A Probabilistic B-Spline Motion Planning Algorithm for Unmanned Helicopters Flying in Dense 3D Environments

Conference Paper · October 2008

DOI: 10.1109/IROS.2008.4651122 · Source: IEEE Xplore

2 authors:

Emre Koyuncu
Istanbul Technical University
88 PUBLICATIONS **480** CITATIONS

SEE PROFILE

Gokhan Inalhan
Cranfield University
186 PUBLICATIONS **3,272** CITATIONS

SEE PROFILE

A Probabilistic B-Spline Motion Planning Algorithm for Unmanned Helicopters Flying in Dense 3D Environments

Emre Koyuncu, Gokhan Inalhan, *Member, IEEE*

***Abstract*—This paper presents a strategy for improving motion planning of an unmanned helicopter flying in a dense and complex city-like environment. Although Sampling Based Motion planning algorithms have shown success in many robotic problems, problems that exhibit "narrow passage" properties involving kinodynamic planning of high dimensional vehicles like aerial vehicles still present computational challenges. In this work, to solve the kinodynamic motion planning problem of an unmanned helicopter, we suggest a two step planner. In the first step, the planner explores the environment through a randomized reachability tree search using an approximate line segment model. The resulting connecting path is converted into flight way points through a line-of-sight segmentation. In the second step, every consecutive way points are connected with B-Spline curves and these curves are repaired probabilistically to obtain a dynamically feasible path. Numerical simulations in 3D indicate the ability of the method to provide real-time solutions in dense and complex environments.**

I. Introduction

Although many kinodynamic motion planning methods have been developed, they rarely can be used in practice because of their computational complexities. Especially, for the aerial vehicles that we focus on, it is hard and time consuming to obtain a feasible path using standard kinodynamic planners.

It is noted in [1] that general kinodynamic motion planners require at least exponential time in that dimension of the state space of dynamical systems which is usually at least twice the dimension of the underlying configuration space. In practice, kinodynamic planners are implementable only for systems that have small state-space dimensions. For example, the work presented in [1] suggests a path-planning relaxation which defines a class of maneuvers from a finite state machine, and uses a trajectory based controller to regulate the unmanned vehicle dynamics into these feasible trajectories. However, the trajectories to be controlled are limited to the trajectories generated by the finite state machine and the computational challenges of generating real-time implementable flight trajectories in 3D complex environments still remains as a challenge.

Manuscript received February 22, 2008. This work was supported in part by DPT HAGU grant administered by ITU ROTAM.

E. Koyuncu is with the Controls and Avionics Laboratory, Istanbul Technical University, Istanbul, Turkey (phone: 90-212-285-3148; e-mail: emre.koyuncu@itu.edu.tr).

G. Inalhan is with the Faculty of Aeronautics and Astronautics, Istanbul Technical University, Istanbul, Turkey (e-mail: inalhan@itu.edu.tr).

In this work, we suggest a real-time implementable two-step planner strategy in which the complex 3D environment and the passages are rapidly explored using an RRT planner. From this feasible but not necessarily direct probabilistic path, line-of-sight critical milestones are extracted [18]. An extended discussion and illustration of the algorithm can be found in [18]. Although these milestones allow point-to-point flyable flight path segmentation, it does not necessarily correspond to a fast agile and continuous motion plan. To address this, as a second step, B-Spline method is used for generating constant acceleration flight paths that pass through these milestones. These time-scalable constant acceleration flight paths approximate unmanned helicopter closed-loop dynamics under trajectory control. In face of collisions, the milestones locations are probabilistically adjusted to eliminate the collisions and the accelerations are limited in time scale to allow dynamic achievability. Screen capture of implementation for city-like path planning problem is illustrated in Fig. 1 that is taken from ITU CAL flight simulator.

Fig. 1. Motion planning in dense and complex environments: Screen capture from the ITU CAL flight simulator

For developing a real-time implementable planner, motion planning researches have been focused on sampling based approaches that rapidly search either the configuration or the state space of the vehicle. In the last few decades, sampling-based motion planning algorithms have shown success in solving challenging motion planning problems in complex geometries while using a much simpler underlying dynamic model in comparison to an air vehicle. Roadmap-based planners, like well-known Probabilistic Road Mapping (PRM) method as mentioned in [2], are typically used as multi-query planners (i.e. simultaneous search of the environment from different points) that connect these multiple queries using a local planning algorithm. PRM planners converge quickly toward a solution path, if one

exists, as the number of milestones increases. This convergence is much slower when the paths must go through narrow passages. For complex environments, some extended algorithms are suggested for PRM like planners in [3] and [4]. Tree-based planners build a new roadmap for each query and the newly produced samples are connected to the samples that are already exists in the tree as in [5], [6], [7], [8] and [9]. Rapidly-Exploring Random Tree (RRT) is the most popular representative of tree-based planners that is an exploration algorithm for quickly searching high-dimensional spaces that have both global and differential constraints. Sampling-based planners, especially tree-based planners (RRT and single-query PRM variants), have been adapted to solve dynamically feasible paths that accommodate kinodynamic constraints. Kinodynamic planning refers to problems in which the motion must satisfy nonholonomic and/or dynamic constraints [10]. The main philosophy behind kinodynamic planning is searching a higher dimensional state space that captures the dynamics of the system [11], [10]. Also note the important experimental results of an RRT based path planning algorithm for micro air vehicles in [12].

"Gradual motion planning" methods, that have resemblances to our strategy, are recently proposed to solve complex path planning in cluttered environments. These methods first solve a relaxed form of the problem and then the approximate solution is refined to solve the original problem with a "repairing" method. In [13], a roadmap is initially generated by allowing some penetration into the collision workspace. Later, "milestones" are carried to collision-free space. In Iterative Relaxation of Constraints (IRC) method [14], first a relaxed version of problem is solved and then this coarse solution is used as a guide to solve original problem iteratively. The strategy of using an approximate solution to obtain a collision-free path is also used in Lazy PRM [15], C-PRM [16], and Fuzzy-PRM [17]. In our earlier work [18], using a similar strategy, a Mode Based PRM method is refined with modal flight-path segments to obtain flyable trajectories.

From a deterministic perspective, B-Spline curves have been used in many dynamic path planning and control problem implementations. In [19], dynamic trajectory is generated with the minimum travel time for two-wheeled-driven type mobile robot. In [20] and [21], visibility-based path is modified to continuous feasible path via B-Spline curves. Using the well known local support property of B-Spline curves, real-time path modification methods are proposed for multiple mobile robots in [22] and robot manipulators in [23].

In comparison, our method utilizes both the probabilistic and the deterministic aspect of both of the above approaches to obtain a real-time implementable two-step planner strategy. In the first step, the algorithm rapidly explores the complex environment and the passages using an RRT planner. In this part, our strategy focuses only finding an obstacle-free path that can be tracked from the initial point to the goal point with line segments in the configuration space. This coarse obstacle-free path will be called as *connectivity path*. After finding the *connectivity path,* this path is filtered with the line-of-sight implementation to eliminate the points that cause long detours [18]. Remaining points that we will call as *way points* generally appear in entering and exiting regions of the narrow passages that are formed between the obstacles. An advantage of this refinement is that we can segment the hard path planning problem to a series of small path generating problems on these locations. In the second step of our strategy, for every consecutive *way points,* third-order B-Spline curves are generated deterministically and checked for collision and dynamic feasibility between them. These third-order B-spline curves present constant inertial acceleration paths with breakpoints. If the generated curve is not feasible, probabilistic repairing is achieved by randomized waypoint expansion on the connecting line path and the unit flight time is expanded to limit the accelerations within controllable regime. Since B-Spline curves have local support property, these repairing processes can be made on local path segments of interest. In Section II and Section III, the motion planning algorithm and the numerical results are presented respectively.

II. MOTION PLANNING

Our general motion planning strategy, as a first step, is aimed at finding a *connectivity path* between the initial and the goal point. Since, this phase needs to add minimal computational time; it should be rapid and have the ability to give a quick solution to the connectivity problem. Therefore, in this phase, RRT algorithm is used because of its well quick spreading ability. RRT algorithm can rapidly search the space in a short time because of its well extension property. To decrease the computational time, we disregard the vehicle's dynamic constraints and use RRT for searching only the configuration space of vehicle. After this first phase *connectivity path* is refined with *line-of-sight* transition algorithm to filter long detours. Remaining points turn into *way points* on collision free region. Finally, dynamically feasible paths are searched by the B-spline path planning algorithm. The approach is illustrated summarized in Fig.2.

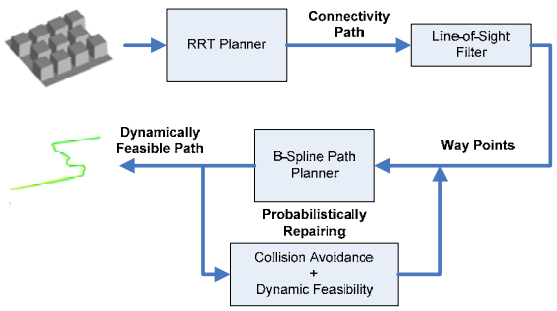

Fig. 2. Motion planning strategy

A. *Finding Connectivity Path and Way Points*

In motion planning problem for complex environments, first major question is whether there is a solution path or not. Planning algorithm in generally try to find solution path in predefined number of iterations. It is hard to say when planners should stop searching or change the searching method (i.e. switching to a more complex planner etc.) or begin repairing algorithms. Especially in real-time applications, planners should be able give a reliable answer in minimal permitted time slot. However, finding an obstacle free geometrical path does not necessarily mean that a dynamically feasible path exists. Although for helicopters these geometrical paths can be implemented via point to point navigation, this flight strategy is not necessarily efficient and agile for operations in under-threat environments. For vehicles that have high dimensional state-space, as the case for a helicopter, searching in high dimensional state-space consumes long computational time to find a feasible path. Specifically, in our earlier work [18], we observed that before the major feasible path planning phase, defining the geometrical obstacle free path and trackable way points accelerates the searching ability and decreases the total computational time of planner.

For finding *connectivity path*, RRT algorithm is used because of its rapid spreading ability. RRT is a considered as being an efficient algorithm to search even high dimensional spaces that has both global and differential constraints [24]. However, one of the important drawbacks of using RRT as a stand-alone planner is biasing of the distribution of milestones towards the obstacle regions if the configuration space has large obstacles. In this phase, we are only motivated by RRT's good property to obtain *connectivity path* that can be tracked during flight. Our strategy does not focus on feasibility in this part of the path planner. Therefore, RRT algorithm is used for searching configuration-space of vehicle with primitive maneuvers that includes level and climbing flight and changing instantaneous heading direction. Construction of *connectivity path* algorithm is given as Goal Biased RRT Algorithm.

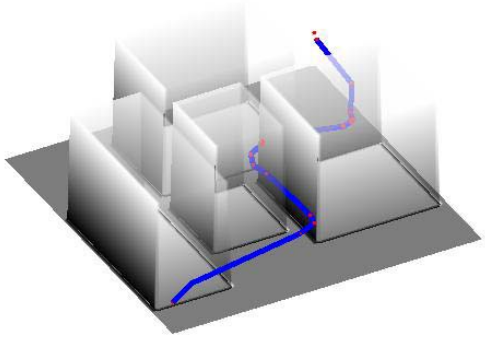

Fig. 3. Construction of connectivity path

In this algorithm as given in Algorithm I, our strategy is Goal Biased RRT-based planner with a single tree that is extended from the initial point. Each loop attempts to extend the tree, first towards the random selected point m_{rand}, and second towards the goal point by adding new points. Specifically, the nearest point already within τ tree to the sampled random point (in Line 4) and the nearest point to the goal point is selected (in Line 11) respectively. *Generate* function generates new point m_{new} on the direction of the selected nearest points m_{near} at random selected distances as shown in Line 5 and 12. If direction angles exceed predefined limits, max direction angles are selected. These boundaries may be chosen according to vehicles kinematic boundaries. If new generated point and its trajectory is in obstacle-free region (checked in Line 7 and 14) then m_{new} is added τ tree as shown in Line 8 and 15. If τ tree reaches *end region*, algorithm returns with success and gives *connectivity path*. *End region* is defined to be within a tolerable capture region as explained in [10]. A solution of the algorithm in a complex city-like environment is illustrated in Fig. 3.

Algorithm I: Goal Biased RRT Algorithm

Input : initial and goal positions; q_{init} , q_{goal}
Output: *connectivity path*
1: $\tau \leftarrow q_{init}$ and $i \leftarrow 1$
2: **repeat**
3: **Select** random point m_{rand} in C
4: **Select** *nearest neighbor* m_{near} of m_{rand} in τ tree
5: **Generate** m_{new} from m_{near} toward m_{rand}
6: **Generate** trajectory e_{new} from m_{near} to m_{new}
7: **if** $e_{new}$is in collision-free region **then**
8: $\tau \leftarrow m_{new}$ tree and $i+1$
9: **if** m_{new} is in *end region* **then**
10: **break** with *success*
11: **Select** nearest neighbor m_{near} of q_{goal} in τ tree
12: **Generate** m_{new} from m_{near} toward q_{goal}
13: **Generate** trajectory e_{new} from m_{near} to m_{new}
14: **if** $e_{new}$is in collision-free region **then**
15: $\tau \leftarrow m_{new}$ tree and $i+1$
16: **if** m_{new} is in *end region* **then**
17: **break** with *success*
18: **if** i= N max iteration number **then**
19: **break** with *fail*
20: **until** *end region* is reached with *success*
21: **Select** *connectivity path* points can be gone back *end region* to initial point in τ tree

Because of the RRT's extending strategy and our simplifications, undesirable detours are frequently seen in the obtained *connectivity path*. Since we only consider finding the obstacle free "open way" in this part; we can simply remove the points that cause these detours. Therefore, we perform a simple *filter algorithm* that finds line-of-sight connections over the *connectivity path*. As can be seen in Fig. 4, the key milestones frequently appear in

while entering and exiting field of narrow passages and hard regions. Hence, these *guard points* also indicate where hard regions are beginning and the direction of the passage.

In this phase of the algorithm, a simple iteration checks whether selected point can be connected with previous points in *connectivity path* with a line segment without colliding with any obstacles. If the line segment collides with an obstacle, in other words, if the current point cannot be connected to the selected point with a line-segment, last connectable point is added to the *way point* sequence and subsequently the search continues from this point on. Search process runs until the last point of *connectivity path* is reached. Pseudo code of this algorithm can be found in [18] and the line-of-sight solution of the connectivity path shown in Fig. 3 is illustrated in Fig. 4.

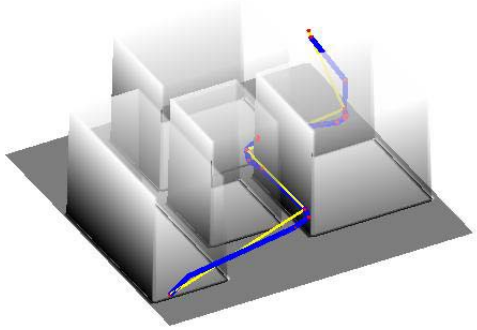

Fig. 4. Refining connectivity path with line-of-sight filter

B. Kinodynamic B-Spline Path Planning Algorithm

Our implementation of the goal-biased RRT algorithm provides a flight path that is represented with *way points*. Filtering the *connectivity path* with straight line segment results in a simple and implementable piecewise flight plan. However, this flight plan is not a fast agile and continuous motion plan – a desirable feature in many urban unmanned helicopter applications. After obtaining the *way points* on the environment, many deterministic and sampling based path planner methods can be used to find the dynamically feasible path between the *way points*. Generated paths must be able connect on the *way points* dynamically. Moreover, the paths should allow reshaping to supply collision avoidance and dynamic feasibility. Therefore, *local support* is also a desirable property on the path generation method. Local support means that the paths only influence a region of the local interest. Thus, obstacle avoidance and dynamic-feasibility repairing can be achieved without changing the whole shape of the generated path.

B-Spline approach can supply these main requirements. An overview of B-Spline can be found in [25]. Basically, output $C(u)$ can be defined in terms of an order k B-Spline curve;

$$C(u) = \sum_{i=0}^{n} P_i N_{i,k}(u) \qquad 0 \le u \le u_{max} \tag{1}$$

The coefficients P_i are called control points. The B-Spline basis functions $N_{i,k}$ are given by the Cox De Boor recursion;

$$N_{i,0}(u) = \begin{cases} 1, & u_i \le u \le u_{i+1} \\ 0, & otherwise \end{cases} \tag{2}$$

$$N_{i,k}(u) = \frac{u-u_i}{u_{i+k-1}-u_i} N_{i,k-1}(u) + \frac{u_{i+k}-u}{u_{i+k}-u_{i+1}} N_{i+1,k-1}(u) \tag{3}$$

A B-Spline curve can be constructed from Bezier curves joined together with a prescribed level of continuity between them. A nondecreasing sequence of real numbers $U = [u_0 \quad \cdots \quad u_{max}]$ is called the knot vector. Frequently, the knot points are referred to as the break points on curve [26]. B-Spline basis function $N_{i,k}$ is zero outside the interval $[u_i, u_{i+k}]$ and non-negative for all values of k, i and u.

Derivatives of B-Spline curve exist on the knot vector span. Since, the kth-order B-spline is actually a degree $(k-1)$ polynomial, produced curve can be differentiated $k-1$ times.

$$C(u)^{(j)} = \sum_{i=0}^{n} P_i N_{i,p}^{(j)}(u) \quad 0 \le u \le u_{max} \tag{4}$$

A valuable property of the B-Spline curves is that the curves can be structured with a specific knot vector form to obtain paths which are tangential to the control polygon (formed by the control points) at the start and end points for each flight segment. This property can be used to define the starting and ending direction of the curve by inserting an extra pseudo control points after the start point and before the end point. These directions are defined according to way points' orientations. Note that the B-Spline curves are generated in between each of these waypoints [27] in a progressive fashion.

In our kinodynamic path planning algorithm, we choose to generate third-order path curves (quadratic polynomials) to obtain constant inertial acceleration paths. This approximates the constant acceleration flight paths that can be tracked via a closed-loop helicopter system. For generating B-Spline path segments between consecutive way points (defined as start and end way points), two pseudo control points are added. The first pseudo control point is added after the start way point. The second pseudo control point is added before the end waypoint. The distance value between the way-point and the corresponding control point defines the end velocity and accelerations of the B-Spline. The first control point is on the heading-direction of the start waypoint. Location of the first inserted pseudo control point (after the initial way point) is selected according to initial velocity condition of vehicle. The distance from the initial way point is compliant with this initial velocity value. This second control point is located at a random selected d distance on the negative heading-direction of the end way

point. Note that, while adding a new pseudo control point for the consecutive spline, the same control point strategy is used. Hence, C^l continuity, in other words, continuous velocity transition is achieved between consecutive B-Splines. We note that the number of control points can be incremented by the algorithm to repair the spline. For generating order-three B-Spline curves, we use specific non-uniform knot vector form $\mathbf{U} = [0 \quad 0 \quad 0 \quad \dots \quad 1 \quad 1 \quad 1]$ to obtain the coincidence between the first and end control points and the beginning and the end points of B-Spline curves respectively. Detailed information about this effect can be found in [25]. We choose using [0,1] interval for parameter u such that it represents unit-time scale [28]. This property is later used to allow velocity and acceleration feasibility via time scaling (i.e. expanding/limiting the time horizon of the maneuver). Overall B-Spline path planning algorithm is shown in Algorithm II.

This algorithm tries to find dynamically feasible B-Spline curve for every consecutive *way points* in a loop which begins in Line 2 and runs until the last *way point* is connected with a feasible path. For every loop, as shown in Line 5, initial way point is selected as beginning control point and goal way point also is selected as end control point. To obtain the initial control polygon, second and third control points are selected as mentioned way before and depicted in Line 6 and 7. Algorithm begins with four control points, thus, $\boldsymbol{P_{new}}$ is null initially. Knot vector as used in our implementation is shown in Line 8. $\boldsymbol{u_{new}}$ vector starts initially having one member for the third-order spline and should be chosen in the unit time[0,1] interval. We choose this initial interior knot value as 0.5. B-Spline curve is evaluated in Line 9 and then the collision and dynamic feasibility checked in Line 10. B-Spline curve is evaluated in Line 9 and then the collision and dynamic feasibility checked in Line 10. Dynamic feasibility check is done by checking the first and the second derivatives of B-Spline curve which gives the velocities and accelerations of the vehicle respectively. If these velocity and acceleration values are within the flight envelope of the vehicle, generated path segment is marked as dynamically feasible.

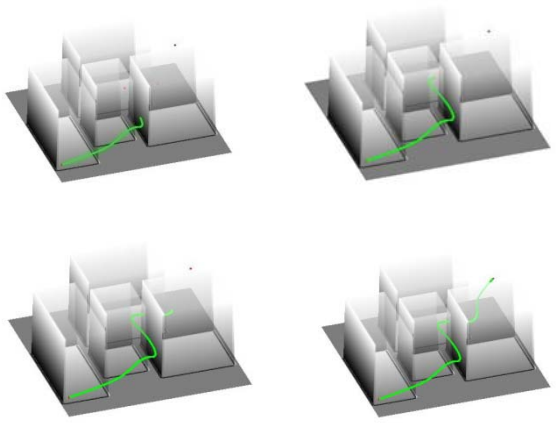

Fig. 5. A typical time history of the path segments with kinodynamic B-Spline path planner

Algorithm II: Path Planning with B-Spline

Input : $\mathbf{g} = [g_1 \quad \cdots \quad g_{last}]$ way points vector
Output: dynamically feasible path
1: $i \leftarrow 1$ and *timescale* $\leftarrow$ *initial timescale*
2: **repeat**
3: $\boldsymbol{P_{new}} \leftarrow \{\}$ and $\boldsymbol{u_{new}} \leftarrow$ *initial* u_{new} initially
4: **repeat**
5: $P_0 \leftarrow g_{ith}$ and $P_m \leftarrow g_{(i+1)th}$ way points
6: **Locate** P_1 according to initial velocity and acceleration of vehicle on positive heading direction of g_{ith}
7: **Select** P_{m-1} that d random distance from P_m on negative heading direction of $g_{(i+1)th}$
8: $\mathbf{P} \leftarrow [P_0 \quad P_1 \quad \boldsymbol{P_{new}} \quad P_{m-1} \quad P_m]$ and $\mathbf{U} \leftarrow [0 \quad 0 \quad 0 \quad \boldsymbol{u_{new}} \quad 1 \quad 1 \quad 1]$
9: **Evaluate** B-Spline with parameter u; 0 to 1
10: **Check** collisions and feasibility
11: **if** collision occurs or spline is not feasible **then**
12: **Change** location P_{m-1} on negative heading direction of $g_{(i+1)th}$ and $m_1 + 1$
13: **if** $m_1 > M_1$ max iteration number **then**
14: **Change** location of $g_{(i+1)th}$ in its small region, $m_2 + 1$ and $m_1 = 0$
15: **if** $m_2 > M_2$ max iteration number **then**
16: $\mathbf{P_{new}} \leftarrow P_{new}$ as new control point around infeasibility, update knot vector and $m_3 + 1$ and $m_1, m_2 = 0$
17: **if** $m_3 > M_3$ max iteration number **then**
18: $i - 1$ and $m_4 + 1$ and $m_1, m_2, m_3 = 0$
19: **if** $m_4 > M_4$ max iteration number **then**
20: **Change** *timescale* to insert min/max values of velocity and acceleration in feasible interval $m_5 + 1$
21: **Check** feasibility of all splines from first
22: **if** ith spline is not feasible **then**
23: $i \leftarrow ith$ and $m_1, m_2, m_3, m_4 = 0$
24: **if** $m_5 > M_5$ max iteration **then**
25: **break** with $fail$
26: **else**
27: $i + 1$
28: **until** path can be evaluated between way points
29: **until** g_{last} last way point is reached

If feasibility cannot be obtained, repairing methods are implemented hierarchically. This is illustrated as the feedback loop in Fig.1.

First, location P_{m-1} pseudo control point is carried randomly on the same tangent direction as shown in Line 12 and the algorithm evaluates the curve again in the main loop. It changes the shape of control polygon, hence, the end velocity of the path and the constant accelerations are changed. Note that, all the repairing steps are tied with predefined threshold iteration times illustrated as $M_i s$ in algorithm. After predefined number of trials, if the spline cannot be repaired, the goal way point is carried to a new collision-free location that is a small random-selected distance away from the prior location. This is shown in Line

14. Still, if the spline cannot be repaired, P_{new} new control point is added in $\mathbf{P_{new}}$ vector around the region in which the collision or infeasibility has occurred. Since we know the infeasible knot value and its interval in the knot vector, the new knot is added to the midpoint of the infeasible interval. Hence, only a limited interval of the knot vector $\boldsymbol{u_{new}}$ is updated. Note that, B-Splines' local support property allows local control over which the spline is infeasible. Specifically, this control is over the curve segments with $\pm k/2$ (where k is the order of the B-Spline) polygon spans around the displaced or newly added point. If the infeasibility still cannot be repaired; the back-propagation method is used. In this method, the spline change is carried to the previous generated spline. The random selected end velocity of the previous generated spline affects the beginning velocity (because of the velocity continuity) and the shape of the current spline. As seen in Line 18, the spline number is decreased to re-evaluate previous spline resulting in a change of the initial conditions of the infeasible proceeding spline.

If all these processes cannot repair the path, the *time scale* is changed to reallocate the min-max velocity and acceleration interval within the dynamically feasible interval that can be achieved by the vehicle. The end result is a time expanded or shortened flight path. Dynamic feasibility of all the preceding generated splines are checked from the beginning. If infeasibility occurs over any previously generated spline according to the new time scale, algorithm continues running from this spline numbers as seen in Line 23. A typical solution of the algorithm is shown in Fig. 5.

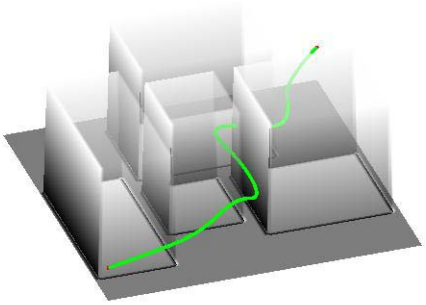

Fig. 6. Result dynamically feasible path

III. Simulation Results

We tested the performance of our method on some environments in varying ratio of obstacle-space to all work space and types. The computational times of the all phases of the algorithm are illustrated in Table I for 3D single-narrow-passage problem, city-like environment, mostly-blocked environment and challenging circuitous-tunnel problem (Fig. 7). All the experiments were conducted on a 3.00 GHz Intel Pentium(R) 4 processor and the average results are obtained over 20 runs.

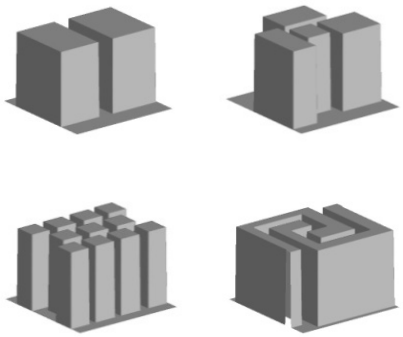

Fig. 7. Test environments: single-narrow passage, city-like environment, mostly-blocked environment and circuitous-tunnel.

TABLE I

KINODYNAMİC PATH CONSTRUCTION TIMES (SECONDS). RESULTS ARE AVERAGED OVER 20 RUNS

		Connectivity Path Planner	B-Spline Planner	Total Time
Single-Narrow Passage	avr	0.558 s	0.071 s	0.629 s
	std	0.343 s	0.094 s	0.354 s
City-Like Environment	avr	1.403 s	0.717 s	2.121 s
	std	1.672 s	0.891 s	1.741 s
Mostly-Blocked Environment	avr	5.045 s	0.514 s	5.561 s
	std	2.722 s	0.905 s	2.671 s
Circuitous Tunnel Problem	avr	24.494 s	0.175 s	24.67 s
	std	28.545 s	0.078 s	28,57 s

Increasing complexity of the environment, as shown in Table I, mainly increases computational time of the *connectivity path* that is implemented with a simplified version of RRT. Since the repairing part of the algorithm is visited much more often for a complex environment, computational time of the B-Spline based planner phase also increases. However, this rising rate does not grow exponentially like RRT based planner as seen in the circuitous-tunnel example. In addition, the solution times suggest that our method will be applicable for real-time implementations as the solution time is favorably comparable to implementation times. Fig. 8 shows the in-house developed robotic test platform that will be using for on ground verifications.

IV. Conclusion

In this work, a real-time implementable two-step planner strategy is implemented for obtaining 3D real-time flight-path generation for an unmanned helicopter flying in a dense and complex environment. The method uses RRT to generate a probabilistic path to connect start and goal regions. Later this is refined to obtain line-of-sight critical milestones are extracted. To obtain a fast agile and continuous motion plan we use a B-Spline method to generate probabilistically refined constant acceleration flight paths that pass through these milestones while providing collision-free and dynamically achievable flight paths. The

numerical implementations suggest that the method has real-time properties that can be used for unmanned helicopters operating in urban environments.

REFERENCES

[1] E. Frazzoli, M.A. Dahleh and E. Feron, "Real-Time Motion Planning for Agile Autonomous Vehicles", *AIAA J. of Guidance, Control, and Dynamics,* 25(1), pp. 116–129, 2002.

[2] L. E. Kavraki, P. Svestka, J.C. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces", In *Proc. IEEE Transactions on Robotics and Automation*, 12(4), pp. 566-580, 1996.

[3] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners", In *Proc. IEEE Int. Conf. Robotics & Automation (ICRA)*, pp 4420-4426, 2003

[4] V. Boor, M.H. Overmars, F.V. Stappen, "The Gaussian Sampling Strategy for Probabilistic Roadmap Planners", *IEEE Int. Conf. on Robotics & Automation*, pp. 1018–1023, 1999.

[5] D. Hsu, J.C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces", In *Proc. IEEE Int. Conf. on Robotics & Automation*, pp. 2719–2726, 1997.

[6] D. Hsu, "Randomized Single-query Motion Planning in Expansive Spaces", *Ph.D. Thesis*, Dept. of Computer Science, Stanford University, Stanford, CA ,2000.

[7] G. Sanchez, J.C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking", Int. *Journal of Robotics Researches*, pp. 403-407, 2003.

[8] E. Plaku, M.Y. Vardi, L.E. Kavraki, "Discrete search leading continuous exploration for kinodynamic motion planning", *Robotics: Science and Systems*, Atlanta, Georgia, 2007.

[9] S.M. LaValle, "Rapidly-exploring random trees: A new tool for path planning", *Technical Report 11*, Computer Science Dept., Iowa State University, 1998.

[10] D. Hsu, R. Kindel, J.C. Latombe, S. Rock," Randomized Kinodynamic Motion Planning with Moving Obstacles" ,*Int. J. of Robotics Research*, 21(3), pp. 233-255, 2002.

[11] S.M. LaValle, J.J. Kuffner, "Raandomized kinodynamic Planning", *Int. Journal of Robotic Research, 20(5), 378-400,2001*

[12] S. Griffiths, J. Sanuders, A. Curtis, T. Maclain, R. Beard," Obstacle and terrain aviodance for miniature aerial vehicles", *IEEE Robotics and Automation Magazine*, 2007.

[13] D. Hsu, L.E. Kavraki, J.-C. Latombe, R. Motwani and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners", In. Proc. *Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, pp. 141-153, 1998.

[14] O. B. Bayazit, D. Xie and N. Amato, "Iterative relaxation of constraints: a framework for improving automated motion planning", Int. Conf. on Intelligent Robots and Systems (IROS), pp. 3433- 3440, 2005.

[15] R. Bohlin and L. E. Kavraki. "A randomized algorithm for robot path planning based on lazy evaluation". In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, Handbook on Randomized Computing, pages pp. 221–249. Kluwer Academic Publishers, 2001.

[16] G. Song, S. L. Miller, and N. M. Amato. "Customizing PRM roadmaps at query time". In Proc. IEEE Int. Conf. Robot. Autom. (ICRA), pp. 1500–1505, 2001.

[17] C. L. Nielsen and L. E. Kavraki. "A two level fuzzy PRM for manipulation planning", Technical Report TR2000-365, Computer Science, Rice University, Houston, TX, 2000.

[18] E. Koyuncu, N. K. Ure, G. Inalhan,"A Probabilistic Algorithm for Mode Based Motion Planning of Agile Air Vehicles in Complex Environments", *Int. Fed. of Automatic Control World Congress (IFAC WC)*, 2008

[19] K. Komoriya and K. Tanie, "Trajectory Design and Control of a Wheel-type Mobile Robot Using B-spline Curve", *Int. Workshop on Intelligent Robots and Systems*, pp. 398-405, 1989.

[20] H. Eren, C. C. Fung and J. Evans, "Implementation of the Spline Method for Mobile Robot Path Control", Instrumentation and Measurement Technology Conference, pp. 739-744, 1999.

[21] V. F. Muñoz, A. Ollero, M. P. Novoa, A. S. Mata, "Mobile Robot Trajectory Planning with Dynamic and Kinematic Constraints", *Int. Conference on Robotics and Automation (ICRA)*, pp. 2802-2807, 1994.

[22] E. Paulos, "On-line Collision Avoidance for Multiple Robots Using B-Splines", UC Berkeley, Computer Science Technical Report, (UCB/CSD-98-977), 1998.

[23] E. Dyllong, A. Visioli, "Planning and real-time modifications of a trajectory using spline techniques", *Robotica*, Vol. 21, pp. 475-482, 2003.

[24] S. M. LaValle, "From dynamic programming to RRTs: Algorithmic design of feasible trajectories", *Control Problems in Robotics*, pp. 19-37. Springer-Verlag, Berlin, 2002.

[25] L. Piegl and W. Tiller, "The NURBS Book", Springer-Verlag, New York, NY, 1997.

[26] M. B. Miliam, "Real-Time optimal trajectory generation for constrained dynamical systems", PhD Thesis, California Institute of Technology, Pasedena, CA, 2003.

[27] I.K. Nikolos, K.P. Valavanis, N.C. Tsourveloudis, and A.N. Kostaras," Evolutionary algorithm based off-line path planner for UAV navigation", *Automatika,* vol. 42, no 3-4, pp. 143-150, 2001.

[28] B. Vazquez G., J. Humberto Sossa A. and Juan L. Diaz-de-Leon S, "Auto Guided Vehicle Control using Expanded Time B-Splines," Int. Conf. on Systems, Man, and Cybernetics, pp. 2786-2791, 1994.