Choosing Replanning Strategies for Unmanned Aircraft Systems *

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

Abstract

Unmanned aircraft systems use a variety of techniques to plan collision-free flight paths given a map of obstacles and no-fly zones. However, maps are not perfect and obstacles may change over time or be detected during flight, which may invalidate paths that the aircraft is already following. Thus, dynamic in-flight replanning is required.

Numerous strategies can be used for replanning, where the time requirements and the plan quality associated with each strategy depend on the environment around the original flight path. In this paper, we investigate the use of machine learning techniques, in particular support vector machines, to choose the best possible replanning strategy depending on the amount of time available. The system has been implemented, integrated and tested in hardware-in-the-loop simulation with a Yamaha RMAX helicopter platform.

1. Introduction

Unmanned aircraft systems (UAS) are used for a wide variety of applications in areas such as reconnaissance, surveillance, power line inspection and support for emergency services in natural catastrophes. 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 motion planning and navigation, given a set of waypoints to visit or fly through and a map of static and dynamic obstacles.

A number of methods for generating motion plans have been proposed in the literature, including probabilistic roadmaps (PRM, Kavraki et al. 1996) and rapidly exploring random trees (RRT, Kuffner and LaValle 2000). Unfortunately, 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. There can then 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 (Wzorek et al. 2006).

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

*This work is partially supported by grants from the Swedish Research Council (VR), the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII, the Swedish Research Council (VR) Linnaeus Center CADICS, the CENIIT Center for Industrial Information Technology, the ELLIIT network for Information and Communication Technology, and LinkLab.

Figure 1: The Yamaha RMAX helicopter with integrated laser range finder mounted on a rotation mechanism.

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.

In this paper, we choose the second option: We investigate the use of machine learning (specifically, support vector machines) to choose the repair strategy and the planning parameters that yield the highest expected plan quality given the time available for replanning. A set of prediction models is built offline for each map. The models are used in flight to predict the time requirements for each repair strategy as well as the resulting plan quality, allowing the system to choose the best strategy given the time available. Reactive obstacle avoidance is still applied if the motion planner should fail to deliver a result in time.

2. Helicopter Platform

Diverse research with unmanned aircraft systems is conducted in the UASTech Lab[1], where one of our main platforms (Doherty et al. 2004) is a Yamaha RMAX helicopter (Figure 1). 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 and an attitude control system.

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 contains three embedded PC104 computers connected by serial lines for hard real-time networking as well as Ethernet for CORBA applications, remote login and file transfer. The primary flight control system runs on a 700 MHz Pentium III, and includes a wireless Ethernet bridge, a GPS receiver, and several additional sensors. The image processing system runs on another 700 MHz PIII, and includes color CCD and thermal cameras mounted on a pan/tilt unit, a video transmitter and a MiniDV recorder. The deliberative/reactive system runs on a 1.4 GHz Pentium-M and executes all high-level autonomous functionality.

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

A hybrid deliberative/reactive software architecture has also been developed for our UAS platforms. Conceptually, this is a concentrically layered system with deliberative, reactive and control components. It can be divided into two parts with respect to timing requirements: (a) hard real-time, dealing mostly with hardware and control laws, and (b) non-real-time, which includes most deliberative services. All three computers have both hard and non real-time components. More details about the software architecture in the context of navigation can be found in Wzorek et al. (2006).

3. Motion Planning Algorithms

Motion planning for helicopters typically takes place in a high-dimensional configuration space involving spatial coordinates as well as other properties such as the yaw angle (the direction in which the aircraft is pointing). Though finding optimal paths between two configurations in such spaces is intractable in general, sample-based approaches often provide good solutions in practice by sacrificing optimality and deterministic completeness. Our framework currently includes two such algorithms, probabilistic roadmaps (PRM) (Kavraki et al. 1996) and rapidly exploring random trees (RRT) (Kuffner and LaValle 2000), and can easily be extended.

The PRM algorithm pre-processes a 3D world model to generate a discrete roadmap graph. Configurations in free space are randomly created, and a local planner tests whether pairs of configurations can be connected by flight paths, taking aircraft-specific kinematic and dynamic constraints into account. In the online phase, the planner connects the given initial and goal configurations ("locations") to configurations in the roadmap. Graph search algorithms such as A* can then be used to find suitable paths.

The UASTech implementation of this algorithm has been extended to handle constraint addition at runtime (Pettersson 2006). Currently supported dynamic constraints include forbidden regions (no-fly zones) and bounds on maximum and minimum altitude and the rate of ascent or descent.

The RRT algorithm constructs a roadmap online rather than offline. Two trees are generated by exploring the configuration space randomly from the start and end configurations. At specific intervals, an attempt is made to connect the trees. Compared to PRM, the success rate for RRT is noticeably lower and the plan quality tends to be lower, with a higher probability of anomalous detours (Pettersson 2006). However, since RRT planners do not require extensive preprocessing, they can be used in situations where PRM planners are inapplicable.

A path generated by our planners consists of a set of segmented cubic polynomial curves. Each curve is defined by start and end points, start and end directions, intended velocity, and intended end velocity. At the control level, the path is executed using a Dynamic Path Following (DPF) controller (Conte, Duranti, and Merz 2004), a reference controller following cubic splines.

Though paths should be collision-free, new no-fly zones can be added by a ground operator during execution, and unknown buildings can be detected by proximity sensors. In case such new obstacles intersect any segment in the current

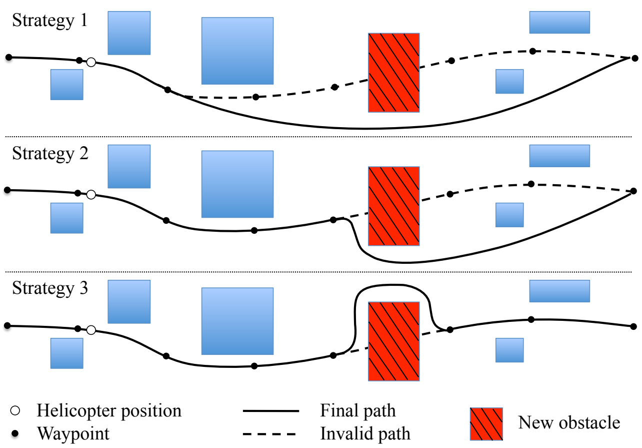

Figure 2: Example replanning strategies.

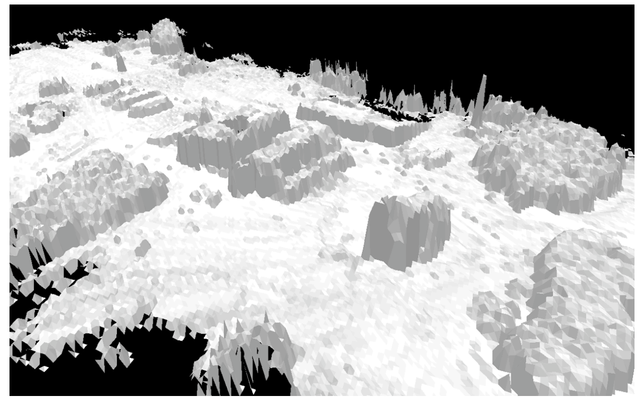

Figure 3: Reconstructed elevation map for a flight test area based on real flight data from the laser range finder.

path, a UAS has a certain time window when the current path can be updated to smoothly avoid new obstacles. Even for a fixed choice of planner (PRM, RRT, or another alternative) and parameters, one can still choose between multiple strategies when determining which parts of the path should be modified. The following are three examples, also illustrated in Figure 2 for a short path consisting of 7 segments. Note that each strategy progressively re-uses more of the plan that was originally generated, thus cutting down on planning times but potentially producing less optimal plans.

Strategy 1 – Full replanning. The path is replanned from the next waypoint to the final end point. Since this gives the planner maximum freedom, it provides the best opportunities for generating a smooth path with a short flight time. However, as the trajectory to be created may be long, more time may be required for planning.

Strategy 2 – Partial replanning. Replanning starts with the waypoint immediately preceding the new obstacle, leaving all previous segments intact. This gives the planner the maximum amount of time to generate a new plan. Since the aircraft waits longer before diverging from the original path, the final plan may be longer and may contain sharper turn angles where the aircraft has to slow down. Due to the partly random nature of each planner, it is also possible for strategy 2 to yield a shorter plan, contrary to expectations.

Strategy 3 – Plan repair. Replanning is done only for colliding segments. This strategy tends to be the fastest, as only a small part of the original path has to be altered, and is suitable when the aircraft is already close to the obstacle and very little time is available for replanning. However, it also tends to lead to the lowest plan quality and may involve sharp turns that force the aircraft to slow down.

4. Strategy Selection using Machine Learning

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, Shim, Chung, and Sastry 2006) 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* should determine which replanning strategy can be expected to yield the best plan within the available time.

The amount of time we have available depends on several factors. The most obvious ones may be the range at which the new obstacle was detected and our current velocity. Given these factors, we can calculate the time remaining before we reach the obstacle. 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 we 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 our current velocity gives us the time window in which we can replan.

We use the popular Sick LMS-291 laser range finder, mounted on a rotation mechanism developed in-house, for obstacle detection (Figure 1). This allows one to obtain half-sphere 3D point clouds even when a vehicle is stationary. An example scene reconstruction from real flight test data is shown in Figure 3.

The laser has a maximum detection range of 80 meters. Based on experiments, 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 (± 1 meter). Thus, we will know the distance to an obstacle as soon as it is detected.

The time required to fly any given part of a path can be calculated using a mathematical model of the dynamic path following controller together with the intended velocity for

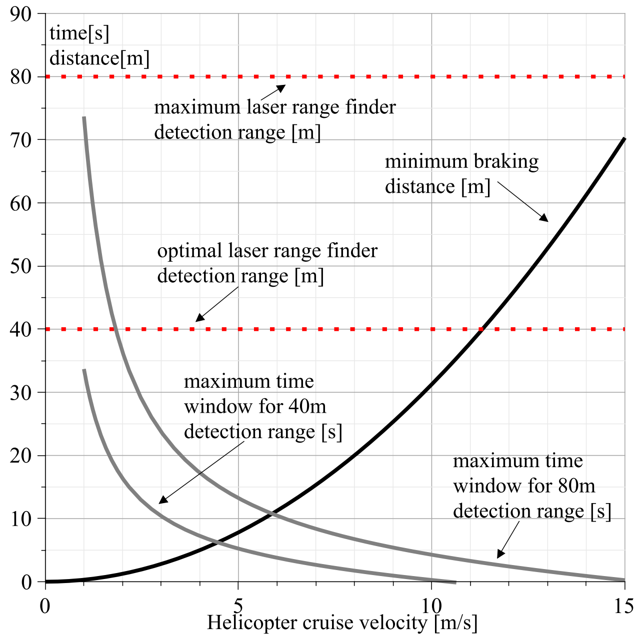

Figure 4: The minimal braking distance and time windows for the Yamaha RMAX as a function of the cruise velocity.

each segment of the path. We can also calculate the distance required for an emergency break at each possible velocity, shown as a black solid line in Figure 4. The figure also depicts the resulting time windows available for decision making for the Yamaha RMAX helicopter 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, assume we fly at only 10 m/s and detect an obstacle at a range of 80 meters. Then, the braking distance is 31.5 meters, and we have $80 - 31.5 - 6 = 42.5$ meters available for replanning, which corresponds to 4.25 seconds. If the obstacle is detected at 40 meters, we have only 0.25 seconds available.

For most strategies, calculating the expected time requirements for replanning is considerably less straightforward than the time window estimation above. 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 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.

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. We assume a stationary distribution, where the relevant properties of the map do not change over time. If signficant changes are detected, for example because large numbers of new obstacles have been detected, the prediction model can be recomputed for use in future

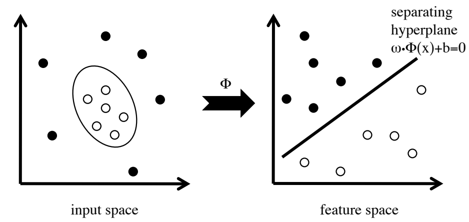

Figure 5: Mapping and separating hyperplane.

missions.

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

4.1 Support Vector Machines

Several machine learning techniques were tested and compared, including support vector machines (SVM) (Vapnik 1998; 1999), least median squared linear regression, gaussian processes for regression, isotonic regression, and normalized Gaussian radial basis function networks.

With their high generalization performance and 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 (Vapnik 1999), face detection and recognition (Li 2004) and vehicle detection (Sun 2005). As it turns out, SVMs also yielded the smallest prediction errors for our domain.

The idea underlying (non-linear) support vector machines is that n-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). 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. We have used the Pearson VII Universal Kernel (PUK) (Üstün, Melssen, and Buydens 2006):

$$K(x_i, x_j) = \frac{1}{\left[1 + \left(2\sqrt{\| x_i - x_j \|^2}\sqrt{2^{1/\omega} - 1}\ /\sigma\right)^2\right]^\omega}$$

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) (Smola 2004; Shevade 2000) to solve the regression problem, which has minimal computational requirements (Witten and Frank 2005).

4.2 Prediction features

Many parameters influencing planning time and plan quality were considered as potential inputs to the SVR 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 6).
- 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.

Correlation-based feature selection (Hall 2000) was used to assess the relevance of these features relative to the six replanning strategies 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.

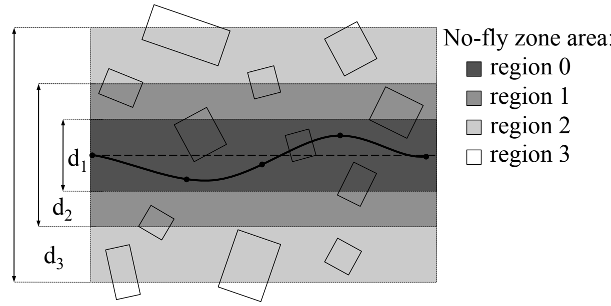

Figure 6: No-fly zone area calculation.

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 6 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.

4.3 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.

We began the learning process using the PRM 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 the laser range finder. 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 2 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

Environ-ment	Strategy	Nodes	Model Evaluation Results: Replanning time prediction [%]	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 1: Relative mean error of prediction and standard deviation for the PRM planner.

cases that were generated for the evaluation. The performance of the machine-learned models is similar in the cases where these assumptions are not present.

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 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

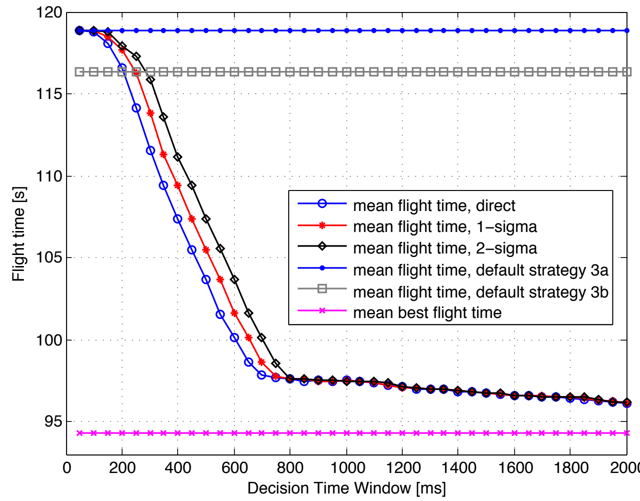

Figure 7: Plan quality (flight time) as a function of the available decision time window for environment 1.

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 7 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 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.

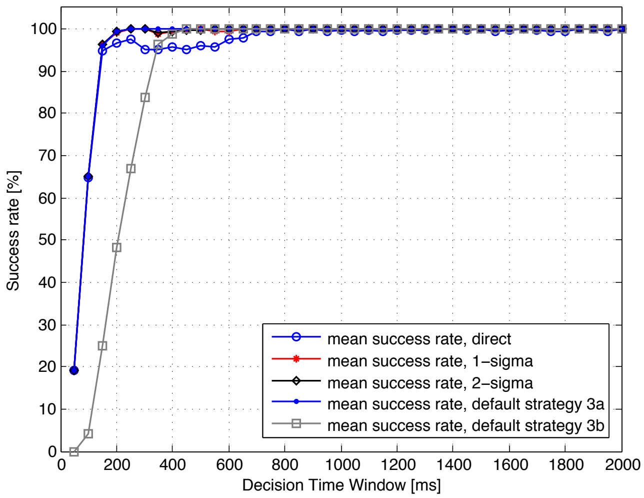

Figure 8: Success rate of execution of the chosen strategy in the available decision time window for environment 1.

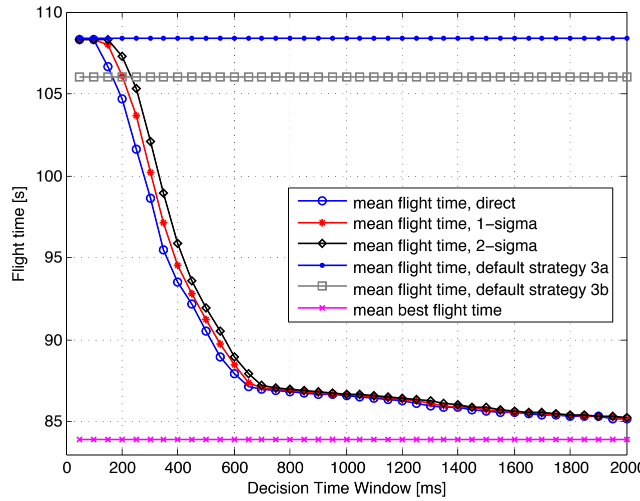

Figure 9: Plan quality (flight time) as a function of the available decision time window for environment 2.

Success rates. Figure 8 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 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 9 and 10 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.

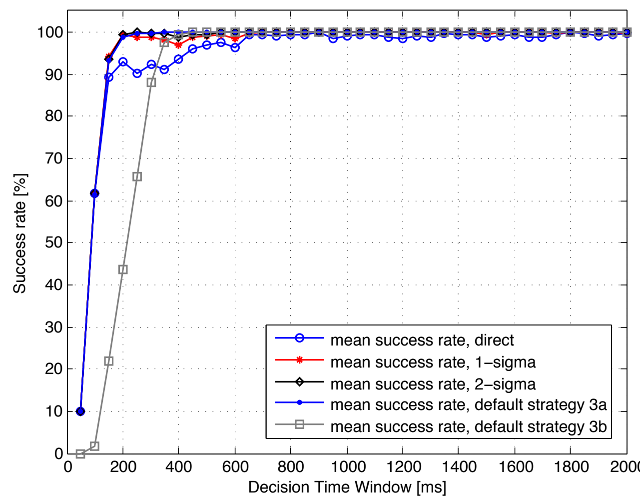

Figure 10: Success rate of execution of the chosen strategy in the available decision time window for environment 2.

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.

5. Related work

In the framework proposed by Morales et al. (2004), 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 in Rodriguez et al. (2006), where the strategies used by the RESAMPL motion planner are guided by the entropy of each region.

Burns (2005) proposes 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 a given time window, which is required for our problem.

Hrabar (2006) presents a UAS 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 (Stentz 1995). 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 presented experiments use a flight velocity of 0.5 m/s.

A number of motion replanning algorithms have been proposed in the literature, including DRRT (Ferguson, Kalra, and Stentz 2006) and ADRRT (Ferguson and Stentz 2007). 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.

6. Conclusions

When an unmanned aircraft detects an obstacle ahead, there is generally little time available for replanning. Reactive behaviors for collision avoidance suffer from problems with local minima. Additionally, they solve the problem locally without a global overview of the environment, which may lead to highly suboptimal paths. Instead, we can identify a number of replanning strategies that vary in time requirements as well as in the resulting plan quality. Much can then be gained by choosing the highest quality strategy that does not exceed the given time window. We have applied machine learning techniques to this problem, with promising results in empirical testing: 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.

References

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

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

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

Ferguson, D., and Stentz, A. T. 2007. Anytime, dynamic planning in high-dimensional search spaces. In *Proc. ICRA*.

Ferguson, D.; Kalra, N.; and Stentz, A. T. 2006. Replanning with RRTs. In *Proc. ICRA*.

Hall, M. 2000. Correlation-based feature selection for discrete and numeric class machine learning. In *Proc. ICML*.

Hrabar, S. 2006. *Vision-Based 3D Navigation for an Autonomous Helicopter*. Ph.D. Dissertation, University of S. California.

Kavraki, L. E.; Švestka, P.; Latombe, J.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *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. ICRA*.

Li, Y. 2004. Support vector machine based multi-view face detection and recognition. *Image and Vision Computing* 22(5):413–427.

Morales, M.; Tapia, L.; Pearce, R.; Rodriguez, S.; and Amato, N. M. 2004. A machine learning approach for feature-sensitive motion planning. In *Proc. Int. Workshop on the Algorithmic Foundations of Robotics*.

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

Rodriguez, S.; Thomas, S.; Pearce, R.; and Amato, N. M. 2006. RESAMPL: A region-sensitive adaptive motion planner. In *Proc. Int. Workshop on the Algorithmic Foundations of Robotics*.

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

Shim, D.; Chung, H.; and Sastry, S. 2006. Conflict-free navigation in unknown urban environments. *Robotics & Automation Magazine, IEEE;* 13(3):27–33.

Smola, A. 2004. A tutorial on support vector regression. *Statistics and Computing* 14(3):199.

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

Sun, Z. 2005. On-road vehicle detection using evolutionary Gabor filter optimization. *IEEE Transactions on Intelligent Transportation Systems* 6(2):125–137.

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

Vapnik, V. 1998. *Statistical Learning Theory*. Wiley-Interscience.

Vapnik, V. 1999. *The Nature of Statistical Learning Theory*. Springer.

Witten, I., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition.

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