Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle

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

Abstract

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

Introduction

The use of Unmanned Aerial Vehicles (UAVs) which can operate autonomously in dynamic and complex operational environments is becoming increasingly more common. While the application domains in which they are currently used are still predominantly military in nature, in the future we can expect widespread usage in the civil and commercial sectors. In order to insert such vehicles into commercial airspace, it is inherently important that these vehicles can generate collision-free motion plans and also be able to modify such plans during their execution in order to deal with contingencies which arise during the course of operation. Motion planners capable of dynamic replanning will be an essential functionality in any high-level autonomous UAV system. The motion planning problem, that of generating a collision-free path from an initial to a goal waypoint, is inherently intractable for vehicles with many degrees of freedom. Recently, a number of sample-based motion planning techniques (Kavraki *et al.* 1996; Kuffner & LaValle 2000) have been proposed which trade off completeness in the planning algorithm for tractability and efficiency in most cases.

The purpose of this paper is to show how one can incorporate dynamic replanning in such motion planners on a deployed and fully operational UAV by integrating the motion planner with the control kernel of the UAV in a novel manner with little modification of the original algorithms. Integrating both high- and low-end functionality seamlessly in autonomous architectures is currently one of the major open problems in robotics research. UAV platforms offer an especially difficult challenge in comparison with ground robotic systems due to the often tight time constraints present in the plan generation, execution and replanning stages in many complex mission scenarios. It is the intent of this paper to show how one can leverage sample-based motion planning techniques in this respect, first by describing how such integration would be done and then empirically testing the results in a fully deployed system.

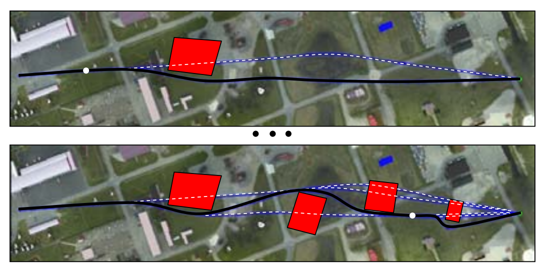

Figure 1: Paths generated during the experimental flight. Solid black line - updated path (white dot - helicopter position); white dashed line - invalid path; polygon box - forbidden region. View from top to bottom.

An example of a dynamic path replanning experiment is shown in Fig. 1. It shows sample paths generated during a flight in which four no-fly zones were added incrementally. The plan was continuously monitored and repaired as new no-fly zones were added through a ground operator interface.

The techniques and solutions described are generic in nature and suitable for platforms other than the one used in this experimentation. An important point to note is that to our knowledge we are the first to use these sample-based motion planning techniques with fully deployed UAVs. The experiments were conducted using the WITAS [1] UAV system (Doherty *et al.* 2004) shown in Fig. 2.

[1] WITAS is an acronym for the Wallenberg Information Tech-

Figure 2: The WITAS UAV platform.

The Path Planning Algorithms

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

The standard probabilistic roadmap (PRM) algorithm (Kavraki *et al.* 1996) works in two phases, one off-line and the other on-line. In the off-line phase a roadmap is generated using a 3D world model. Configurations are randomly generated and checked for collisions with the model. A local path planner is then used to connect collision-free configurations taking into account kinematic and dynamic constraints of the helicopter. Paths between two configurations are also checked for collisions. In the on-line or querying phase, initial and goal configurations are provided and an attempt is made to connect each configuration to the previously generated roadmap using the local path planner. A graph search algorithm such as A^* is then used to find a path from the initial to the goal configuration in the augmented roadmap. The PRM path planner implemented in the WITAS UAV system uses an OBBTree-algorithmfor collision checking and an A^* algorithm for graph search. Here one can optimize for shortest path, minimal fuel usage, etc. The following extensions have been made with respect to the standard version of the PRM algorithm in order to adapt the approach to our UAV platform.

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

The use of rapidly exploring random trees (RRT) provides an efficient motion planning algorithm that constructs a roadmap online rather than offline. The algorithm (Kuffner & LaValle 2000) generates two trees rooted in the start and end configurations by exploring the configuration space randomly in both directions. While the trees are being generated, an attempt is made at specific intervals to connect them to create one roadmap. After the roadmap is created, the remaining steps in the algorithm are the same as with PRMs.

In the current implementation the mean planning time for both planners is below 1000 ms and the use of runtime constraints do not noticeably influence the mean. In the case of RRT planner the success rate is much lower than in PRM and generated plans are not optimal which may sometimes cause anomalous detours (Pettersson 2006).

The Path Execution Mechanism

The standard path execution scheme in our architecture for static operational environments is depicted in Fig. 3. A UAV

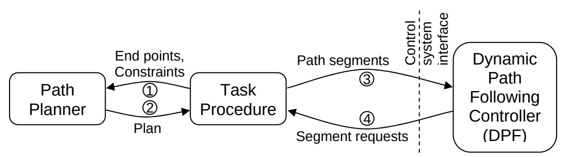

Figure 3: Plan execution scheme

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. For the purposes of this paper, it can be viewed as an augmented state machine.

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

If successful, the path planner (step 2) generates a segmented cubic curve. Each segment is defined by start and end points, start and end directions, target velocity and end velocity. The TP sends the first segment (step 3) of the trajectory via the control system interface and waits for the *Request Segment* event that is generated by the controller.

At the control level, the path is executed using a Dynamic Path Following (DPF) controller (Conte, Duranti, & Merz 2004) which is a reference controller that can follow cubic splines. When a *Request Segment* event arrives (step 4) the TP sends the next segment. This procedure is repeated (step 3-4) until the last segment is sent. However, because the high-level system is not implemented in hard real-time it

---

nology and Autonomous Systems Lab which hosted a long term UAV research project (1997-2004).

This research is partially founded by the Wallenberg Foundation under the WITAS Project and an NFFP4-S4203 grant.

may happen that the next segment does not arrive at the control kernel on time. In this case, the controller has a timeout limit after which it goes into safety braking mode in order to stop and hover at the end of the current segment. The timeout is determined by a velocity profile and current position.

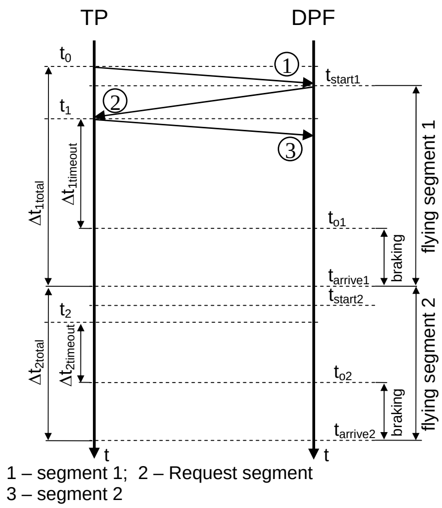

Figure 4: Execution timeline for trajectory consisting of 2 segments

Fig. 4 depicts a timeline plot of the execution of a trajectory (2 segments). At time t_0, a TP sends the first segment of the path to the DPF controller and waits for a *Request segment* event which arrives immediately (t_1) after the helicopter starts to fly (t_{start1}). Typical time values for receiving a *Request segment* event ($t_1 - t_0$) are well below $200ms$. Time t_{o1} is the timeout for the first segment which means that the TP has a $\Delta_{t_1 timeout}$ time window to send the next segment to the DPF controller before it initiates a safety braking procedure. If the segment is sent after t_{o1}, the helicopter will start braking. In the current implementation, segments are not allowed to be sent after a timeout. This will be changed in a future implementation. In practice, the $\Delta_{t_1 timeout}$ time window is large enough to replan the path using the standard path planner. The updated segments are then sent to the DPF controller transparently.

Dynamic Replanning of the Path

There are several services that are used during the path replanning stage. They are called when changes in the environment are detected and an update event is generated in the system. The augmented state machine associated with the TP used for the dynamic replanning of a path is depicted in Fig. 5. The TP takes a start and an end point and a target velocity as input. The TP then calls a path planning service (*Plan* state) which returns an initial path.

If the helicopter is not aligned with the direction of the flight, a command to align is sent to the controller (*Align* state).The TP then sends the first segment of the generated path to the DPF controller (*Send segment* state) and calls the Prediction service to estimate a timeout for the current segment (*Estimate timeout* state). Based on the segment timeout and system latency, a condition is calculated for sending the next segment. If there is no change in the environment the TP waits (*Wait* state) until a timeout condition is true and then sends the next segment to the DPF controller. In case

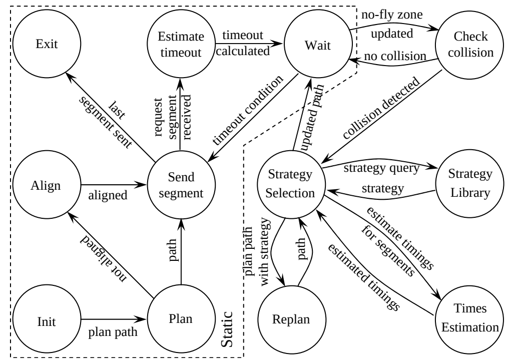

Figure 5: The dynamic path replanning automaton

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

All time estimations that have to do with paths or parts of paths are handled by the Prediction service. It uses the velocity profile of a vehicle and path parameters to calculate timeouts, total times, and combinations of those. For instance, in the case of flying a two-segment trajectory (see execution timeline in Fig. 4) 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).

There are many strategies that can be used during replanning step which can give different results depending on the situation. The Strategy Library stores different replanning strategies including information about the replanning algorithm to be used, the estimated execution time and the priority. Example strategies are shown in Fig. 6.

Strategy 1: Replanning is done from the next waypoint (start point of the next segment) to the final end point. This implies longer planning times and eventual replacement of collision-free segments that could be reused. The distance to the obstacle in this case is usually large so the generated path should be smoother and can possibly result in a shorter flight time.

Strategy 2: Segments up to the colliding one are left intact and replanning is done from the last collision-free waypoint to the final end point. In this case, planning times are cut down and some parts

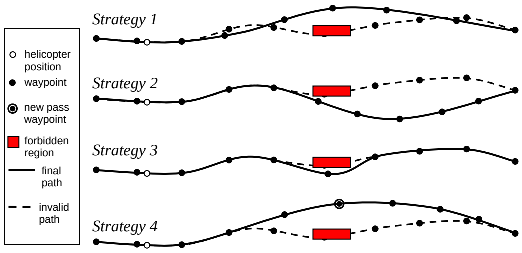

Figure 6: Examples of replanning strategies.

of the old plan will be reused. But since the distance to the obstacle is shorter than in the previous case, it might be necessary for the vehicle to slow down at the joint point of two plans, this can result in a longer flight time.

Strategy 3: Replanning is done only for colliding segments. The helicopter will stay as close to the initial path as possible.

Strategy 4: There can be many other strategies that take into account additional information that can make the result of the replanning better from a global perspective. An example is a strategy that allows new pass waypoints that should be included in the repaired plan.

Note that each of these strategies progressively re-uses more of the plan that was originally generated, thus cutting down on planning times but maybe producing less optimal plans.

The 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 a path or parts of a path. Based on that and 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.

Experimental Results

In our experiments we have used both the PRM and the RRT planner. We included the first three strategies from Fig. 6 in the Strategy Library. During the flight, forbidden regions were randomly added by a 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 Table 1. The number of segments is taken from the final path.

Observe that in the case of *Strategy 1*, Δ_t (time window for replanning) 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*. 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 many more segments (up to 15). Such paths usually imply a smaller average velocity which can result in a longer flight time.

Table 1: Results of the experiments while using two strategies. Values presented in the table are the avarage from many experiments. [1]-*Strategy 1*,[2]-*Strategy 3*

Planner	path length (m)	number of segments	added forbidden regions	min. segment length(m)	max. replannig time (ms)	min. Δ_t (ms)
PRM[1]	464.5	6.6	4.9	48.6	579.9	3229.0
PRM[2]	564.2	12.0	4.7	26.2	174.6	2336.7
RRT[1]	477.2	6.4	4.9	43.2	512.9	3379.4
RRT[2]	558.8	12.3	4.7	19.4	178.4	2128.6

Conclusions

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

References

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

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

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

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

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