the development of a distributed architecture for autonomous unmanned aerial vehicles. In addition to the software architecture, many AI technologies have been developed such as path planners, task planners, chronicle recognition and situational awareness techniques. The architecture supports modular and distributed integration of these and any additional functionalities added in the future.

An experimental version of the WITAS UAV hardware/software platform has been developed and successfully used in a VTOL system capable of achieving a number of complex autonomous missions flown in a challenging *urban* environment populated with building and road structures. In one mission, our UAV autonomously tracked a moving vehicle for up to 20 minutes. In another, several building structures in the test area were arbitrarily chosen as survey targets and our UAV autonomously generated collision free path plans to fly to each and take photographs of each of the building's facades. This and similar missions have been successfully executed.

Figure 1 shows an aerial photo of our primary flight test area located in Revinge, Sweden. An emergency services training school is located in this area and consists of a collection of buildings, roads and even makeshift car and train accidents. This provides an ideal flight test area for experimenting with traffic surveillance, photogrammetric and surveying scenarios, in addition to scenarios involving emergency services. We have also constructed an accurate 3D model for this area which has proven invaluable in simulation tests and as a visualization tool. Parts of the model are integrated in the on-board geographic information system (GIS) and are used by many of the services in the architecture including the path planner which will be considered in detail in this paper.

In the remainder of the paper, we will concentrate on a solution to path planning for our UAV based on the use of probabilistic roadmaps. This path planning module is implemented and used in the on-board system. Before providing

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

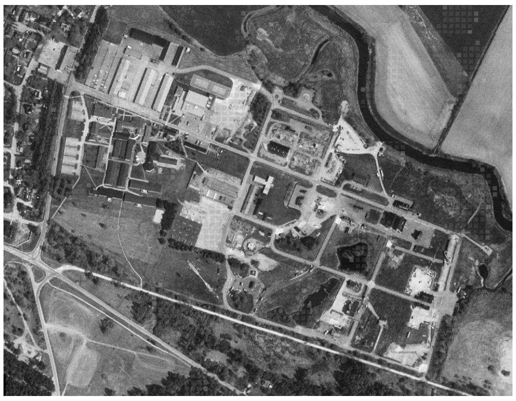


Fig. 1. Aerial photo over Revinge, Sweden

details, the hardware and software platforms for the WITAS UAV will be considered briefly in order to provide a context for understanding how path planning is integrated with the system.

II. The VTOL and Hardware Platform

The WITAS Project UAV platform we use is a slightly modified Yamaha RMAX (figure 2). It has a total length of 3.6 m (incl. main rotor), a maximum take-off weight of 95 kg, and is powered by a 21 hp two-stroke engine. Yamaha equipped the radio controlled RMAX with an attitude sensor (YAS) and an attitude control system (YACS). Figure 3 shows a high-level schematic of the hardware platform that we have built and integrated with the RMAX platform. The hardware platform consists of three PC104 embedded computers (figure 3).[2]

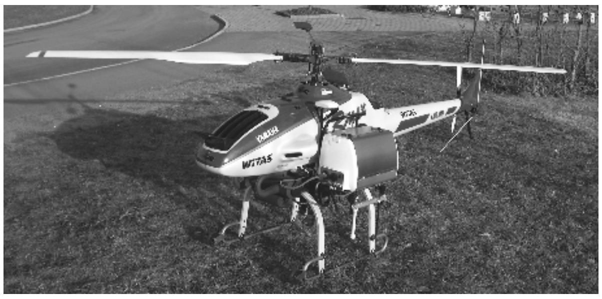


Fig. 2. The WITAS RMAX Helicopter

III. The Software Platform

CORBA[3] has been chosen as a basis for the design and implementation of a loosely coupled distributed software architecture for the WITAS aerial robotic system [3]. It is believed that this is a good choice which enables us to manage the complexity of a deliberative/reactive (D/R) software architecture with as much functionality as we require for our applications. It also ensures clean and flexible interfacing to the deliberative and control components in addition to the hardware platform via the use of IDL (Interface Definition Language).

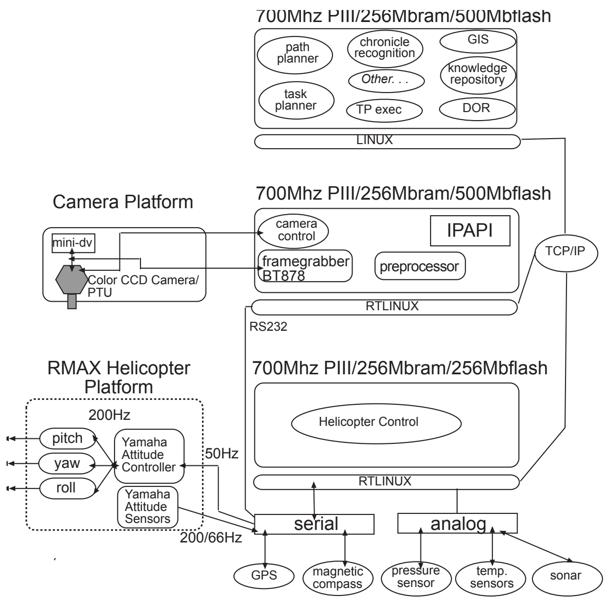


Fig. 3. On-Board Hardware Schematic

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

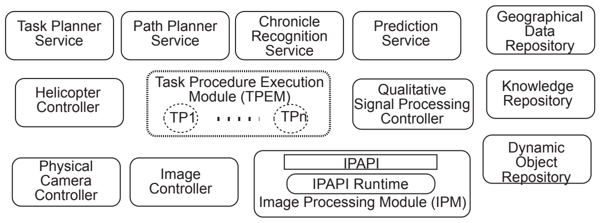
Many of the functionalities which are part of the architecture can be viewed as CORBA objects or collections of objects, where the communication infrastructure is provided by CORBA facilities and other services such as real-time and standard event channels. This architectural choice provides us with an ideal development environment and versatile runtime system with built-in scalability, modularity, software relocatability on various hardware configurations, performance (real-time event channels and schedulers), and support for plug-and-play software modules.

Figure 4 depicts an (incomplete) high-level schematic of some of the software components used in the architecture.

2. The primary flight control (PFC) system consists of a PIII (700MHz) processor, a wireless modem (serial line RS232C) and the following sensors: an integrated INS/DGPS (serial), a barometric altitude sensor (analog), a sonar and infrared altimeter (analog), and a compass (serial). It is connected to the YAS and YACS (serial), the image processing computer (serial) and the deliberative computer (Ethernet). The image processing (IP) system consists of a second PC104 embedded computer (PIII 700MHz), a color CCD camera (S-VIDEO, serial interface for control) mounted on a pan/tilt unit (serial), a video transmitter (composite video) and a recorder (miniDV). The deliberative/reactive (D/R) system runs on a third PC104 embedded computer (PIII 700MHz) which is connected to the PFC system with Ethernet using CORBA event channels and standard CORBA method calls.

3. TAO/ACE [7] is currently being used. **T**he **A**ce **O**rb is an open source implementation of CORBA 2.6.

Each of these may be viewed as a CORBA server/client providing or requesting services from each other and receiving data and events through both real-time and standard event channels. Each of these functionalities has been implemented and all are being used and developed in our applications. A great deal of effort has gone into the development of a



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

Fig. 4. Some deliberative, reactive and control services

control system for the WITAS UAV which incorporates a number of different control modes and includes a high-level interface to the control system. This enables other parts of the architecture to call the appropriate control modes dynamically during the execution of a mission. The ability to switch modes contingently is a fundamental functionality in the architecture and can be programmed into the task procedures associated with the reactive component in the architecture. We have developed and tested the following autonomous flight control modes:

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

These modes and their combinations have been successfully demonstrated in a number of missions at the Revinge flight test area.

A. A Path Planning Mission Scenario

One application area we have focused on is surveying and data collection. The WITAS UAV contains an on-board GIS which includes a terrain model of the Revinge area accurate to within decimeters in the x, y, z directions and equally accurate models of building and road structures. A plan template has been designed using a task procedure which can be parameterized with an arbitrary number of building structure identifiers from Revinge. When this task procedure is called, it iterates through each of the building identifiers and generates a set of camera positions (one fore each building facade) and corresponding positions for the UAV to take photos of each facade. The camera positions are computed using additional functionality built into the GIS.

The UAV must then fly to each of these positions, hover and yaw appropriately and take pictures of the facade before moving onto the next. To fly to these positions the task procedure calls the path planning service which generates a collision free path passing all positions. The resulting 3D path consists of a segmented trajectory, with the waypoints corresponding to camera positions marked with an index. Each of these segments is passed to functional units in the primary flight controller which call an appropriate flight mode to follow the trajectories. Progress is monitored by the calling task procedure.

This is a complex scenario which must use deliberative (path planner), reactive (task procedures) and control (trajectory following, hovering modes) functionalities concurrently in an integrated manner. In the next two sections, we show how this problem has been solved efficiently using probabilistic roadmaps as a basis for the path planner. We then consider actual survey missions similar to the above that have been executed successfully by our UAV.

IV. The Path Planner

The path planner used for the helicopter is an adaptation of the probabilistic roadmap (PRM) algorithm [5], to our application domain. The problem of finding an optimal path between two robot configurations in a high-dimensional configuration space such as the helicopter's is intractable. The PRM algorithm *hedges* the intractability problem by sacrificing completeness and optimality, while utilizing information about the environment known in advance for improving the runtime efficiency. The PRM-algorithm works in two phases, one off-line and the other during runtime. The main processing stages for our adaptation of the PRM algorithm are shown in figure 5.

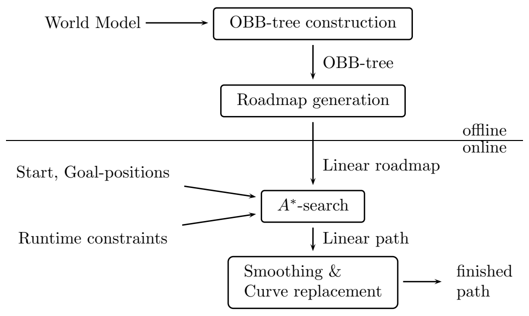
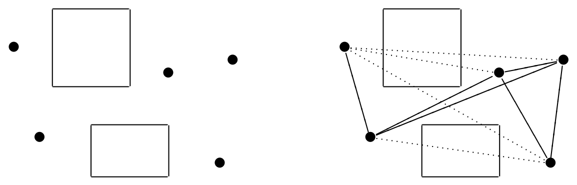


Fig. 5. Generating a Flight Trajectory

In the offline phase, a roadmap graph is generated for the area of interest (e.g.. Revinge). An example of this process is shown in figure 6. Helicopter configurations[4] are randomly generated and checked for collisions with the model and



(a) Adding random configurations.

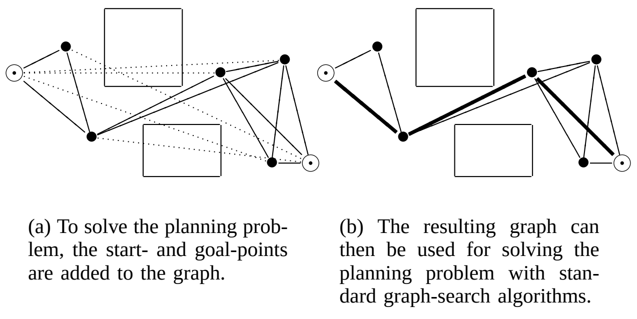
(b) Connect nodes if possible (continuous lines), but not if a connection would intersect obstacles (dashed lines).

Fig. 6. Construction of the roadmap.

4. A helicopter configuration consists of three coordinates for position and a vector describing the direction of flight. The orientation of the helicopter is omitted and is handled independently by the control system.

the collision free configurations are added as nodes to the graph (figure 6(a)). An attempt is then made to connect the collision-free configurations using a local path planner that takes the kinematic and dynamic constraints of the helicopter into account. Each of the local paths generated also have to be checked for collisions and edges are added between configurations where the connection is collision free (figure 6(b)). The collision checker, used to check whether a given curve or line intersects any obstacle in the environment, is based on the OBBTree-algorithm [4], that uses a tree of oriented bounding boxes as its central component.

There are a number of choices that can be made for the local path planner at this stage depending on how much or little work is chosen to be done during run-time. In the first case, which is more in keeping with the original PRM algorithm, the roadmap is generated in the off-line phase with spline-curves, taking nonholonomic constraints into account. Another alternative, the one described in diagram 5, is to initially ignore the nonholonomic constraints of the helicopter in the off-line phase and add them as refinements to the plan in the smoothing and curve replacement phase during run-time as will be described below. In this case straight lines is used for connecting the nodes. We have experimented with variations of both of these approaches.


(a) To solve the planning problem, the start- and goal-points are added to the graph.

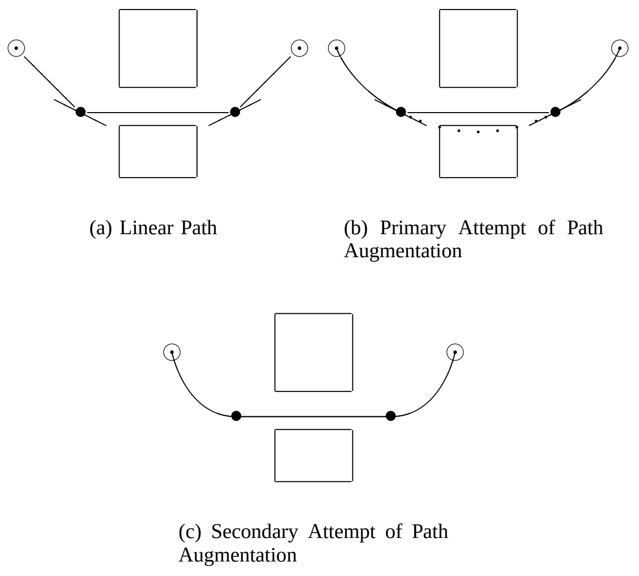
(b) The resulting graph can then be used for solving the planning problem with standard graph-search algorithms.

Fig. 7. Online planning using the precompiled roadmap.

During the mission or run-time phase, the path planning service is called with an initial and goal helicopter configuration. An attempt is made to connect the two configurations to the previously generated roadmap using the local path planner which is illustrated in figure 7(a). If this is successful, an A* search is used on the graph to generate a multi-segmented trajectory (figure 7(b)). Additional constraints may influence which parts of the roadmap that are legally usable as will be described below. The resulting path is smoothed and in the case of a straight-line roadmap, the straight segments are replaced with spline curves wherever possible, using the local path planner.

A. Some Modifications to the Standard Probabilistic Roadmap Algorithm

One of the most important modifications we have made to the standard PRM algorithm is to delay some of the processing of constraints normally done off-line in the roadmap generation stage and instead do the processing during runtime


(a) Linear Path

(b) Primary Attempt of Path Augmentation

(c) Secondary Attempt of Path Augmentation

Fig. 8. Augmentation of linear paths to cubic paths required for smooth flight.

when path plans are actually being generated. As long as it is not too expensive (meaning slow) to process certain types of constraints during runtime, this approach is of great benefit since one can dynamically apply constraints rather than building them into the roadmap in the off-line phase. We consider two modifications to the PRM algorithm which have been incorporated into our path planner.

1) Multi-level nonholonomic roadmap planning: The standard probabilistic roadmap algorithm is formulated for robots where the dynamics plays a small role for the behavior of the system. This assumption is more or less true for a helicopter flying at low speed in a hovering mode. However, when the speed is increased the helicopter is no longer able to negotiate turns of too small radius, which imposes similar demands on the planner as the nonholonomic constraints on car-like robots.

There are a number of proposals for dealing with nonholonomic constraints in the probabilistic roadmap paradigm. One approach taken in [8], is to first solve a relaxed problem using only the holonomic constraints, and then refine the solution by adding nonholonomic constraints one at a time. For robots respecting certain topological properties (e.g. all locally controllable robots as well as car-like robots), it is always possible to upgrade the solution so as to take the nonholonomic constraints into account as long as there exists a non-zero margin between the solution to the relaxed problem and the obstacles.

Inspired by this approach, the path planner for our UAV initially generates a piecewise linear plan, which is later refined to the preferable piecewise cubic curve required for smooth high speed flight. This replacement is done by associating direction vectors to each node parallel to a line between the two adjacent nodes (see figure 8(a)). The line-segments are then replaced with cubic space curves where that is possible without intersecting obstacles as shown in figure 8(b). For segments where a collision occur, e.g., the middle segment in figure 8(b), an attempt to is made to align the two adjacent segments which is shown in figure 8(c). Only if this replacement also fails, a sharp corner will remain in the final path, where the helicopter will have to stop into hover. In practice, this situation rarely occurs. The few cases where it does occur is usually in cramped locations with many nearby obstacles and it is not unreasonable for the helicopter to go into hover when changing direction of flight in such environments.

This method of handling nonholonomic constraints gives a drastic improvement for planning with local path planners that have a limited ability to connect configurations. With an earlier local planner that only allows connections if the target waypoint is within 45° from the direction of flight, the multi-level method has a success-rate of over 99 % in the test-flight area using a roadmap with only 500 nodes. For the straight forward method with cubic curves already in the roadmap, a graph with more than 3000 nodes is needed to reach this success-rate.

2) Delayed Constraint Handling: It can also be useful to delay ordinary holonomic constraints, that are not known during roadmap construction time, until the runtime phase. Recompiling the roadmap at this stage is often not feasible since it can take several minutes to do in larger environments. However, for constraints that can be evaluated rapidly, it is possible to check them during the graph search without a big performance penalty. This is done by testing the constraints for each outbound edge when expanding a node during the graph search, and only add nodes for which the test is positive to the search-queue. In this way the planner can make use of the information that was built up during the roadmap construction phase, even if new constraints are added at a later time.

Currently we have implemented runtime constraints pertaining to maximum and minimum altitude, forbidden regions and limits on ascent-/descent-rate, which are useful when setting up UAV missions in the field. In fact, this extension to PRMs is necessary in practical applications. Mission constraints rapidly change and one can not afford to recompile the roadmap to incorporate such constraints. This extension to the PRM-algorithm grew out of a practical necessity as we experimented with PRMs in the field and found the basic algorithm lacking in this respect.

B. Local Path Planner and Helicopter Controller

The control mode that carries out the plan produced by the path planner takes a cubic polynomial space curve parameterized from 0 to 1 as its input. The curve is derived from the two end positions and the direction of flight through these points. This leaves one degree of freedom at each end in the magnitude of the derivative, which is currently set to the distance between the two points and which generally produces nice curves. In the future, the magnitude of this vector can be used to adjust the curvature of the curve segment to suit the requested flight speed.

Even if the trajectory following controller is able to fly most well-behaved curves, there are a number of limits in the physical platform and the current controller implementation that makes different curve-forms more or less effective to fly. The responsibility of staying inside these limits is shared between the controller, the task procedure that calls the path planner and the path planner, while most of the constraints are handled by the controller.

The limitations include maximum acceleration which puts a limit on how fast the helicopter can fly along a curve with a certain radius as well as ascent rate. The descent rate is even more limited due to aerodynamic effects. Limitations that stem from the system architecture include a timeout limit which is a safety feature in the controller that requires the next curve segment to be ready some time in advance in order to be able to stop in time if no next segment arrives.

Currently the path planner is only aware of the timeout limitation on speed while the other speed limits are imposed dynamically by the controller. Using A^* search, the path planner currently optimizes only on shortest distance but we are planning to incorporate the limitations mentioned in a flight time estimate so that the path planner can also optimize on flight time. The trajectory-following control-mode used is described in more detail in Conte [1] and the low-level control architecture is described in Merz [6].

C. Collision Checker

Both during roadmap-construction and online path planning queries, possible paths have to be tested for collision.

The collision checking algorithm used for the path planner in the WITAS-project is based on the OBB-tree algorithm presented in [4]. The OBBTree-algorithm constructs a tree of bounding boxes around the obstacles in the environment by including all polygons in the root-box and then recursively dividing the polygons into smaller and smaller boxes. The orientations of the bounding boxes are determined by doing a principal component analysis on the vertices.

However, in addition to checking for overlap between two stationary objects, the collision checker must also be able to check that a full path between two helicopter configurations does not collide with any obstacles.

Since the helicopter is quite small compared to the environment, a simplifying assumption has been made where the helicopter is regarded as point-object. The actual size of the helicopter is instead added to a safety margin that is placed around the obstacles in the 3D Revinge world model. If the helicopter is regarded as a point-object, it suffices to check if the cubic polynomial describing a flight path intersects an OBB. This can be done by analytically solving the intersection points for the cubic curve and the bounding planes of the OBB.

The 3D Revinge model we are using is large. It covers an area of 800×800 m^2 and consists of roughly 140,000 polygons of which 120,000 polygons represent ground terrain. The remaining 20,000 polygons represent other types of obstacles, mainly buildings and trees.

Since the majority of polygons in the model represent the ground terrain and the majority of missions are not performed very close to the ground, the depth of the OBB-tree for terrain can be limited to 10. Using this cut-off-value, the generation of the OBB-tree takes approximately 200 seconds, while collision-checking against a typical curve-segment of

40 m length takes 22 milliseconds. In the case of take-off and landing which obviously takes place near terrain, visual navigation and GPS techniques are used for maneuvering close to the ground.

V. The Mission Planning procedure

The path planning module is used in the following manner during missions. At any one time during flight, a number of task procedures are running which call control and other functionalities as needed. During missions over Revinge when the UAV needs to fly from one point to another, the task procedure in control will call the path planning module with its current position and the position to which it wants to fly. In addition, the task procedure may also provide additional runtime constraints on the path such as minimum or maximum altitude or forbidden flight areas.

The path planning module then attempts to resolve the request by first connecting the start and end position to the roadmap which is stored onboard and then applying the A^* algorithm with an appropriate search policy such as shortest distance. The run-time constraints are resolved by eliminating nodes and edges that violate them during the search.

The resulting piecewise linear curve is then subjected to a series of smoothing steps in order to reduce the jagginess that is often the result of randomized path planning algorithms. All these smoothing steps are done by applying the smoothing operation on a certain segment or waypoint and then checking if the resulting path is still collision-free.

The most important of these are the replacement of the linear segments with cubic space curves that makes it possible to join the segments with a continuous first derivative which is required to fly through the waypoints at any greater speed. In principal C^2-continuity (continuous 2nd-order derivative) could also be achieved but this would make all segments of the curve (if we restrict ourself to cubic functions) interdependent which would make the replacement more complicated and demanding to apply.

Other smoothing operations that are applied are elimination of waypoints which results in fewer and longer path segments, and stepwise alignment of the waypoints to make the path straighter.

VI. Missions

The path planner described in this paper is fully integrated and part of our on-board software architecture. It is possible to use the path planner in a fully autonomous manner or to use it in combination with a ground operator for interactive missions. Although, we focus on flight in Revinge, it is straightforward to add and use 3D models from other regions and take advantage of this technology. In this section, we briefly describe some of the missions that have been flown using the path planner functionality.

The first mission demonstrates the functionality of adding runtime constraints when the helicopter is already in the air. In this mission, the helicopter is asked to fly from the starting point in the upper right of figure 9 down behind a house in the middle of the picture. The plan (white) is displayed to the operator, and after approval the helicopter flies over the house to goal position (logged flight-path shown in black).

The operator might disapprove of the path above the house for various reasons, and can in such cases abort the flight and mark a no-fly-zone covering the building, as in figure 10. The figure also shows the resulting plan under the additional constraint, which can be flown by the helicopter after the operator have accepted it. The no-fly-zone is handled by excluding edges that violate this constraint during the graph search.

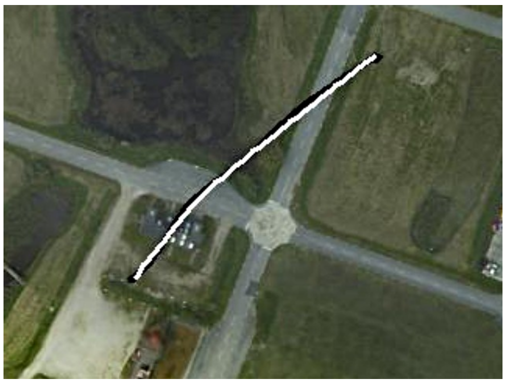


Fig. 9. Autonomous flight between two points over building. Plan is shown in white overlapping the black flight-path plotted from log-data.

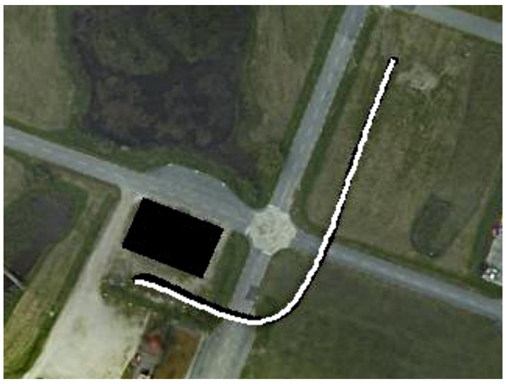


Fig. 10. Same as figure 9 but with no-fly-zone (black rectangle).

The second mission involves starting at home base and flying to two buildings in Revinge and photographing each of its facades. For any given building structure in Revinge, the on-board GIS has functionality to generate suitable camera positions to fly to for photographing each facade. In figure 11, the path generated by the path planning module (white) is shown together with the actual plotted log-data from the real flight (black).

VII. Conclusions

In this paper, we described the use of the probabilistic roadmap path planning paradigm for an unmanned aerial vehicle application. During the development and experimental use of our PRM path planning prototype, a number of limitations to the standard PRM approach were identified.

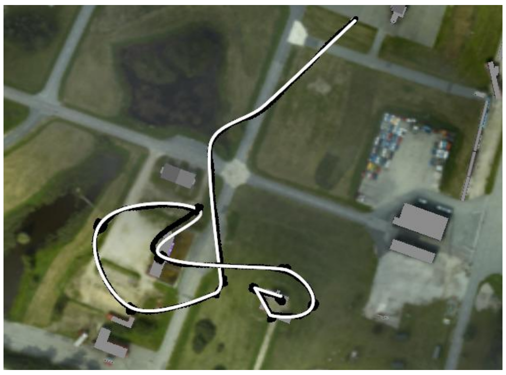


Fig. 11. Autonomous survey mission to building1 and then building2 photographed from each side.

One of the most important issues that arose relates to how one deals with nonholonomic constraints which arise in the interaction between the path planner and the low-level flight controller. In this particular case, the most problematic aspect has been to respect the nonholonomic constraints required to achieve smooth transitions between different trajectory segments, while retaining an efficient planning procedure. We have observed that it is useful to postpone the nonholonomic constraints to the run-time stage in order to reduce the dimensionality of graph generation during the off-line stage.

Another important issue that arose is in regard to the efficiency tradeoff between what is done off-line and what is done on-line. It appears that flexibility as regards adjustment to contingent changes in the UAV environment during plan execution time is traded off against efficient runtime planning based on a PRM generated offline. Since many of our mission scenarios would involve dealing with runtime contingencies such as new no-fly zones or additional visibility constraints, we have moved towards a lazy PRM philosophy, while still retaining the advantages of preprocessing where that is possible.

Ideally, one would like a flexible and uniform means of adding mission constraints both during the off-line and on-line stage. The extent to which this is done may very well be mission dependent and based on the preferences of ground operators. The interaction between constraints that are known in advance and run-time constraints and how both can be incorporated in a planner is an issue that we are currently pursuing.

VIII. Acknowledgments

This work is funded in part by grants from the Wallenberg Foundation and COMPAS NFFP nr-539.

References

[1] G. Conte, S. Duranti, and T. Merz. Dynamic 3D path following for an autonomous helicopter. In *Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

[2] P. Doherty, G. Granlund, K. Kuchcinski, K. Nordberg, E. Sandewall, E. Skarman, and J. Wiklund. The WITAS unmanned aerial vehicle project. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 747–755, 2000.

[3] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. Submitted 2004.

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

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

[6] T. Merz. Building a system for autonomous aerial robotics research. In *Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

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

[8] S. Sekhavat, J-P. Laumond P. Svestka, and M. H. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *The int'l journal of robotics research*, 17:840–857, 1996.