Flight Trajectory Path Planning

David Šišlák and Přemysl Volf and Michal Pěchouček
Agent Technology Center, FEE, Czech Technical University in Prague

Abstract

The paper describes an application of the A* algorithm for flight path planning for airplanes with defined motion dynamics operating in a continuous three-dimensional space constrained by existing physical obstacles. The presented A* algorithm modification provides significant acceleration (reduction of the state space) of the path planning process. The described algorithm is able to find a path through small gaps between obstacles using a pre-defined searching precision. The paper documents a set of path planning benchmarks where the solution quality and internal algorithm properties are compared against the A* algorithm for flight trajectory path planning.

Introduction

The paper addresses the area of flight trajectory path planning – domain where airplanes operate in a dynamic continuous three-dimensional space and avoid given obstacles and restricted areas. For airplanes, motion dynamics is defined by means of constraints on the driving manoeuvres and restrictions on smoothness of the trajectory[1]. As its result the path planning algorithm should provide a spacial arrangement of the trajectory and velocity control for each part of the trajectory. The paper addresses only the spatial part[2] of the path planning. The formalization of the spatial path planning is provided in Section [Problem Formalization]. Extended problems like incremental planning are not considered by this paper. The dynamic environment means that obstacles' and restricted areas' definitions are altered almost after each particular search.

The field of the path planning problem has been studied by the research community for many decades. The problem is still topical as all intelligent autonomous vehicles have to include path planning into their deliberation mechanisms. There exist very efficient (fast) algorithms based on randomness, e.g. the random-walk planner (Carpin and Pillonetto 2005b), the rapidly exploring random tree (RRT) (La Valle and Kuffner 2001) and the randomized potential field (Carpin and Pillonetto 2005a) algorithms. Although there are many extensions of these path-finding concepts, their search processes are still based on random sampling and it cannot be said that paths generated by them are optimal with respect to a pre-specified criterion. In many cases, the path require additional smoothing to remove unnecessary random curvature from the path trajectory.

There exist algorithms providing an optimal solution for the given domains with pre-built structures for the given environment definition, e.g. the vector field (Lindemann and La Valle 2005), the potential field (Conner, Rizzi, and Choset 2003), the 3D field D* (Carsten, Ferguson, and Stentz 2006) and the hierarchical path-finding A* (Botea, Müller, and Schaeffer 2004) algorithms. The search process of these algorithms is pretty fast but they require very expensive (especially for large-scale environments) re-building of their pre-built structures after each change in the environment. For the addressed domain it means that structures need to be re-built almost before each search run.

Using the A* algorithm (Hart, Nilsson, and Raphael 1968), there is a trade-off between the search speed (efficiency) and the search precision defined by the pre-specified sampling density of the continuous space (ability to find path through small gaps). There exists an extension of the A* algorithm, the incrementally refined A* (Cormen et al. 2001), which iteratively repeats the search process with increased sampling density. Once the path is found, it cannot be guaranteed that there does not exist a better (shorter) trajectory going through smaller gaps. On the other hand, to check that the path does not exist it first needs to iteratively fail several times.

In the paper, the modification of the A* algorithm called *Accelerated A** (AA*) algorithm applied to flight trajectory planning is presented. The AA* algorithm generates samples by airplane elementary motion actions using elements' adaptive parametrization to reduce the number of states. The size of the sampling varies based on the identified distance to the nearest obstacle – if the current state is far from any obstacle the sampling is sparser, see Figure 1. This approach requires a definition of the minimal sampling size – the *search precision* – to avoid an infinite step reduction towards zero. The AA* algorithm finds a path for a airplalne whose body is bounded by a sphere. The path shown in Fig-

[1]The smoothness means that the path is continuous as well as its first derivation.

[2]The spatial part of the path planning provides the result as a sequence of positions and directions which define only their mutual ordering but do not include the velocity vectors.

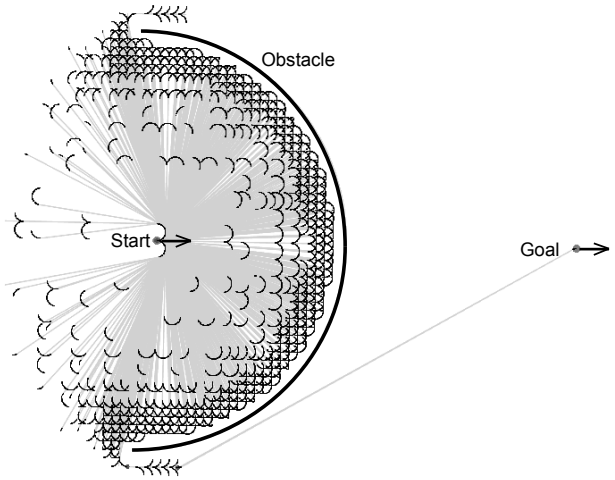

Figure 1: The example of an adaptive sampling in the two-dimensional setup.

ure 1 corresponds to the sphere center.

Problem Formalization

This section provides formalization of the flight path planning in a three-dimensional continuous space for the airplane having constrained dynamics. The airplane is always able to change its horizontal direction only by moving along a horizontal circle with a radius greater than the *minimum horizontal turn radius* r_h. The change of vertical orientation of the airplane is also restricted by moving along a vertically oriented circle with a radius greater than the *minimum vertical turn radius* r_v. The *maximum airplane pitch angle* (max deviation from the horizontal orientation, positive or negative) is denoted as Θ_{max}. The Θ_{max} constrains the allowable parts of the vertical circle for the airplane. Beside these two direction vector changing manoeuvres, the airplane direction can be changed along a spiral which is used for changing vertical position in a confined area. The airplane can apply only the entire loop of the spiral.

Airplane path planning is transformed to motion planning for its reference point called *pivot*. The problem of generating internal control actions, so that pivot moves along the defined path, is not in the scope of the paper. The physical shape of the airplane is bounded by a sphere with a radius r_{bound}. The bounding sphere center is identical to the airplane pivot position $\overline{x}$. The orientation of the airplane is uniquely identified by the direction vector $\overline{v}$. The set of all possible direction vectors is[3]

$$\mathcal{V}_{\Theta_{max}} = \{\overline{v} \in \mathrm{R}^3 : || \overline{v} || = 1, | \arctan \frac{v_z}{|| (v_x, v_y) ||} | \leq \Theta_{max}\}. \quad (1)$$

The airplane operates in a continuous three-dimensional space R^3 where its operation is further restricted by the existing *obstacles* and *operation area boundaries*, both together denoted as $\mathcal{O} \subset \mathrm{R}^3$. Thus the *free space* is defined as $\mathcal{X} = \mathrm{R}^3 \setminus \mathcal{O}$. The ε*-free space* $\mathcal{X}_\varepsilon \subseteq \mathcal{X}$ is defined as

$$\mathcal{X}_\varepsilon = \{\overline{x} \in \mathcal{X} : \forall \overline{o} \in \mathcal{O}, || \overline{x} - \overline{o} || \geq \varepsilon\}, \quad (2)$$

in each ε-free position, the distance to the nearest obstacles is at least ε. Thus, the airplane with its shape bounded by a sphere with a radius r_{bound} has the *airplane operating space* $\mathcal{X}_{r_{bound}}$. The *configuration* c is defined as the tuple $\langle \overline{x}, \overline{v} \rangle$, where $\overline{x}$ is the airplane pivot position and $\overline{v}$ is the direction vector. The **configuration is valid** for the airplane if and only if $\overline{x} \in \mathcal{X}_{r_{bound}}$ and $\overline{v} \in \mathcal{V}_{\Theta_{max}}$. The *start configuration* for the path planning is denoted as c_S and the *goal configuration* as c_G.

The *path* of the airplane is represented as a finite ordered sequence of n elements corresponding to airplane actions $\Phi = \langle e_0, \dots e_{(n-1)} \rangle$ where $e_i \in \mathcal{E}$. The set $\mathcal{E}$ has four construction elements $\mathcal{E} = \{e^S, e^{HT}, e^{VT}, e^{SPIRAL}\}$: *straight* e^S, *horizontal turn* e^{HT}, *vertical turn* e^{VT} and *spiral* e^{SPIRAL} element. The $\mathcal{E}$ reflects the airplane motion constraints defined at the beginning of this section. Each element is defined by the number of parameters fully describing its shape position and orientation. For the $e^S(c_0, l)$, there is an start configuration c_0 and the length l. The $e^{HT}(c_0, r, \alpha, o)$ and $e^{VT}(c_0, r, \alpha, o)$ are defined as an arc of a circle with the radius r, beginning at an start configuration c_0. The arc angle α and the orientation o define whether the horizontal turn is left or right, resp. up or down for the vertical turn. The e^{HT} can be applied only to the c_0 with a zero pitch angle. The $e^{SPIRAL}(c_0, r, n, o)$ is described by the start configuration c_0, horizontal radius r of the spiral, the number of spiral loops $n \in \mathbb{N}^+$ and the spiral orientation o (left or right). The climbing angle is given by the c_0 direction pitch angle. The e^{SPIRAL} provides the same direction vector of the final configuration as its start configuration due the natural number of spiral loops. The spiral element is used when the Θ_{max} is too restricted and the airplane needs to change its vertical position within a limited free space.

There is defined the function $\mathbf{p}(e, t)$ which returns the configuration given by an element e at the position $t \in \langle 0, 1 \rangle$ within the element. The $\mathbf{p}(e, 0)$ returns the start configuration defined by that element and $\mathbf{p}(e, 1)$ returns the final configuration of that element. The function $\mathbf{l}(e)$ returns the Euclidean length of the given element e. The function $\mathbf{u}(e)$ defines if the **element** e **is valid**

$$\mathbf{u}(e) = \begin{cases} 1 \text{ if } \forall t \in \langle 0, 1 \rangle, \ \mathbf{p}(e, t) \text{ is valid} \\ 0 \text{ otherwise} \end{cases} . \quad (3)$$

The **path** Φ **is valid** if and only if for all $e_i \in \Phi$, $\mathbf{u}(e_i) = 1$ and

$$\forall i = 1, \dots n-1 : \ \mathbf{p}(e_i, 0) = \mathbf{p}(e_{i-1}, 1) \ .$$

The valid path is always smooth because all e_i and their connections are smooth as well.

Definition 1 *The* ***path planning*** *for given start c_S and goal c_G configurations is a process searching for the valid path Φ. The search optimization criterion is the length of the path – it searches for a path which is as short as possible. If the path is not found, the planning process returns $\Phi = \emptyset$.*

[3]The $|| \ \overline{v} \ ||$ denotes the Euclidean length of the vector $\overline{v} = \langle v_x, v_y, v_z \rangle$. The (v_x, v_y) is the horizontal part of the vector $\overline{v}$.

The algorithm should use only the turn elements with their minimum radii r_h and r_v because it is shown in (Dubins 1957) that the shortest path between any two configurations is constructed using these turn elements.

AA* Algorithm

This section provides a detailed description of A* modifications for the flight path planning problem as specified in Definition 1. The described modification is called *Accelerated A** (AA*) path planning. First, the AA* algorithm concept is introduced and then the algorithm is described in detail.

AA* Path Planning Concept

The AA* algorithm extends the A* algorithm to be usable in large-scale environments without forgetting about the search precision. The AA* algorithm removes the trade-off between the speed and the precision by introducing the *adaptive sampling*. During the expansion, child states are generated by applying vehicle elementary motion actions using elements' adaptive parametrization. The set of elementary motion actions is defined by the model of the non-holonomic airplane movement dynamics. The adaptive parametrization varies so that the algorithm makes larger steps when the current state is far from obstacles and restricted areas and smaller steps when it is closer.

There is a defined *search precision* l_{min} specifying the minimal sampling grid step which is used in the areas closest to obstacles. The search precision is defined so that the AA* algorithm does not skip any existing gap between obstacles larger than this precision. The adaptive parametrization sampling uses only variants which correspond to sampling sizes equal to the precision to the power of two. Specifically, the AA* algorithm uses the highest possible parametrization which ensures that the distance to the closest obstacle is not smaller than the distance corresponding to two respective sampling steps.

The adaptive sampling in the AA* algorithm requires a different definition of identity tests when working with $OPEN$ and $CLOSED$ lists. The original equality implementation is replaced by a similarity check. Two states are similar if their Euclidean distance and their direction vector variation is less than a threshold derived from the respective sampling parametrization. Otherwise, the adaptive sampling of a non-holonomic airplane trajectory causes an infinite state generation in the continuous space. To remove effects of varying sampling, each path candidate generated during the search is smoothed.

The search uses the *heuristics* which is computed as the length of the shortest valid path from the current state to the goal configuration without restriction on airplane operation space $\mathcal{X}_{r_{bound}}$. This construction of heuristics avoids the expansion of the states which are very close to the goal configuration but the path to goal is much longer than the distance to the goal. Each state has an attribute marking if such shortest path from this state to the goal is going outside the airplane operating space $\mathcal{X}_{r_{bound}}$ or not. When the best candidate is picked from the $OPEN$ list and the attribute indicates that the shortest path to the goal is valid, the algorithm ends the search because it found the path planning solution – the path from the start state to the current one plus this shortest path to the goal.

AA* Description

The pseudocode of the AA* algorithm is provided in Algorithm 1. Initially, validity of the given start c_S and goal c_G configurations is checked (lines 1 and 2). If any of them is not valid, the algorithm fails. Next, the start state is constructed (line 3). The state of the algorithm is defined as a tuple $\langle c, \xi, t, s, g, h, pred \rangle$. The $c = \langle \overline{x}, \overline{v} \rangle$ is the configuration of the state (position and direction). The $\xi = \langle l, \alpha_h, \alpha_v \rangle$ is the sampling parametrization returned by the function `GetSamplingParams` (see Section). The $t \in \{START, STRAIGHT, HTURN, VTURN, SPIRAL, CONNECTION\}$ is the identification of the element used to transit from the previous state configuration to the current one. The t is given by the function `TypeOf` in the algorithm. The $CONNECTION$ is used for the complex element constructed by the `Connect` function, see Section . The $s \in \{true, false\}$ is the attribute presenting the validity of the shortest connection to the goal state. The g is the overall length of the path from the start configuration to the current one. The h holds the heuristics – length of the shortest path to the goal regardless of obstacles. The $pred$ is the reference to the preceding state which is used for path reconstruction.

At lines 4 and 5, the $OPEN$ and $CLOSED$ lists are initialized. The $OPEN$ list contains candidate states and $CLOSED$ list contains the already expanded configurations. The $OPEN$ list is initialized with the start state. The $OPEN$ list is implemented as a priority queue in combination with a modified hash set (Cormen et al. 2001). The modification of the classical hash set supports fast checking of the state similarity test (defined in Section) using the given state sampling parametrization ξ. The main search loop (lines 6–26) repeats until $OPEN$ list is empty. If $OPEN$ list is empty, the search fails (line 27).

During the search, the algorithm removes the best candidate (a state with the minimum value of $g + h$) from the $OPEN$ list, function `RemoveTheBest`, and inserts its configuration to the $CLOSED$ list, function `Insert into` (lines 7 and 8). At line 9, the algorithm checks whether the shortest connection to the goal provides a valid path by reading s part from the state. If it is valid (lines 10–13), the path represented by the previous states and connection to the goal is the final solution of the search. The function `SmoothPath` does the path smoothing, see Section . The path reconstruction at line 12 does the reverse backtracking via $pred$ reference in the state and prepares the final search result Φ using the t attribute from the state.

At lines 15–25, the algorithm iterates over all proposed expansion elements from the function `Expand`, see Section . Initially, it reads the element's final configuration c^{new} and identifies its sampling parametrization ξ^{new} (lines 16 and 17). Then, it skips the rest of the cycle if a similar configuration has been already visited (lines 18 and 19) or the proposed element is not valid (line 20). The function

Input: c_S, c_G
Output: Φ

{1} **if** `not IsValid(`c_S`)` **or** `not IsValid(`c_G`)` **then**
{2} **return** $\emptyset$;
{3} $s_I \leftarrow \langle c_S, \texttt{GetSamplingParams}(c_S), \text{START}, false, 0, 0, - \rangle$;
{4} $OPEN \leftarrow \{s_I\}$;
{5} $CLOSED \leftarrow \emptyset$;
{6} **while** $OPEN \neq \emptyset$ **do**
{7} cur $\leftarrow$ `RemoveTheBest(`*OPEN*`)`;
{8} `Insert` c^{cur} `into` $CLOSED$;
{9} **if** s^{cur} **then**
{10} $cur \leftarrow \langle c_G, -, \text{CONNECTION}, true, g^{cur} + h^{cur}, 0, cur \rangle$;
{11} $cur \leftarrow$ `SmoothPath(`*cur*`)`;
{12} $\Phi \leftarrow$ `ReconstructPath(`*cur*`)`;
{13} **return** Φ;
{14} **end**
{15} **foreach** $e_i \in$ `Expand(`*cur*`)` **do**
{16} $c^{new} \leftarrow \mathbf{p}(e_i, 1)$;
{17} $\xi^{new} \leftarrow$ `GetSamplingParams(`c^{new}`)`;
{18} **if** `Contains(`c^{new}, *CLOSED*, ξ^{new}`)` **then**
{19} **continue**;
{20} **if** `not IsValid(`e_i`)` **then continue**;
{21} $e^{end} \leftarrow$ `Connect(`c^{new}, c_G`)`;
{22} $new \leftarrow \langle c^{new}, \xi^{new}, \texttt{TypeOf}(e_i), \texttt{IsValid}(e^{end}), g^{cur} + \mathbf{l}(e_i), \mathbf{l}(e^{end}), cur \rangle$;
{23} $new \leftarrow$ `SmoothPath(`*new*`)`;
{24} `InsertOrReplaceIfBetter(`*new, OPEN,* ξ^{new}`)`;
{25} **end**
{26} **end**
{27} **return** $\emptyset$;

Algorithm 1: **The AA* algorithm pseudocode**

`Contains` includes a similarity check using the given sampling parametrization ξ^{new}. The function `IsValid` checks the validity of the element. The `IsValid` returns $true$ if and only if $\mathbf{u}(e) = 1$, see Equation 3. Internally, the function `IsValid` is implemented as a geometrical interpolation of the element which is further checked against the representation of $\mathcal{X}_{r_{bound}}$ and $\mathcal{V}_{\Theta_{max}}$, providing an efficient test of point and line validity. The detailed description of their representation is not within the scope of this paper.

At lines 21–24, the new state is prepared and the path to new state is smoothed. The heuristics for the new state is computed as the shortest length to the goal using the `Connect` function, see Section . Finally, the newly expanded state is inserted into $OPEN$ list if it does not contain any similar state yet. Alternatively, the existing similar state in the $OPEN$ list is replaced if the sum of values of g and h for new state is lower than the sum of the existing one (the function `InsertOrReplaceIfBetter` at line 24).

Get Sampling Parameters Function

For the input configuration c, the function `GetSamplingParams` prepares the sampling parametrization tuple $\xi = \langle l, \alpha_h, \alpha_v \rangle$: l is the sampling length for the e^S, α_h is the horizontal turn sampling angle and α_v is the vertical turn sampling angle. Initially, the sampling level a is determined using the desired search algorithm *precision* l_{min} (see Section)

$$\begin{aligned} &a = \max(n \in \mathbb{N}_0 : \ \overline{x}^c \in \mathcal{X}_{\varepsilon(n)}), \\ &\text{where } \varepsilon(n) = 2^{(n+1)} l_{min} + r_{bound} \ . \end{aligned} \tag{4}$$

The l is then computed as $l = 2^a l_{min}$. This construction of l provides state density gradually increasing towards obstacles which guarantees that the search is able to find any existing gap larger than l_{min} in the airplane operating space $\mathcal{X}_{r_{bound}}$.

The α_h and α_v are chosen so that the Euclidean distance of their start and final arc points is l and is limited to $\frac{\pi}{2}$ for α_h and Θ_{max} for α_v. Turn radii are always set to their minimums during the search (see Section), so that

$$\alpha_h = \begin{cases} \frac{\pi}{2} & \text{if } l \geq r_h\sqrt{2} \\ \arccos\left(\frac{2r_h^2 - l^2}{2r_h^2}\right) & \text{otherwise} \end{cases} ,$$

$$\alpha_v = \begin{cases} \Theta_{max} & \text{if } l \geq r_v\sqrt{2(1-\cos\Theta_{max})} \\ \arccos\left(\frac{2r_v^2 - l^2}{2r_v^2}\right) & \text{otherwise} \end{cases} . \tag{5}$$

Connect Function

The function `Connect` constructs a complex element e^C composed as a sequence of basic geometrical elements $e_i \in \mathcal{E}$ connecting the two given configurations c_1 and c_2 by the shortest path. The provided e^C fulfills the same constraints as the path Φ except for the obstacle intersection criterion given by Equation 3. The construction of the shortest path uses only the airplane motion constraints r_h, r_v and Θ_{max}. The intersection criterion is not included in the path composition as it is used for counting the best admissible heuristics for the search algorithm. However, the functions $\mathbf{p}(e, t)$, $\mathbf{l}(e)$, `TypeOf` and `IsValid` are extended to work properly with the complex element e^C.

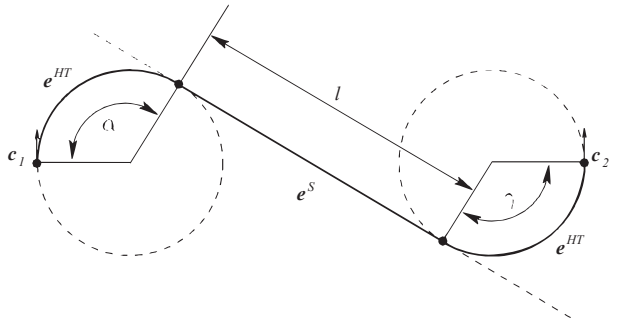

Figure 2: The shortest path example between two configurations c_1 and c_2.

A two-dimensional example of one such connection is in Figure 2. The e^C in the example is composed of three basic elements: *right horizontal turn*, *straight* and *left horizontal turn*. The shortest path sequence is always constructed using the *turn* and *spiral* elements with the radius set to the minimum available for the airplane (Souères and Boissonnat 1998). The problem of finding the shortest connection is transformed to the problem of sequence composition identification and the computation of elements parameters.

The two-dimensional problem is referred to as *Dubins curves*. In (Souères and Boissonnat 1998), it is shown how the sequence can be identified. The function `Connect` is an extension of Dubins curves to a three-dimensional domain using also vertical turn and spiral elements still providing the shortest connection.

Smooth Path Function

The `SmoothPath` function tries to remove unnecessary middle states in the path from the start configuration to the current state. This tries to make the path shorter than its original version.

Input: inState
Output: outState

{28} $\Gamma \leftarrow$ `ExtractPrevStates(`*inState*`)`;
{29} **foreach** $s \in \Gamma$ **do**
{30} $\quad e^C \leftarrow$ `Connect(`c^s, $c^{inState}$`)` ;
{31} $\quad$ **if** $\mathbf{l}(e^C) = (g^{inState} - g^s)$ **then**
{32} $\quad\quad$ **return** *inState*;
{33} $\quad$ **if** `IsValid(`e_s`)` **then**
{34} $\quad\quad$ $outState \leftarrow \langle c^{inState}, \xi^{inState},$ CONNECTION, $g^s + \mathbf{l}(e^C), h^{inState}, s\rangle$;
{35} $\quad\quad$ **return** *outState*;
{36} $\quad$ **end**
{37} **end**
{38} **return** *inState*;

Algorithm 2: **The function `SmoothPath`**

The function `SmoothPath`, see Algorithm 2, tries to find the shortest valid replacement for the current path by applying the `Connect` function to as many parts of the path as possible[4]. At line 1, the list Γ is filled with states from the start search state to the previous state of the $inState$. Next, the function tries to replace the path from s to the input state $inState$ with the shortest connection given by the function `Connect` (lines 29–37). If the length of the shortest connection is the same as the existing segment length, the path cannot be made shorter (also any subsequent part cannot be made shorter) and the function returns the unchanged path. At lines 33–36, the function checks whether the shorter connection e^C is valid. If there is no intersection with any obstacle, the function replaces the rest of the path with this shortest connection and the function returns the updated path with a newly prepared $outState$.

[4]Only the path parts to the current state are replaced because this function is already called to every previous state in the path (Algorithm 1, line 23).

Expand Function

The pseudocode of the function `Expand` is stated in Algorithm 3. The function returns a set of elements $\mathcal{E}$ extending the given $state$. First, it extracts the sampling parametrization component from the state and computes pitch angle Θ (lines 38 and 39). The Θ represents the angle of the state direction vector from the horizontal plane. It is positive if the direction vector heads upwards from the horizontal plane and negative if downwards. At line 40, the set $\mathcal{E}$ is initialized with the straight element from the state configuration with its length taken from the current sampling parametrization.

Input: state
Output: $\mathcal{E}$

{39} $len, \alpha_h, \alpha_v \leftarrow \xi^{state}$;
{40} $\Theta \leftarrow$ `PitchAngle(`c^{state}`)`;
{41} $\mathcal{E} \leftarrow \{e^S(c^{state}, len)\}$;
{42} **if** $\Theta = 0$ **then**
{43} $\quad \mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{HT}(c^{state}, r_h, \alpha_h, LEFT)\}$;
{44} $\quad \mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{HT}(c^{state}, r_h, \alpha_h, RIGHT)\}$;
{45} **else**
{46} $\quad \mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{SPIRAL}(c^{state}, r_h, 1, LEFT)\}$;
{47} $\quad \mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{SPIRAL}(c^{state}, r_h, 1, RIGHT)\}$;
{48} $\quad$ **if** $\Theta > 0$ **then**
{49} $\quad\quad \mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{VT}(c^{state}, r_v, \Theta, DOWN)\}$;
{50} $\quad$ **else**
{51} $\quad\quad \mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{VT}(c^{state}, r_v, -\Theta, UP)\}$;
{52} $\quad$ **end**
{53} **end**
{54} $\mathcal{E} \leftarrow \mathcal{E} \bigcup \{e^{VT}(c^{state}, r_v, \min(\alpha_v, \Theta_{max} - \Theta), UP)\}$;
{55} $\mathcal{E} \leftarrow$
$\mathcal{E} \bigcup \{e^{VT}(c^{state}, r_v, \min(\alpha_v, \Theta_{max} + \Theta), DOWN)\}$;
{56} **return** $\mathcal{E}$;

Algorithm 3: **The function `Expand`**

Horizontal turn elements e^{HT} with right and left direction are added only if the airplane is not climbing or descending (lines 42–45). This restriction comes from the problem definition (Section). Otherwise, spiral elements e^{SPIRAL} with both orientations are added (lines 43 and 44). At lines 48–52, the `Expand` appends an element which re-aligns the airplane direction with the horizontal plane. This correction is necessary as the application of e^{VT} with appropriate sampling parametrization can cause that the horizontal direction is not reached in many following states. Finally, vertical turn elements bounded to the airplane maximum pitch Θ_{max} are added (lines 54 and 55). The `Expand` uses the turn elements with minimum turns allowed for the airplane r_h and r_v because it is shown in (Dubins 1957) that the shortest path between any two configurations can use only these turn elements.

Experiments

Properties of the AA* concept were evaluated on a set of two and three-dimensional setups, Figure 3. The A* algorithm

with a fixed sampling size and a distance-to-target heuristics was chosen as a comparator. The fixed sampling size in the A* algorithm is equal to the *search precision* l_{min} of AA* as it has to be able to find a path through the same smallest gap between any two obstacles. In all experiments, the path cost is constructed by the length of the path[5] and the admissible heuristics is computed as the length of the shortest connection of the current state configuration and the goal including their vectors respecting the airplane motion constraints. The size of testing environments was selected at the maximum which is computable by the A* algorithm within one hour on the standard 2.5 GHz desktop computer with 8 GB of memory.

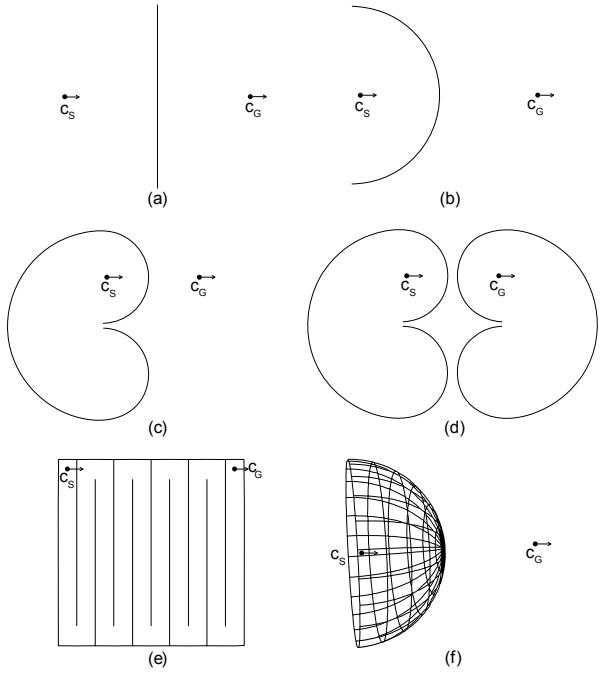

Figure 3: Experiment setups: (a) wall, (b) half circle, (c) single gap, (d) double gap, (e) maze and (f) half sphere.

The parameters of the airplane are the same in all setups: (i) bounding sphere radius r_{bound} of the airplane body is 10; (ii) minimum horizontal r_h and vertical r_v turn radii are 10; and (iii) maximum pitch angle Θ is $\frac{\pi}{6}$. Several testing two and three-dimensional setups have been selected (see Figure 3): *wall*, *half circle*, *single* and *double gaps*, *maze* and *half sphere*. The start configuration is denoted as c_S and the goal as c_G. Three-dimensional versions of the algorithms are reduced to two-dimensional versions by using only elements e^S and e^{HT} in the `Expand` function.

In the *wall*, *half circle*, *single gap*, *double gap*, and *half sphere* setups the distance between c_S and c_G is 500, in the *maze* setup it is 900, the length of the maze square side is 1000 and the width of each tunnel is almost 100. First two configurations, *wall* and *half circle*, have obstacles positioned so that they are causing deviation from paths which are strongly preferred by the used heuristics – the shortest path from the current configuration to the goal. The next two setups, *single* and *double gap*, demonstrate the ability of the algorithm to find a small hole in the obstacle to find a path to the outside point or through the next gap to another one. The gap size in these two setups is 23 which means that the gap width available to the path search algorithm is no more than 3 because of the airplane bounding sphere $r_{bound} = 10$. The detail of the path through the gap with the visualization of the airplane dimension is shown in Figure 4 – expanded paths cannot go closer than the bounding radius to the obstacle.

The *maze* configuration verifies the acceleration capabilities of the AA* algorithm in the case where the selected heuristics is completely inefficient. To verify the benefits of AA* also in the three-dimensional domain, the *half sphere* scenario has been included. The start path point c_S is inserted in the sphere to let the algorithm identify that it has to go slightly backward first to find the correct path. Only this one three-dimensional setup is documented in the experiments because more complicated setups are not feasible for the A* algorithm without large increase of sampling size. However, the example of the path search by AA* for complex three-dimensional scene is provided in Figure 5.

Table 1 summarizes the measured properties of both A* and AA* algorithms in all these six setups. The time necessary for finding the path from start to goal configuration is expressed as a ratio to the duration of the A* algorithm in the same configuration and on the same computer. The table displays (*i*) the length of the final path provided by both approaches, (*ii*) the number of states processed from the open list, (*iii*) the number of all expanded states, (*iv*) the number of expanded states which are applicable (the path to them does not intersect with any obstacle and it lies in the vehicle operating space as well) and (*v*) the number of smoothing operations applied to the path are documented in the table.

The *search precision* l_{min} for the first five setups is set to 1.25 which guarantees that the algorithm should be able to find a path through the small gaps in the gap setups, see the detail in Figure 4. In the three-dimensional setup, the *search precision* is set to 10 which is the smallest sampling for which the A* algorithm finds the path due to the memory limits of the used computer.

In the first four setups, the AA* algorithm significantly reduces the number of expanded states and thus reduces the number of generated states more than fifty times in the *double gap* setup and up to more than two hundred times in the *wall* configuration. On the other hand, the quality of the path is no worse than a half percent in comparison to the best one which can be found for the given search precision. The number of elements in the path is almost the same for both algorithms. The speed-up effect is more significant than the reduction of states. In the first four configurations, the AA* algorithm is faster more than one hundred and seventy times and up to more than one thousand and four hundred times in the *wall* setup. This is caused by the fact that the $OPEN$ and $CLOSED$ lists operation for finding of the similar state is much more efficient if the hash table structures work with less members – e.g. in the *double gap* setup there are almost

[5]The path length is computed as Euclidean length in $\mathbb{R}^3$.

Experiment Setup			SD	PL	ES	GS	US	AS
Setup	Algorithm	Search precision						
Wall	A*	1,25	1	731,20 (3)	335 128	763 631	756 042	483 619
	AA*	1,25	0,000 69	733,87 (4)	1 282	3 217	2 590	1 660
Half circle	A*	1,25	1	879,14 (3)	360 999	819 579	807 576	483 994
	AA*	1,25	0,001 95	879,30 (3)	3 767	10 136	7 023	4 779
Single gap	A*	1,25	1	879,14 (3)	359 878	816 645	804 712	481 795
	AA*	1,25	0,002 48	879,30 (3)	3 581	9 607	6 561	4 387
Double gap	A*	1,25	1	1 117,54 (3)	801 752	1 849 055	1 805 778	1 016 483
	AA*	1,25	0,005 911	1 118,09 (3)	12 876	34 733	23 291	14 073
Maze	A*	1,25	1	8 557,30 (19)	1 737 858	4 080 222	3 813 707	1 046 204
	AA*	1,25	0,039 85	8 591,56 (20)	111 525	292 931	192 574	42 258
Half sphere	A*	10	1	880,33 (3)	340 922	1 260 113	1 212 951	474 603
	AA*	10	0,182 51	880,90 (3)	75 440	292 209	248 339	98 062

Table 1: Experiments results: SD - search time duration in ratio to A* run; PL - path length, in brackets is the number of elements in the path; ES - number of expanded states; GS - number of generated states; US - number of usable states (reachable without any intersection) after expansion; AS - number of applied smoothing operations to the paths.

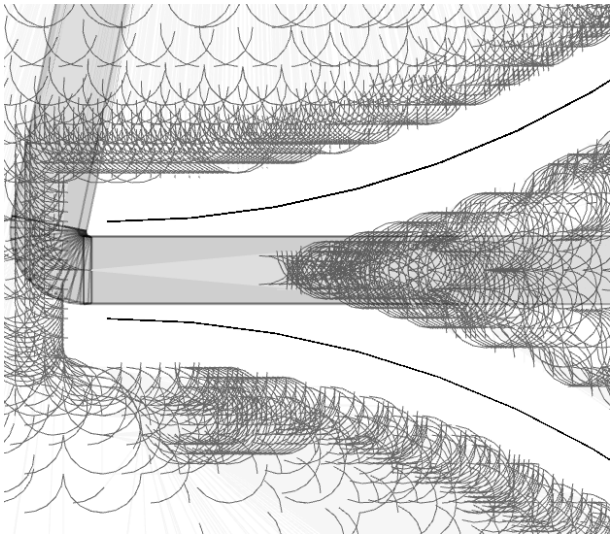

Figure 4: The detail of the searched paths around the first gap in the double bug scenario. The highlighted path represents the final path with $r_{bound} = 10$. The free space between the obstacle and the final path in the gap is given by the search precision $l_{min} = 1.5$.

two million usable states for the A* algorithm. Moreover, the higher number of states causes several times higher load on the `IsValid` test as each element can be approximated by many lines.

In the *maze* setup (Figure 3 e), the AA* concept efficiency has been proved although the heuristics does not help during the search progress in the five tunnels going down – away from the goal configuration. In this configuration, the AA* algorithm found the solution twenty five times faster than A* and reduced the number of usable states from almost four million to two hundred thousand only. In the three-dimensional setup, the AA* algorithm accelerates the search only five times which is caused by the fact that the sampling size is too large relatively to the size of the scenario. So the hierarchical adaptive sampling parametrization cannot provide greater reduction in the AA* algorithm. But it was impossible to measure the configuration for smaller basic sampling size because the number of states for the A* algorithm grows very quickly in the three-dimensional case.

Another positive effect of the states reduction in the AA* algorithm is the reduction of the memory requirements. These memory savings allow the AA* algorithm to find the path also in the configuration where it is impossible with the A* algorithm. One such example is shown in Figure 5.

Conclusion

The modification of the A* algorithm for flight path planning, called *Accelerated A** (AA*), has been introduced in the paper. The AA* algorithm utilizes the concept of the dynamic adaptive sampling used during a single run of the search. The benefits of this approach has been documented in several path planning benchmarks for the nonzero-sized airplane with defined dynamics, where the AA* algorithm proved its capability to significantly reduce the number of visited states in the search. The AA* algorithm provides the acceleration (up to more than 1400 times) of the path planning not only for the positive cases (where the path is found), but in the same way for the negative cases (path doesn't exist) as the adaptive sampling reduces the number of generated states in the same setup.

Such state reduction implies the reduction of the required memory during the search which makes AA* suitable also for the complex configuration space in the huge continuous space, see Figure 5. On the other hand, the quality of the path is almost the same as for the A* algorithm using the same search precision. For the given sampling precision, the AA* algorithm is still able to find the path through the small gaps between obstacles and the feasibility of the path planning is not affected by the acceleration.

The extension of the AA* algorithm is used as a trajectory planner for the experimental simulation platform AGENTFLY. The AGENTFLY supports simulation of multiple UAVs based on the free flight concept with replanning and collision avoidance abilities. The AA* is extended by a temporal planner used to fulfil time and speed restrictions for the trajectory. The temporal phase of the planning assigns acceleration and deceleration to straight elements. If the trajectory needs to be changed, a backward call of the AA* is performed.

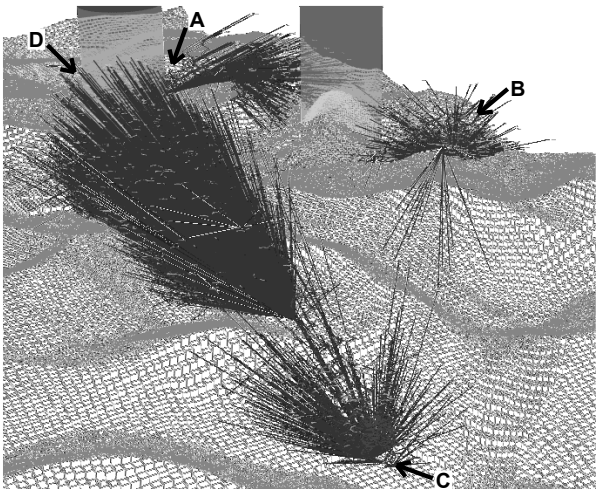

Figure 5: Complex three-dimensional path planing with many obstacles in three planning segments: A→B, B→C and C→D.

The described AA* algorithm utilizes the single-tree method. Another speed-up of the search process can be brought by its extension to the bidirectional search (Pohl 1971). In such an extension, the algorithm will search in both directions concurrently from the start configuration to the goal and vice versa. The bidirectional search has been successfully applied to the RRT search (Kuffner and La Valle 2005). Its application to the AA* algorithm is more complicated because of the non-trivial mapping from the forward tree to the backward tree. The algorithm searches in the continuous space and the searched states from two different start configurations are completely different because the states are generated by the elementary moves with an adaptive size.

Acknowledgement

The AGENTFLY is supported by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3073 and by Czech Ministry of Education grant number 6840770038.

References

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1(1):7–28.

Carpin, S., and Pillonetto, G. 2005a. Merging the adaptive random walks planner with the randomized potential field planner. In *Proceedings of IEEE International Workshop on Robot Motion and Control*, 151–156.

Carpin, S., and Pillonetto, G. 2005b. Robot motion planning using adaptive random walks. *IEEE Transactions on Robotics & Automation* 21(1):129–136.

Carsten, J.; Ferguson, D.; and Stentz, A. 2006. 3D Field D*: Improved Path Planning and Replanning in Three Dimensions. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3381–3386.

Conner, D. C. C.; Rizzi, A. A.; and Choset, H. 2003. Composition of local potential functions for global robot control and navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms (2nd Edition)*. Cambridge, MA: MIT Press and McGraw-Hill.

Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* (79):497–516.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* (2):100–107.

Kuffner, J. J., and La Valle, S. M. 2005. An efficient approach to path palnning using balanced bidirectional rrt search. Technical Report Technical Report CMU-RI-TR-05-34, Robotics Institute, Carnegie Mellon University, Pittsburgh.

La Valle, S. M., and Kuffner, J. J. 2001. Rapidly exploring random trees: Progress and prospects. In Donald, B. R.; Lynch, K. M.; and Rus, D., eds., *Algorithmic and Computational Robitics: New Directions*, 293–308. MA, USA: A K Peters, Wellesley.

Lindemann, S. R., and La Valle, S. M. 2005. Smoothly blending vector fields for global robot navigation. In *Proceedings of IEEE Conference Decision & Control*.

Pohl, I. 1971. Bi-directional search. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, 127–140. New York: Elsevier.

Souères, P., and Boissonnat, J.-D. 1998. Optimal trajectories for nonholonomic mobile robots. In Laumond, J.-P., ed., *Robot Motion Planning and Control*, 93–169. Berlin: Springer-Verlag.