International Journal of Mechanical, Industrial and Aerospace Sciences
ISSN: 2517-9950
Vol:12, No:5, 2018

Application of Rapidly Exploring Random Tree Star-Smart and G^2 Quintic Pythagorean Hodograph Curves to the UAV Path Planning Problem

Luiz G. Véras, Felipe L. Medeiros, Lamartine F. Guimarães

***Abstract*—**This work approaches the automatic planning of paths for Unmanned Aerial Vehicles (UAVs) through the application of the Rapidly Exploring Random Tree Star-Smart (RRT*-Smart) algorithm. RRT*-Smart is a sampling process of positions of a navigation environment through a tree-type graph. The algorithm consists of randomly expanding a tree from an initial position (root node) until one of its branches reaches the final position of the path to be planned. The algorithm ensures the planning of the shortest path, considering the number of iterations tending to infinity. When a new node is inserted into the tree, each neighbor node of the new node is connected to it, if and only if the extension of the path between the root node and that neighbor node, with this new connection, is less than the current extension of the path between those two nodes. RRT*-smart uses an intelligent sampling strategy to plan less extensive routes by spending a smaller number of iterations. This strategy is based on the creation of samples/nodes near to the convex vertices of the navigation environment obstacles. The planned paths are smoothed through the application of the method called quintic pythagorean hodograph curves. The smoothing process converts a route into a dynamically-viable one based on the kinematic constraints of the vehicle. This smoothing method models the hodograph components of a curve with polynomials that obey the Pythagorean Theorem. Its advantage is that the obtained structure allows computation of the curve length in an exact way, without the need for quadratural techniques for the resolution of integrals.

***Keywords*—**Path planning, path smoothing, Pythagorean hodograph curve, RRT*-Smart.

I. Introduction

SOME Unmanned Aerial Vehicles (UAVs) have limited maneuverability due to their kinematic constraints. Thus, in the path planning problem, the constraint coming from the dynamic capacity of the UAV must be taken into account in the solution of this problem. A solution to the path planning problem can be reached and made feasible for the UAV by two steps: finding a path that, in a given navigation environment, does not collide with obstacles; and smoothing the parts of the path that the UAV would not be able to perform, due to its limitations of maneuverability.

This work was supported in part by the National Council for Scientific and Technological Development from Brazil.

L. G. D. O. Veras is with the Associate Laboratory of Computation and Applied Mathematics, National Institute for Spatial Research, São José dos Campos, SP, Brazil, on leave from the Federal Institute of Education, Science and Technology of São Paulo, São Paulo, Brazil (e-mail: gustavo_veras@ifsp.edu.br).

F. L. L. Medeiros is with the C4ISR Division, Institute for Advanced Studies, São José dos Campos, SP, Brazil (e-mail: felipe@ieav.cta.br).

L. N. F. Guimarães is with the Nuclear Energy Division, Institute for Advanced Studies, São José dos Campos, SP, Brazil (e-mail: guimarae@ieav.cta.br).

The algorithms normally used for this type of problem are based on sampling. In this class of algorithms the navigation environment is not represented explicitly. Instead, samples of the navigation environment are collected to compose your solution. To verify if the collected sample generates a collision situation, these algorithms use a collision module responsible for this verification. These classes of algorithms are popular in large part because of their computational efficiency and simplicity. The Rapidly-exploring Random Tree (RRT) algorithm [5] is one of the most studied algorithms. They are probabilistically complete, that is, if a solution exists, the chance of the algorithm finding such a solution approaches 1 as the number of samples collected increases. The RRT algorithm has the ability to quickly explore the navigation environment and return a solution. In this algorithm, samples of the navigation environment are collected randomly. Each collected sample is added to closest node of the tree, if the straight line segment formed by those nodes is collision free. It is expanded until a path between the start and end navigation point for the UAV is found.

Although RRT is probabilistically complete, there is no guarantee that the solutions found will approach the best possible path (the one with the shortest length) for a given navigation environment. The RRT* algorithm [6] incorporates asymptotic optimality to RRT algorithm. In this algorithm, for each new node added, the neighborhood of that node is generated, considering a given distance. As in the other algorithm, the new node is not added to the nearest node. Instead, the new node is added to the neighbor node that provides the smallest path to the root node (navigation starting point). Next, it is verified if the other nodes of the neighborhood can also be reconnected with this new node, reducing the cost of the path between them and navigation root node. Thus, when a complete path is planned, it is expected to obtain the shortest path between the root node and the final navigation point [6].

The RRT* presents a considerably longer planning time due to the addition of the process of reconnection of the edges during its expansion. In addition, the algorithm only ensures that the shortest path is planned if the planning time is infinite. By means of intelligent sampling and path optimization, the RRT *-Smart algorithm [4] is able to return lower cost paths faster than the RRT*, being the candidate suitable for UAV path planning.

Paths planned by sampling-based algorithms are composed by waypoints connected with straight line segments, forming

sharp pointed corners. These sharp pointed corners are not suitable for the trajectory of UAVs with curvature constraints. To smooth the sharp corners of the paths planned by an algorithm such as RRT, RRT* and RRT*-Smart, it is necessary to use structures such as Bezier and Splines curves. These structures require integral resolution methods to obtain their length, which entails considerable additional cost depending on the application. In [9], the Pythagorean Hodograph curve (also called PH curve) is introduced, where the hodograph components of a parametric curve $r(\xi)$ are defined by polynomials that combined result in a new polynomial satisfying with them the Pythagorean Theorem. Its advantage, among others [1], is that the obtained structure allows computation of the curve length in an exact way, without the need for quadrature techniques for the resolution of integrals, a very useful feature in the path planning problem based on costs. In addition they may be suitable for smoothing the sharp edges presented by the sampling methods, as shown in [3].

However, it should be taken into account that in cases where a corner is very close to one of the convex vertices of the obstacles, the generated curve can enter its limits, which would lead to an eventual collision of the UAV during its navigation.

In this work, a scheme is proposed to define the distance to the generation of safety enclosures around the obstacles so that the smoothed corners near the convex vertices do not penetrate the limits of these obstacles. From the structures defined for the construction of PH curves, it is possible to obtain the ideal distances for the creation of the wrappings, given the curvature of the UAV. In order to guarantee continuity of the curve with the segments that were the corners, in [3] a formulation of PH curves with G^2 continuity with the segments is proposed. This work is organized as follows: in Section II the RRT, RRT* and RRT* -Smart algorithms are described; in Section III the components that define a PH curve are described; in Section IV, the formulations of the PH curve for the sharpening smoothing problem are described; in Section V, the distance definition scheme for the generation of safety enclosures is presented; in Section VI, the results obtained are described; and in Section VII, the conclusions of this work and the future works are described.

II. RRT* and RRT*-Smart Algorithms

The RRT algorithm for path planning is described in Alg. 1. RRT plans a navigation path in a Q configuration space, which are the set of points/positions of a navigation environment. Q is divided into two subsets, Q_{free}, representing the navigable regions of the navigation environment, i. e., the regions without obstacles and no collision risks, and Q_{obs}, the spatial representation of the obstacles. The operations performed by the algorithm are defined as follows:

- **RAND_CONFIG**: generate a random position q_{rand} in the configuration space.
- **NEAREST_NODE**: search the nearest node q_{near} from q_{rand} in G.
- **NEW_CONFIG**: generate a new node q_{new} in the line segment $\overline{q_{near}q_{new}}$ with a distance from q_{near}.
- **EXTEND**: expand the tree, assigning q_{near} as the predecessor/parent node of q_{new} and q_{new} to G.
- **PATH**: collect all nodes that connect q_{goal} to the q_{init} that form the path R.

The root node of the tree ($q_{start} \in Q_{free}$) is the starting point of the path to be planned. The algorithm works by expanding a G tree randomly from the root node until one of its branches reaches the final point ($q_{goal} \in Q_{free}$) of the path to be planned, or until a maximum number of iterations (n) is reached. Since each node is a sample/point of the navigation environment and has information about its predecessor node, the path is planned from the q_{goal} point to the q_{start} origin point, adding each node that connects them to a path R. A leaf node q_{new} is a point of the straight line segment $\overline{q_{near}q_{rand}}$, such that: $q_{goal} \in Q_{free}$ is a point generated as a random sampling of the navigation environment; q_{near} is the node closest to q_{rand}; the distance between q_{near} and q_{new} has the constant value Δq; and the straight line segment $\overline{q_{near}q_{new}}$ is collision-free, that is, it does not intercept any obstacle (Q_{obs}) of the navigation environment.

Algorithm 1 RRT algorithm

```
1: procedure RRT(Q, q_start, q_goal, Δq, l_d, n)
2:     G ← {}
3:     R ← {}
4:     s ← 0
5:     i ← 0
6:     while (s = 0) and (i ≤ n) do
7:         i ← i + 1
8:         q_rand ← RAND_CONFIG(Q)
9:         q_near ← NEAREST_NODE(q_rand, G)
10:        q_new ← NEW_CONFIG(q_near, q_rand, Δ_q)
11:        if q_near q_new do not intercept Q_obs then
12:            EXTEND(G, q_near, q_new)
13:            if d(q_new, q_goal) ≤ l_d and (q_new q_goal) do not
                   intercept Q_obs) then
14:                EXTEND(G, q_new, q_goal)
15:                s ← 1
16:                R ← ROUTE(q_init, q_goal)
```

RRT* will deviate from RRT in the way tree expansion occurs. During the expansion operation, a path cost function formed by the addition of a new node is considered. The cost function represents a measure of generic magnitude (depending on the state space associated with the tree). The main difference in RRT* is the replacement of RRT's $EXTEND$ by the RRT operation RRT_STAR_EXTEND. Additional RRT* operations are described below:

- **RRT_STAR_EXTEND**: extend the tree by selecting the neighbor node $q_{neighbor} \in Q_{neighbors}$ that allows creating a path with the smaller cost from q_{new} to q_{init}. $Q_{neighbors}$ is constituted by all nodes that are at a distance $\beta(log(n)/n)$ from q_{new}.
- **REWIRE**: q_{new} is connected as a predecessor node of the nodes from $Q_{neighbors}$ if these nodes form with q_{new} a lower cost path to the q_{init}.

The cost is assigned to each pair of edges formed between two consecutive nodes (q_i, q_{i+1}). Each pair has an associated cost c. The cost of the path delimited by any two nodes of the

tree, with edges and intermediate nodes that connect them, is calculated by the $COST(q, q_n)$ function, which is defined as:

$$COST(q_n) = \sum_{i=1}^{n} c(q_i, q_{i+1}) \quad (1)$$

where n is the number of nodes of the path into the interval $[q, q_n]$, q_i is the i-th node of the tree, q_{i+1} is the successor node to q_i and $c(q_i, q_{i+1})$ is the cost from each pair of nodes (q_i, q_{i+1}).

Algorithm 2 Extend procedure of RRT* algorithm

```
1: procedure RRT_STAR_EXTEND(G, q_near, q_new)
2:     Q_neighbors ← NEIGHBORHOOD(G, q_new)
3:     q_min ← {}
4:     for q_neigh ∈ Q_neighbors do
5:         if q_neigh q_new do not intercept Q_obs then
6:             c' ← COST(q_new) + c(q_new, q_neigh)
7:             if c' < COST(q_near) + c(q_new, q_near) then
8:                 q_min ← q_neigh
9:     G.ADD_VERTEX(q_min)
10:    G.ADD_EDGE(q_min, q_new)
11:    REWIRE(Q_neighbors, q_min, q_new)
12:    return
```

Algorithm 3 Rewire procedure of RRT* algorithm

```
1: procedure REWIRE(Q_neighbors, q_min, q_new)
2:     for q_neigh ∈ Q_neighbors do
3:         if q_neigh q_new do not intercept Q_obs then
4:             c' ← COST(q_new) + c(q_neigh, q_new)
5:             if c' < COST(q_neigh) then
6:                 q_parent ← PARENT(q_neigh)
7:                 G.REMOVE_PARENT(q_neigh, q_parent)
8:                 G.ADD_PARENT(q_neigh, q_new)
```

For each q_{new}, RRT* algorithm verifies if there are no obstacles between it and the selected q_{near}. If it does not, the neighbor nodes of qnew are selected within the range of the radius defined by $\beta(log(n)/n)$ from q_{new} and assigned to $Q_{neighbors}$. Each $q_{neighbor} \in Q_{neighbors}$ is verified with q_{new} to identify the one that allows the connection with lower cost, i. e., the $q_{neighbor}$ that allows the local shortest path between q_{start} and q_{new}, passing through $q_{neighbor}$.

If $COST(q_{init}, q_{neighbor}) + c(q_{new}, q_{neighbor}) < CUSTO(q_{init}, q_{near}) + c(q_{new}, q_{near})$, then $q_{neighbor}$ generates a path shorter than the path with q_{near} and q_{new}. The $q_{neighbor}$ with the lower cost with q_{new} will be selected. This process is defined by RRT_STAR_EXTEND, as described in the Alg. 2.

After defining the least cost path (local shortest path) of q_{new} in $Q_{neighbors}$, it is verified the possibility of reducing the cost of the paths of the other neighbors by connecting them to the new node (q_{new}) added by means of the $REWIRE$ function. For each neighbor node q_{neigh}, if $COST(q_{new}) + c(q_{neighbor}, q_{new}) < COST(q_{neighbor})$ then its predecessor changed to q_{new}. This process is described in Alg. 3.

A. Better Convergence Solution with RRT-Smart Algorithm*

Proposed in [4], the RRT*-Smart is an algorithm based on the RRT*, where two new concepts were incorporated: the path optimization; and intelligent sampling. It was presented in [4] that this algorithm has a higher convergence speed for the optimal solution to be obtained for a given navigation environment than the RRT*. This algorithm is described in Alg. 4. Additional procedures in RRT*-Smart are described below:

- **PATH_OPTIMIZATION**: the nodes that form the path from q_{goal} to q_{init} visible to each other are connected directly, removing intermediaries between them. The visible ones among them are those that can form a new collision-free edge without the need of other nodes for such a part of both.
- **COLLECT_BEACONS**: collect the nodes of the optimized path. These nodes, called beacons, are used to force sampling of navigational space points near them.

Algorithm 4 RRT*-Smart algorithm

```
1: procedure RRT(Q, q_start, q_goal, Δq, l_d, n, t)
2:     G ← {}
3:     R ← {}
4:     i ← 0
5:     cost_old ← ∞
6:     while (i ≤ n) do
7:         i ← i + 1
8:         if i = t + b, t + 2b, t + 3b... then
9:             q_rand ← RAND_CONFIG(Q_beacons)
10:        else
11:            q_rand ← RAND_CONFIG(Q)
12:        q_near ← NEAREST_NODE(q_rand, G)
13:        q_new ← NEW_CONFIG(q_near, Δq)
14:        if q_near q_new do not intercept Q_obs then
15:            RRT_STAR_EXTEND(G, q_near, q_new)
16:            if d(q_new, q_goal) ≤ l_d and (q_new q_goal) do not
                   intercept Q_obs) then
17:                EXTEND(G, q_new, q_goal)
18:                founded_path ← true
19:            if founded_path then
20:                t ← i
21:                cost_new ←
                   PATH_OPTIMIZATION(q_init, q_goal)
22:                if cost_new < cost_old then
23:                    R ← ROUTE(q_init, q_goal)
24:                    Q_beacons ←
                       COLLECT_BEACONS(q_init, q_goal)
25:                    cost_old ← cost_new
26:            founded_path ← false
```

Given a tringle formed by the waypoints w_a, w_b and w_c, path optimization is a strategy based on the fact that the largest side ($w_a w_b$) is always smaller than the sum of the sizes of the two smaller sides ($w_b w_c + w_a w_c$) of the triangle. Then, if two waypoints can connected directly without collision, the waypoints and edges between those two waypoints may be eliminated. Alg. 5 presents this strategy.

Smart sampling is an approach that uses the nodes of a path previously planned to induce the collection of new samples close to them. When a path is planned, your nodes are stored

Algorithm 5 Path Optimization procedure from RRT*-Smart algorithm

```
procedure OPTIMIZE_PATH(q_init, q_goal)
    q_current ← q_goal
    while q_current ≠ q_init do
        q_candidate−parent ← PARENT_TO_PARENT(q_current)
        if q_current q_candidate−parent do not intercept Q_obs then
            q_current ← q_parent−to−parent
```

in z_{beacon}. These nodes are updated each time a path shorter than the last one is found. Then, as the path planning iterations proceed, eventually new nodes are generated next to one of the z_{beacon} nodes. This occurs given the frequency defined by

$$n + cb \tag{2}$$

where n is the iteration in which the first path was planned, c is a value incremented at each intelligent sampling and b a constant which defines the scale of the frequency that the new nodes will be sampled near of one of the nodes on Z_{beacon}. It is verified at each iteration of the planning if the unitarily increased planning index i has value equal to (2). If it is true, a new node is generated arround a node randomly selected in Z_{beacon} until a distance defined by a given radius. Otherwise, the node is randomly sampled, as in RRT and RRT*.

These two strategies were incorporated into the RRT*, reducing considerably the acquisition of a path with reduced cost.

III. Pythagorean Hodograph curve

Proposed in [9], the Hodographic Curve of Pythagoras (also denominated PH curve) and a type of curve where the components of its hodograph have a relation with a polynomial in such a way as to meet Pythagoras's Theorem, or the sum of the squares of the components of the hodograph is equal to the square of this polynomial. A direct impact of this ratio is that the integral of the calculation of the length of the curve can be transformed into a closed form, allowing its algebraic resolution, thus avoiding the need for of quadratures for the resolution of integrals. This characteristic is used to gain computational advantage in this work. The formulations that define the Pythagorean Hodograph will be defined below. Examples and other properties can be verified in [1].

Let a curve be defined by $r(\xi) = (x(\xi), y(\xi))$ e $\xi \in [0, 1]$ the parameter that defines each position in the curve. The length of the curve is given by:

$$S = \int_0^1 ||\mathbf{r}'(\xi)|| \tag{3}$$

where S is the total length of the curve and $r'(\xi)$ is the velocity vector of the curve (first derivative). The equation above can be rewritten as:

$$S = \int_0^1 \sqrt{x'(\xi)^2 + y'(\xi)^2} \tag{4}$$

where $x'(\xi) = dx/d\xi$ and $y'(\xi) = dy/d\xi$. By relating the internal root term to $\sigma(\xi)$, the curve length formulation can be defined by

$$S = \int_0^1 \sqrt{\sigma(\xi)^2} = \int_0^1 |\sigma(\xi)| \tag{5}$$

It can be seen that $\sigma(\xi)$ represents the parametric speed of the curve. It is necessary to define $\sigma(\xi)$ in terms of polynomials that satisfy (30). Given the polynomials $u(\xi)$, $v(\xi)$, the following definitions of $x'(\xi)$ and $y'(\xi)$ satisfy the Pythagorean Theorem in the definition of $\sigma(\xi)$

$$x'(\xi) = u(\xi)^2 - v(\xi)^2, \quad y'(\xi) = 2u(\xi)v(\xi) \tag{6}$$

Therefore, $\sigma(\xi)$ can be defined in terms of $u(\xi)$, $v(\xi)$ as

$$\sigma(\xi) = u(\xi)^2 + v(\xi)^2, \tag{7}$$

Incorporating these definitions into a quintic curve in the Bézier form, defined by

$$\mathbf{r}(\xi) = \sum_{k=0}^{5} \mathbf{p}_k \binom{5}{k} (1-\xi)^{5-k} \xi^k, \tag{8}$$

the control points $\mathbf{p}_k$, given by Bernstein coefficients, are defined in terms of the polynomials $u(\xi)$ and $v(\xi)$ as

$$\begin{aligned}
\mathbf{p}_1 &= \mathbf{p}_0 + \tfrac{1}{5}(u_0^2 - u_0^2, 2u_0u_0) \\
\mathbf{p}_2 &= \mathbf{p}_1 + \tfrac{1}{5}(u_0u_1 - v_0v_1, u_0v_1 + u_1v_0) \\
\mathbf{p}_3 &= \mathbf{p}_2 + \tfrac{2}{15}(u_1^2 - u_1^2, 2u_1u_1) + \tfrac{1}{15}(u_0u_2 - v_0v_2, u_0u_2 + u_2v_0) \\
\mathbf{p}_4 &= \mathbf{p}_3 + \tfrac{1}{5}(u_1u_2 - v_1v_2, u_1u_2 + u_2v_1) \\
\mathbf{p}_5 &= \mathbf{p}_4 + \tfrac{1}{5}(u_2^2 - v_2^2, 2u_2v_2)
\end{aligned} \tag{9}$$

where $\mathbf{p}_0$ is arbitrarily specified. The polynomial that defines the parametric speed $\sigma(\xi)$ is given by

$$\sigma(\xi) = \sum_{k=0}^{4} \sigma_k \binom{4}{k} (1-\xi)^{4-k} \xi^k, \tag{10}$$

The Bernstein coefficients are calculated by

$$\begin{aligned}
\sigma_0 &= u_0^2 + u_0^2 \\
\sigma_1 &= v_0u_1 + v_0v_1 \\
\sigma_2 &= \tfrac{2}{3}(u_1^2 + v_1^2) + \tfrac{1}{3}(u_0u_2 + v_0v_2) \\
\sigma_3 &= u_1u_2 + v_1v_2 \\
\sigma_4 &= v_2^2 + u_2^2
\end{aligned} \tag{11}$$

The length of the curve, given as a function of the coefficients defined in (11), is

$$S = \frac{\sigma_0 + \sigma_1 + \sigma_2 + \sigma_3 + \sigma_4}{5} \tag{12}$$

The curvature value of the curve is given by

$$\kappa = \frac{2[u(\xi)v'(\xi) - u'(\xi)v(\xi)]}{u(\xi)^2 + v(\xi)^2} \tag{13}$$

A. Complex Representation of Planar Quintic PH Curve

The PH curves, described in the previous section, were defined in the Cartesian plane. However, the *PHquintic* library, proposed in [8] and that was used in this work, use the representation of the curve in the complex plane, where the coordinates of the curve are defined by a real and imaginary axis. The advantage the use of complex numbers is the significant simplification in the expressions associated with the PH curve. PH curves with complex representations were introduced in [2]. For brevity, this section will only present the formulations of the control points, the coefficients of parametric velocity, the coefficients of the length of the curve and other properties of the curve associated with complex variables. The relationships necessary to obtain the PH curve in the complex plane can be found in [1] and [2].

To define a parametric curve in the complex plane, the polynomials associated with the hodograph in (6) must be defined in complex form such that

$$\mathbf{w}(\xi) = u(\xi) + i\, v(\xi) = \sum_{k=0}^{m} \mathbf{w}_k \binom{m}{k} (m-\xi)^{m-k} \xi^k; \quad (14)$$

with $m = 2$ and Bernstein coefficients $\mathbf{w}_k = u_k + i\, v_k$ by integration of the equality

$$\mathbf{r}'(\xi) = \mathbf{w}(\xi)^2 \quad (15)$$

The control points are defined in terms of $\mathbf{w}(\xi)$ as

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_0 + \tfrac{1}{5}(\mathbf{w}_0^2) \\ \mathbf{p}_2 &= \mathbf{p}_1 + \tfrac{1}{5}(\mathbf{w}_0 \mathbf{w}_1) \\ \mathbf{p}_3 &= \mathbf{p}_2 + \tfrac{1}{5}(\tfrac{2\mathbf{w}_1^2 + \mathbf{w}_0 \mathbf{w}_2}{3}) \\ \mathbf{p}_4 &= \mathbf{p}_3 + \tfrac{1}{5}(\mathbf{w}_1 \mathbf{w}_2) \\ \mathbf{p}_5 &= \mathbf{p}_4 + \tfrac{1}{5}(\mathbf{w}_2^2) \end{aligned} \quad (16)$$

The Bernstein coefficients of the parametric velocity $\sigma(\xi)$ in (10) are defined as

$$\begin{aligned} \sigma_0 &= |\mathbf{w}_0|^2 \\ \sigma_1 &= Re(|\bar{\mathbf{w}}_0 \mathbf{w}_1) \\ \sigma_2 &= \tfrac{2|\mathbf{w}_1|^2 + Re(\bar{\mathbf{w}}_0 \mathbf{w}_2)}{3} \\ \sigma_3 &= Re(\bar{\mathbf{w}}_1 \mathbf{w}_2) \\ \sigma_4 &= |\mathbf{w}_2|^2 \end{aligned} \quad (17)$$

In terms of $w(\xi)$, the parametric velocity and the curvature can be formulated as

$$\sigma(\xi) = |\mathbf{w}(\xi)|^2, \kappa(\xi) = 2\frac{Im(\bar{\mathbf{w}}(\xi)\mathbf{w}'(\xi))}{\sigma(\xi)^2} \quad (18)$$

The total curve length is still given by (12).

IV. Round Corner Problem with G^2 PH Quintic Curves

In the problem of rounding (smoothing) of corners, the junction between two segments must be transformed into a curve such that there is continuity between the segments and the end points of the curve, that is, $r(0)$ and $r(1)$ must belong to each of the segments that form the corner, respectively. The use of quintic PH curves for this problem was proposed in [3].

The PH curve is modeled so that it meets a continuity curve G^2 - where position, tangents and curvature must be equal for both segments along the curve. The formulations for the PH curve in smoothing corner problem [3] are described in this section."

There are three points represented by p_i, p_c and p_o, where $\overline{p_i p_c}$ and $\overline{p_c p_o}$ are distinct segments interconnected by p_c. Defining the canonical form of representation of these points as

$$p_i = (0,0),\ p_c = (L,0),\ p_0 = (1 + cos(\theta)L, sin(\theta)L) \quad (19)$$

where $L = |p_c - p_i| = |p_o - p_c|$ and θ is the displacement angle of the $\overline{p_c p_o}$ in relation to the $\overline{p_i p_c}$. The canonical form and some examples of smoothed corners by PH curve are described in Fig. 1.

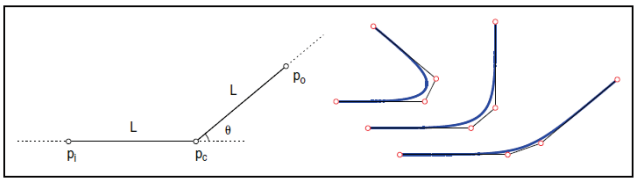


Fig. 1 Canonical form for the smoothing problem and examples, considering $\theta = \pi/4$, $\theta = \pi/2$ and $\theta = 3\pi/4$. Font: [3]

The solution to the canonical form, as defined in [3], is given in terms of L and θ by

$$\mathbf{w}_0 = \lambda\sqrt{L}, \quad \mathbf{w}_1 = 0, \quad \mathbf{w}_2 = \lambda\sqrt{L} exp(i\frac{1}{2}\theta) \quad (20)$$

where

$$\lambda = \sqrt{(30cos(\frac{1}{2}\theta))/(6cos(\frac{1}{2}\theta) + 1)} \quad (21)$$

The parameter velocity in (10) has the coefficients of Bernstein defined as

$$\begin{aligned} \sigma_0 &= \lambda^2 L \\ \sigma_1 &= 0 \\ \sigma_2 &= \tfrac{\lambda^2 L cos(\frac{1}{2}\theta)}{3} \\ \sigma_3 &= 0 \\ \sigma_4 &= \lambda^2 L \end{aligned} \quad (22)$$

The curvature at any point ξ of the curve is given by

$$\kappa(\xi) = 4\lambda^2 L sin(\frac{1}{2}\theta)(\frac{(1-\xi)\xi}{\sigma(\xi)^2}) \quad (23)$$

The highest curvature of the curve κ_e, given in $\xi = 1/2$, is calculated by

$$\kappa_e = \frac{32(6cos(\frac{1}{2}\theta) + 1)tan(\frac{1}{2}\theta)}{15L(cos(\frac{1}{2}\theta) + 1)^2} \quad (24)$$

The total arc length is given by

$$S = \frac{2L(6 + cos(\frac{1}{2}\theta)cos(\frac{1}{2}\theta)}{6cos(\frac{1}{2}\theta) + 1} \quad (25)$$

The curve generated by these formulations generates a deviation at the midpoint of the curve $r(1/2)$ at the point p_c equal to

$$\delta = \frac{(3cos(\frac{1}{2}\theta)+8)|sin(\frac{1}{2}\theta)|L}{8(6cos(\frac{1}{2}\theta)+1)} \quad (26)$$

The formulations for the PH curve as a function of L and θ are used in the solution presented in the next section.

V. Smoothing Paths Planned by RRT*-Smart with G^2 PH Quintic Curves

In the path planning problem applied to real scenarios, it should be considered to use a safety hulls around the obstacles, in case paths are planned that are very close to the obstacle limits, thus avoiding the risk of collision of the UAV. Nevertheless, the path planned by the RRT*-Smart is increasingly close to the edges of the obstacles as the number of iterations to get it grows. Thus, it is important to define the safety hull at a safe distance from the obstacles, which ensures that the curve that smoothes the path at the junctions between the segments does not invade on the boundaries these obstacles. In this section we present a strategy of defining this distance from a maximum curvature that UAV theoretically is capable of performing. This is important in order to avoid that in a geometry presented by the path it is possible to create a free collision curve but in another it is not possible. The presented strategy defines a single safety envelope distance for the constraints of a navigational environment so that any smoothed curve close to the convex vertices is free of collision.

For a safety hull generated at a distance d from an obstacle and selecting one of the convex vertices of this obstacle, exists a convex vertex at a postion at a distance $d\ sec(\frac{1}{2})$ in the direction of the hull generation from the vertex of the obstacle, where θ is the opening angle of one segment relative to the other in counterclockwise direction (as illustrated in the canonical case, ilustrated in Fig. 1) [3]. Thus, the following relationship can be defined to prevent a curve from crossing any obstacle [3]:

$$d\ sec(\frac{1}{2}\theta) > \delta \quad (27)$$

Be the curvature value of a UAV given by κ_{UAV}, the L_{UAV} distance from p_c that generates a curve $r(\xi)$ with the curvature value equal to that of the $\xi = 1/2$ (maximum curvature of the curve), can be obtained from (24), reformulating it as

$$L_{UAV} = \frac{32(6cos(\frac{1}{2}\theta)+1)tan(\frac{1}{2}\theta)}{15\kappa_{UAV}(cos(\frac{1}{2}\theta)+1)^2} \quad (28)$$

By associating L_{UAV} with (27), the distance of the obstacles to the UAV with curvature κ_{UAV} can be obtained by

$$d > \frac{(3cos(\frac{1}{2}\theta)+8)|sin(\frac{1}{2}\theta)|L_{UAV}}{8(6+sec(\frac{1}{2}\theta))} \quad (29)$$

Let d_i be the distance resulting from (29) for the i-th vertex of an obstacle of the navigation environment with the curvature κ_{UAV}, the value d_{Obs}, which defines the distance that the security hull must be generated for this obstacle, is given by

$$d_{Obs} = max(d_i) \quad (30)$$

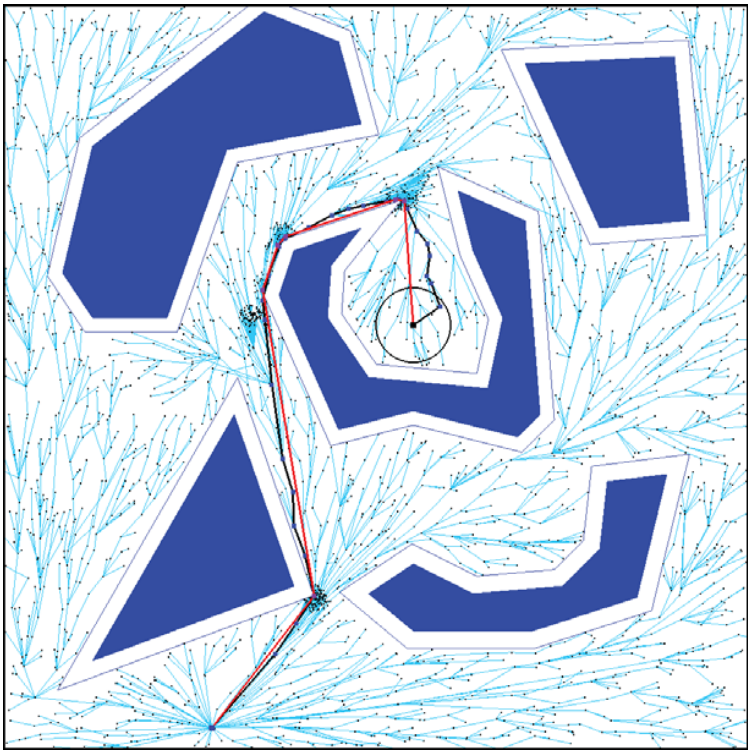
Since the distance d_{Obs} must be greater than the value of d in the formulation (29) to guarantee non-collision of the UAV during the navigation by the curve, the width d_{UAV} of the UAV can added to it. In this way it is guaranteed that the UAV of curvature κ_{UAV} will be able to carry out its navigation by curves constructed close to any convex vertex of the navigation environment. A direct advantage of this method is that as the distance is defined algebraically from the constraint (27), it is not necessary to perform a collision check for the obtained curve.

VI. Results e Analysis

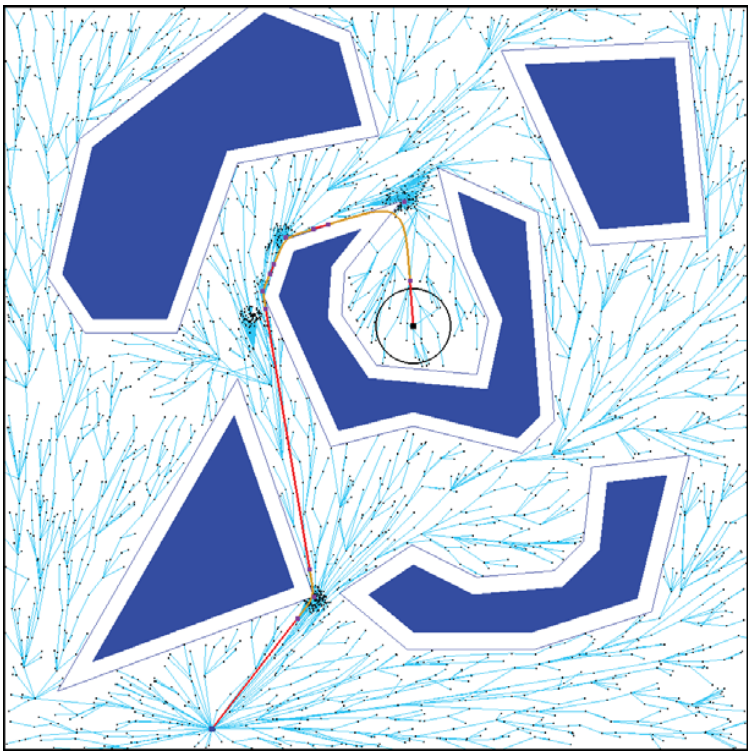
Simulations were performed for two UAV curvature values, equal to 0.009 and 0.015. A navigation environment with 5 obstacles of different geometries was used in the experiment. This navigation environment has coordinate ranges equal to $[0, 5000]$ on the x-axis and $[0, 5000]$ on the y-axis. The algorithms were implemented in the C ++ Language. To support the development of the *OpenGL* graphic library solution, the *CGAL* computational geometry library [7] and the library with the basic structures of the PH curve *PHquintic* [8] were used

Two simulations were performed, each for one of the UAV curvature values specified at the beginning of this section. Initially, the distance d is assigned to the value of curvature considered by means of the strategy described in Section V. Security hulls are remotely generated with $d_{Obs} + d_{UAV}$ from each obstacle in the navigation environment. The value adopted for d_{UAV} in this experiment is equal to 2 on the navigation space scale. For simplicity, the distance from the safety hulls to the obstacles will be referenced only as d_{Obs}. The path planning is performed with the RRT*-Smart algorithm, considering the collision check with the safety hulls generated, and not the contour of the obstacles. The algorithm is configured for a planning with 4000 iterations, so that it is possible to reasonably minimize the cost of the resulting path. With the generated path, the corners formed by the junction of the segments that connect q_{init} and q_{goal} of the tree are smoothed using the PH curve. The costs between smoothed and non-smoothed paths were compared. The cost for each segment is obtained by means of the Euclidean distance between the points that define it. The cost for the curve is obtained by (12).

Figs. 2 and 3 show the planned paths for each UAV curvature considered in this work. In Figs. 2(a) and 2(b) the paths for $k_{UAV} = 0.009$, with and without smoothing, are presented, respectively. Figs. 3(a) and 3(b) present, respectively, the paths for $\kappa_{UAV} = 0.015$, with and without smoothing. As indicated in Table I, the lengths of the paths generated for each value of κ_{UAV} are smaller in relation to the non-smoothed paths. For $\kappa_{UAV} = 0.009$ there was a reduction of 3.21% in path cost and for $\kappa_{UAV} = 0.015$ the reduction was 1.88%. Therefore, the lower the curvature value, the greater the reduction of the path cost generated by the smoothing in relation to its un-smoothed version. Because

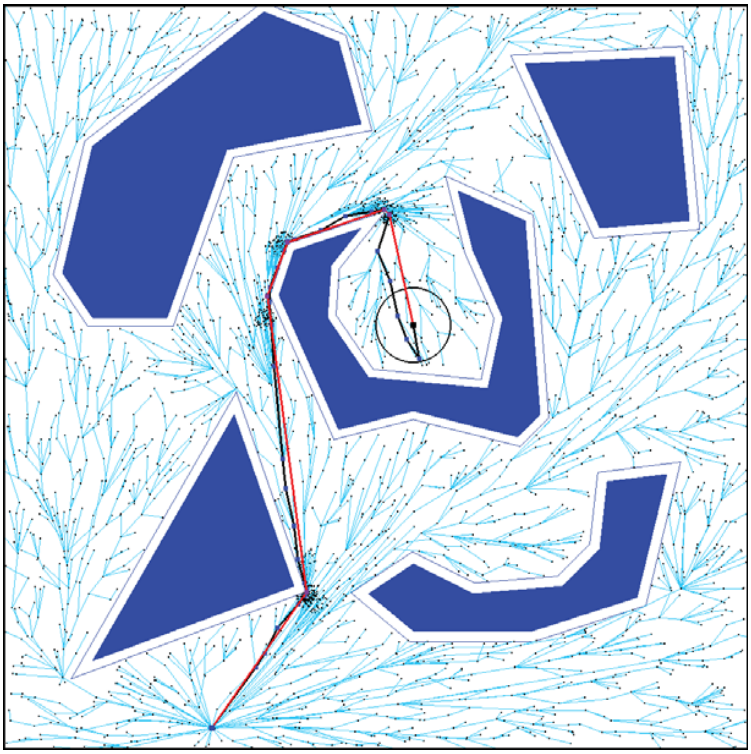


(a)

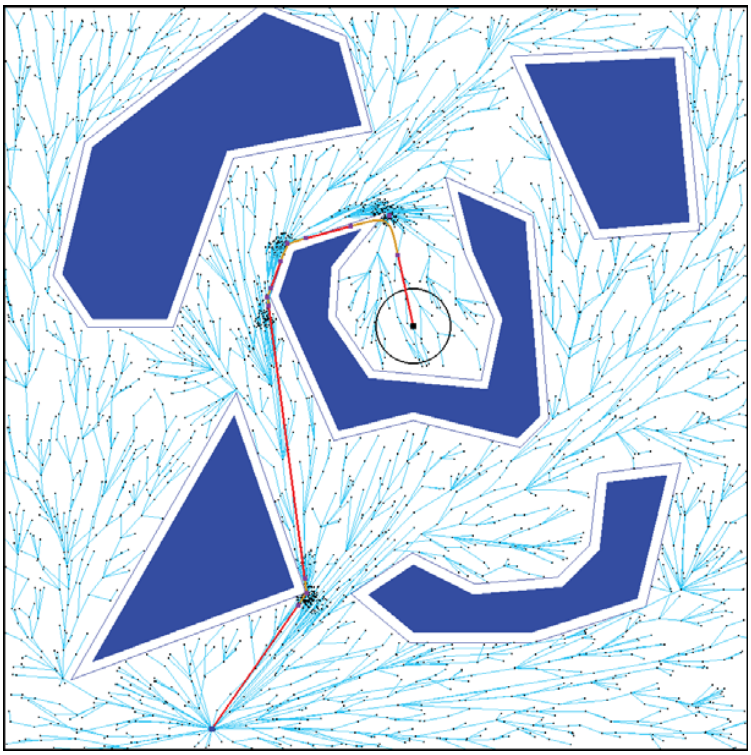


(b)

Fig. 2 Path planned by the RRT*-Smart algorithm without smoothing (a) and with smoothing (b) with PH curves for $\kappa_{UAV} = 0.009$. The black path in (a) is not optimized by the process and the red path is optimized. In (b), the curves generated for the sharp edges are represented by the segments in the beige color. The security hulls are represented by the blue color. Black circunferences represents minimal distance to connect q_{new} to q_{goal}



(a)



(b)

Fig. 3 Path planned by the RRT*-Smart algorithm without smoothing (a) and with smoothing (b) with PH curves for $\kappa_{UAV} = 0.015$. The black path in (a) is not optimized by the process and the red path is optimized. In (b), the curves generated for the sharp edges are represented by the segments in the beige color. The security hulls are represented by the blue color. Black circunferences represents minimal distance to connect q_{new} to q_{goal}

TABLE I
COSTS OF PATHS SMOOTHED BY PH CURVES, AND COSTS OF PATHS PLANNED WITHOUT SMOOTHING, CONSIDERING DIFFERENT VALUES OF κ_{UAV}

κ_{UAV}	Smoothed route cost	Not smoothed route cost	RRT*-Smart time planning
0.009	5099.210	5268.710	4.629
0.015	4891.920	4985.790	5.024

TABLE II
VALUES OF L AND d_{Obs} DEFINED FOR EACH OBSTACLE OF THE NAVIGATION ENVIRONMENT, CONSIDERING DIFFERENT VALUES OF κ_{UAV}

	$\kappa_{UAV} = 0.009$		$\kappa_{UAV} = 0.015$	
Obstacles	L	d_{Obs}	L	d_{Obs}
1	299.068	93.793	179.441	56.276
2	417.910	102.277	250.746	61.366
3	251.821	90.699	151.093	54.420
4	570.376	113.609	342.225	68.165
5	429.676	103.147	257.805	61.888

a smaller curvature leads to higher values of d_{Obs}, which increases the cost of the path planned by the RRT*-Smart, since the area taken by the security hulls is larger. In Table II, the values of d_{Obs} and L for the safety hull generation are summarized to determine the smoothing points in the segments of the path for each value of curvature of the UAV.

VII. CONCLUSION

In this work, a scheme was presented for the generation of smoothed paths for safe navigation of UAVs in a navigation environment with static obstacles. Initially a path is planned using the RRT*-Smart algorithm. Subsequently, the sharp corners formed by the junction of the straight line segments of the solution generated by RRT*-Smart are smoothed using G^2 quintic PH curves.

A scheme of how to define the distance for the generation of the PH curve based on the safety hulls of obstacles has also been presented. The estimation of the suitable distance for the construction of the safety hulls is important to ensure that the path does not intercept any obstacle, avoidind eventual collisions of the UAV during its navigation. This distance is defined from formulations of the PH curve for a given curvature of the air vehicle.

In future works, the geometry of the PH curves will be associated with a dynamic model of an UAV, aiming to control its speed and acceleration for navigation through paths planned by the strategy presented in this work.

REFERENCES

[1] Farouki, Rida T. Pythagorean—hodograph Curves. Springer Berlin Heidelberg, 2008.
[2] Farouki, Rida T. The conformal map $z \rightarrow z^2$ of the hodograph plane. Computer Aided Geometric Design, v. 11, n. 4, p. 363-390, 1994.
[3] Farouki, Rida T. et al. Path planning with Pythagorean-hodograph curves for unmanned or autonomous vehicles. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 2017.
[4] Islam, Fahad et al. RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. In: Mechatronics and Automation (ICMA), 2012 International Conference on. IEEE, 2012. p. 1651-1656.
[5] Lavalle, Steven M. Rapidly-exploring random trees: A new tool for path planning. 1998.
[6] Karaman, Sertac; Frazzoli, Emilio. Sampling-based algorithms for optimal motion planning. The international journal of robotics research, v. 30, n. 7, p. 846-894, 2011.
[7] Fabri, Andreas; Pion, Sylvain. CGAL: The computational geometry algorithms library. In: Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems. ACM, 2009. p. 538-539.
[8] Dong, Bohan; Farouki, Rida T. Algorithm 952: PHquintic: A library of basic functions for the construction and analysis of planar quintic Pythagorean-hodograph curves. ACM Transactions on Mathematical Software (TOMS), v. 41, n. 4, p. 28, 2015.
[9] Farouki, Rida T.; Sakkalis, Takis. Pythagorean hodographs. IBM Journal of Research and Development, v. 34, n. 5, p. 736-752, 1990.