

Iterative Accelerated A* Path Planning

Štěpán Kopřiva, David Šišlák, Dušan Pavlíček and Michal Pěchouček

Abstract—The paper provides a description of an iterative version of the Accelerated A* algorithm for path planning and its application in the air traffic domain for airplanes with defined motion dynamics operating in the Earth-centered, Earth-fixed coordinate system (GPS) on a spherical model of the Earth constrained by the landscape and special use airspaces (SUA). The motion dynamics of the airplanes is modeled using the Base of Aircraft Data (BADA) airplane performance models. The presented algorithm provides an extension of the A* algorithm that significantly reduces the search space and makes planning of the flight trajectories computationally tractable.

I. INTRODUCTION

This paper addresses the problem of trajectory planning for vehicles operating in dynamic continuous three-dimensional space. The vehicle is modeled using a non-holonomic model, the motion dynamics is restricted by constraints of the driving maneuvers and restrictions on the smoothness of the trajectory. In general, the planning problem is a 4-D problem – the agent needs to plan in a 3-D space and time. However, this paper addresses the path planning problem without the time component.

In the research community there are several well known approaches to the trajectory planning problem. These approaches may be divided into two basic groups – algorithms optimizing the speed of the solution and algorithms optimizing the quality of the solution in terms of the length of the final plan. The first group uses random techniques to find the plan – the random walk planner [1], the rapidly exploring random tree (RRT) [2] and the randomized potential field [3], [4]. The other group of the planners is driven by an optimality criterion. The efficiency of these planners is limited by changes of the environment definitions, varying initial and goal positions and directions between particular planner invocations. The vector field [5] and potential field [6] methods require very expensive rebuilding of the fields when the goal position or space definition is changed.

The A* algorithm uses a heuristic-informed search to find a plan if one exists in the given state space. The continuous space needs to be discretized in order to be searched by the discrete algorithm. The discretization step is crucial for the algorithm's performance – the larger the step, the faster and less optimal the search is. Also note that the choice of the step size may be domain dependent.

We present the *Iterative Accelerated A** (IAA*) algorithm for trajectory planning in Section 3. This algorithm is an extension of the *Accelerated A** (AA*) algorithm [7]. The original AA* uses a variable discretization step size to reduce

The work was supported by the Federal Aviation Administration (FAA) under project number DTFAC08-C-00033 and by Czech Ministry of Education under grant number 6840770038.

All authors are with Agent Technology Center, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, 166 27 Prague, Czech Republic, <http://agents.fel.cvut.cz>.

the number of states in the state space and yet keep the ability to plan precisely. The specific domain in which we plan is the National Airspace System (NAS) of the USA. There are almost 1000 irregular special use airspaces and the Accelerated A* algorithm is not fast enough.

II. PROBLEM FORMALIZATION

There exist numerous versions of the path planning problem [8]. This section provides formalization of the path planning in a three-dimensional continuous space for the vehicle having constrained dynamics. The vehicle is 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 the vertical orientation of the vehicle is also restricted by moving along a vertically oriented circle with a radius greater than the *minimum vertical turn radius* r_v . The *maximum vehicle pitch angle* (max deviation from the horizontal orientation, positive or negative) is denoted as Θ_{max} . The Θ_{max} constraints the allowable sections of the vertical circle for the vehicle. Besides these two direction vector changing maneuvers, the vehicle direction can be changed along a spiral which is used for changing the vertical position in a confined area. The vehicle can apply only an entire loop of the spiral.

Vehicle path planning is transformed to motion planning for its reference point called *pivot*. The problem of generating internal control actions so that the pivot moves along a defined path is not in the scope of the paper. The physical shape of the vehicle is bounded by a sphere with a radius r_{bound} . The bounding sphere center is identical to the vehicle pivot position \bar{x} . The orientation of the vehicle is uniquely identified by the direction vector \bar{v} . The set of all possible direction vectors is¹

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

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

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

in each ε -free position, the distance to the nearest obstacles is at least ε . The vehicle with its shape bounded by a sphere with a radius r_{bound} has the *vehicle operating space* $\mathcal{X}_{r_{bound}}$.

An obstacle \mathcal{O} is defined by its geometric shape \mathcal{O}_G and a time interval \mathcal{O}_T during which the obstacle is active:

$$\mathcal{O} = (\mathcal{O}_G, \mathcal{O}_T).$$

¹The $\|\bar{v}\|$ denotes the Euclidean length of the vector $\bar{v} = \langle v_x, v_y, v_z \rangle$. The (v_x, v_y) is the horizontal part of the vector \bar{v} .

The geometric shape \mathcal{O}_G of an obstacle \mathcal{O} is based on a convex spherical polygon SP , specified by a list of vertices $V = \{v_1, \dots, v_n\}$ such that $v_i = (lon_i, lat_i)$ where lon_i denotes longitude and lat_i denotes latitude. \mathcal{O}_G is furthermore defined by the minimal and maximal altitudes alt_{min} and alt_{max} corresponding to the lower and upper altitudinal extents of the obstacle \mathcal{O} :

$$\mathcal{O}_G = (V, alt_{min}, alt_{max}).$$

Geometrically, \mathcal{O}_G can be expressed as

$$\mathcal{O}_G = \{(lon, lat, alt) | (lon, lat) \in SP, alt_{min} \leq alt \leq alt_{max}\},$$

where lon denotes longitude, lat denotes latitude, and alt denotes altitude in the spherical coordinate system (see Figure 1).

The temporal component \mathcal{O}_T of the definition of \mathcal{O} is

$$\mathcal{O}_T = (t_{from}, t_{to}),$$

specifying the time interval during which \mathcal{O} is active. Outside this interval, \mathcal{O} is ignored by intersection tests (discussed in Section III-E). Note that the intersection tests only take into consideration the "vertical" (side) planes of the obstacles referred to as *faces*.

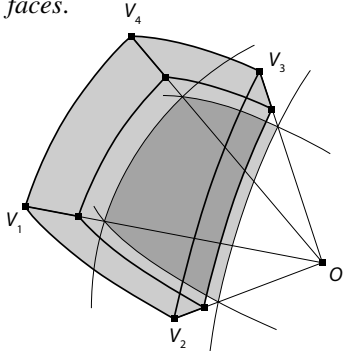


Fig. 1. Geometric definition of an obstacle.

The *configuration* c is defined as a tuple $\langle \bar{x}, \bar{v} \rangle$, where \bar{x} is the vehicle pivot position and \bar{v} is the direction vector. The **configuration is valid** for the entity if and only if $\bar{x} \in \mathcal{X}_{r_{bound}}$ and $\bar{v} \in \mathcal{V}_{\Theta_{max}}$. The *initial configuration* is denoted as c_I and the *goal configuration* as c_G .

The *path* of the vehicle is represented as a finite ordered sequence of n geometrical elements $\Phi = \langle e_0, \dots, e_{(n-1)} \rangle$ 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 vehicle motion constraints defined at the beginning of this section. These constraints are for the most of the current commercial airplanes defined in the Base of Aircraft Data (BADA), see [9]. Each element is defined by a number of parameters fully describing its shape, position and orientation. For the $e^S(c_0, l)$, there is an initial 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 initial 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 c_0 with a zero pitch angle. The $e^{SPIRAL}(c_0, r, n, o)$ is described by the initial configuration c_0 , horizontal radius r of the spiral, the number of spiral loops $n \in \mathbb{N}^+$ and the orientation o of the spiral (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 to the natural number of spiral loops. The spiral element is used when the Θ_{max} is too restricted and the vehicle needs to change its vertical position within a limited free space.

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 initial 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

$$\forall i = 1, \dots, n-1 : \mathbf{p}(e_i, 0) = \mathbf{p}(e_{i-1}, 1) \quad (4)$$

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

Definition 1: The **path planning** for the given initial c_I and goal c_G configurations is a process searching for a 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 $\Phi = \emptyset$. The algorithm should only use turn elements with their minimum radii r_h and r_v because it is shown in [10] that the shortest path between any two configurations is constructed using such turn elements.

III. ITERATIVE AA* ALGORITHM

IAA* is an extension of AA*. IAA* includes several significant improvements, introduced in order to make the computation of the path tractable in reasonable time. The main improvement is the iterative version of the algorithm considering only a limited subset of obstacles in individual iterations.

A. Original AA* Algorithm

The AA* algorithm [7] extends the original A* algorithm to be usable in large-scale environments and provides fast planning while preserving the optimization criterion and precision. The AA* removes the trade-off between the speed and the precision by introducing 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 a model of non-holonomic vehicle 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 makes smaller steps when it is closer, see Figure 2.

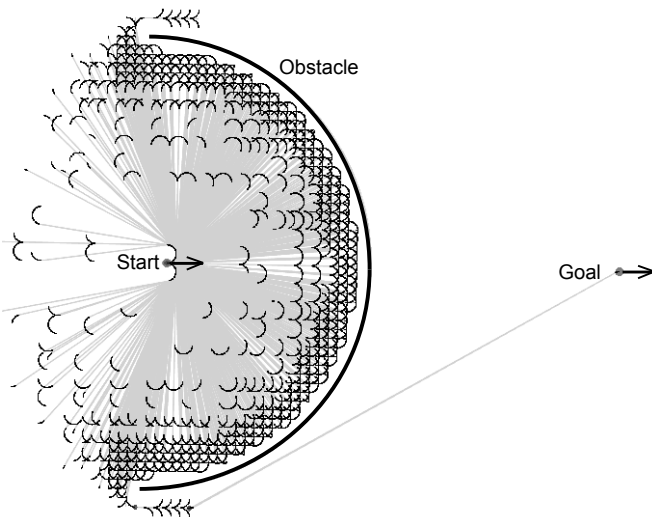


Fig. 2. The adaptive sampling example in the two-dimensional setup.

There is a defined search precision specifying a minimal sampling grid step which is used in the areas closest to obstacles. The search precision is defined in such a way 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 raised 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 CLOSE 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 vehicle 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.

AA* significantly reduces the number of samples (states) generated during the search in large-scale environments and it does not decrease the quality of the solution.

B. Iterative AA* Concept

IAA* extends AA* in order to be usable in large-scale domains with a high number of complex obstacles. The algorithm pushes the limits of fast precise path planning further by running the planning process from the *initial configuration* c_I to the *goal configuration* c_G with a very limited subset of obstacles in the first run. The obstacles considered in the first run of the planning process are specified by the function `DeriveTestedSubset`. This relaxation significantly reduces the number of tests of intersections of newly generated states with the obstacles and thus saves a significant amount of computational time. If the planning with the reduced subset of obstacles is not successful and

the plan intersects any obstacle, this obstacle is added to the tested set and the planning algorithm is run again.

C. Iterative AA* Algorithm

The function `IterativeAA*` has four inputs - the initial configuration c_I , the goal configuration c_G , the set of all obstacles in the domain \mathcal{O} and the parameter α defining the ratio of the Inclusion Space (IS) and the distance from c_I to c_G . The output of the method is the plan Φ .

Inclusion Space is a 3D space constructed as a set of points where for each point exists a point on a great circle connecting c_I and c_G such that the distance of these points is lower or equal to $r_{inclusion}$, see Figure 3. Formally, IS is defined as

$$\forall x \in IS \exists y \in |c_I c_G|; \text{sphere}(x, y) \leq r_{sur}; x, y \in R^3,$$

where $|c_I c_G|$ denotes a great circle with end points c_I and c_G and $\text{sphere}(x, y)$ denotes the spherical distance between points x and y .

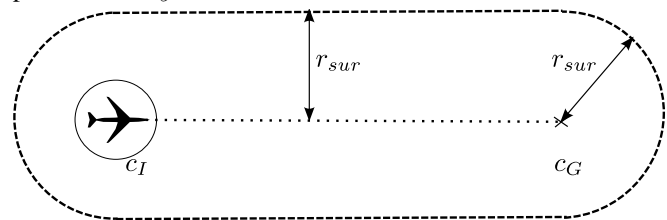


Fig. 3. The inclusion space.

The algorithm is provided in Algorithm 1. First the validity of the given configurations c_I and c_G is checked at line 1. If any of the configurations are not valid, an empty plan is returned. At line 3 the plan connecting the initial and final configuration is computed and stored into Φ using the `Connect` function. The plan stored in Φ is then checked against all obstacles \mathcal{O} in the environment by the function `TestObstacleIntersection`. If there is any obstacle colliding with the plan, the flag *intersecting* is set to true. Otherwise the algorithm returns the plan at line 23. If the plan does intersect an obstacle, the radius r_{sur} of IS is computed at line 8 using the `Dist` function, which computes the distance of c_I and c_G . The radius of the Inclusion Space r_{sur} determines the space containing the obstacles considered during the first iteration of the *while* cycle, see Figure 3. The obstacles that have an intersection with this space are considered for in first iteration.

In the *while* cycle the obstacles that do intersect with the plan are added to the set of obstacles which are going to be tested in the next iteration of the planning - the \mathcal{O}_{tested} . Then if the set of obstacles tested in the previous iteration of planning $\mathcal{O}_{testedOld}$ and the newly created set of obstacles \mathcal{O}_{tested} are the same, the algorithm terminates and returns an empty plan. Otherwise AA* is called at line 12 only for a limited subset of obstacles - \mathcal{O}_{tested} . After completion of the planning procedure, a check of intersection of the plan with the set of all obstacles in the environment is done. If no intersection is found, the algorithm returns the plan Φ returned by the function `AA*Search`. Otherwise another iteration of planning is executed.

Input: $c_I, c_G, \mathcal{O}, \alpha$

Output: Φ

```

{1} if  $\neg$  IsValid( $c_I$ ) or  $\neg$  IsValid( $c_G$ ) then
{2}   | return  $\emptyset$ ;
{3}
{4}  $\Phi \leftarrow$  Connect( $c_I, c_G$ );
{5} intersecting  $\leftarrow$ 
    TestObstacleIntersection( $\Phi, \mathcal{O}$ );
{6}  $\mathcal{O}_{testedOld} \leftarrow \emptyset$ ;
{7}  $\mathcal{O}_{tested} \leftarrow \emptyset$ ;
{8} if intersecting then
{9}   |  $r_{sur} \leftarrow \alpha * \text{Dist}(c_I, c_G)$ ;
{10}  |  $\mathcal{O}_{tested} \leftarrow$ 
    DeriveTestedSubset( $\mathcal{O}, \alpha, c_I, c_G$ );
{11}  | while true do
{12}  |   |  $\mathcal{O}_{tested} \leftarrow$ 
    |   | AddIntObstacles( $\mathcal{O}, \mathcal{O}_{tested}, \Phi$ );
{13}  |   |  $\Phi \leftarrow$  AA*Search( $c_I, c_G, \mathcal{O}_{tested}$ );
{14}  |   | if  $\Phi = \emptyset$  then
{15}  |   |   | break;
{16}  |   |   | end
{17}  |   | intersecting  $\leftarrow$ 
    |   | TestObstacleIntersection( $\Phi, \mathcal{O}$ );
{18}  |   | if  $\neg$  intersecting then
{19}  |   |   | break;
{20}  |   |   | end
{21}  |   |  $\mathcal{O}_{testedOld} \leftarrow \mathcal{O}_{tested}$ ;
{22}  |   | end
{23} end
{24} return  $\Phi$ ;

```

Algorithm 1: The Iterative AA* algorithm

D. 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 vehicle 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.

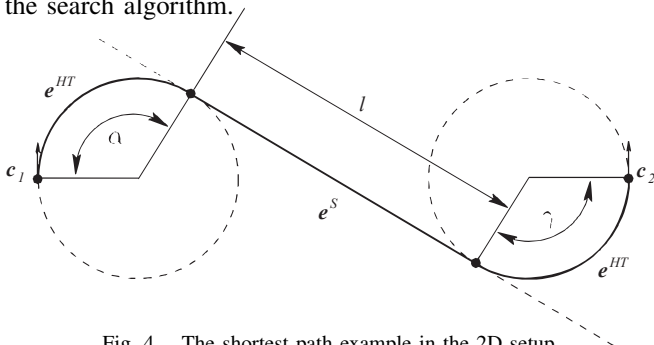


Fig. 4. The shortest path example in the 2D setup.

A two-dimensional example of one such connection is in Figure 4. The e^C in the example is composed of three

basic elements: *right horizontal turn, straight element* and *left horizontal turn*. The shortest path sequence is always constructed using the minimal radius of the *turn* and *spiral* elements available for the vehicle [11]. 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 [11], it is shown how the sequence can be identified. The function `Connect` is an extension of Dubins curves to a three-dimensional domain additionally using vertical turns and spiral elements, still providing the shortest connection.

E. TestObstacleIntersection Function

The `TestObstacleIntersection(Φ, \mathcal{O})`, where Φ is a plan to be tested and \mathcal{O} is a set of obstacles the plan is tested against returns *true* if any element e_i of the plan Φ intersects any obstacle and *false* otherwise. The algorithms for intersection checking are instantiated for each respective element $e^S, e^{HT}, e^{VT}, e^{SPIRAL}$.

We will now discuss the algorithm for testing the intersection of a straight flight element (represented geometrically by an arc a) with an obstacle o defined formally in Section II. Note that all non-straight elements are approximated by a number of straight elements prior to the testing.

The arc-obstacle intersection test has two parts: geometric and temporal, see Algorithm 2. The geometric intersection is tested for each *face* (side plane) f of the obstacle o independently. If no intersection is found, then the flight element does not intersect the obstacle. If any intersection is detected, the earliest and the latest one i_{first} and i_{last} is determined (measured by its angular distance from the current position of the airplane). Next, time stamps t_{first} and t_{last} corresponding to those intersection points are calculated. They are used for determining whether the detected intersection falls within the active time interval $o_T = (t_{from}, t_{to})$ associated with \mathcal{O} . If not, the flight path element and the obstacle do not intersect. Otherwise, an intersection has been detected.

A description of implementation of the function `CalculateIntersection` follows. Intersection of a face f with an arc a describing a straight element of the flight plan is calculated as a geometric intersection of two planes in 3D: the plane of f and the plane in which the arc a lies. The result of this intersection is a line from which we pick a single point i corresponding to the altitude of the airplane. If the altitude does not fall within the interval o_T of the obstacle, the intersection test fails ($i = \text{None}$). Next, we test whether the intersection point i falls within both the arc a and also within f . If either of these tests fails, f and a do not intersect.

Next, we will discuss the algorithm for testing whether a point falls inside an obstacle, or is located within a certain distance d from it. First, we compare the altitude of the point with the minimal and maximal altitudes of the obstacle. If the point falls outside the altitudinal interval of the obstacle, the test fails. Otherwise, we inflate the geometry of the obstacle

Input: arc a , obstacle o

Output: boolean (True iff a and o intersect)

```

{1}  $dist_{min} \leftarrow +\infty; dist_{max} \leftarrow -\infty;$ 
{2}  $i_{first} \leftarrow \text{None}; i_{last} \leftarrow \text{None};$ 
{3}  $counter \leftarrow 0;$ 
{4} foreach face  $f$  in  $o$  do
{5}    $I \leftarrow \text{CalculateIntersection}(a, f);$ 
{6}   if  $I \neq \text{None}$  then
{7}      $counter \leftarrow counter + 1;$ 
{8}      $dist = \text{CalculateDistance}(I);$ 
{9}     if  $dist < dist_{min}$  then
{10}       $dist_{min} \leftarrow dist;$ 
{11}       $i_{first} \leftarrow i;$ 
{12}     end
{13}     if  $dist > dist_{max}$  then
{14}       $dist_{max} \leftarrow dist;$ 
{15}       $i_{last} \leftarrow i;$ 
{16}     end
{17}   end
{18}   if  $counter = 0$  then
{19}     return False;
{20}   end
{21}    $t_{first} \leftarrow \text{CalculateTime}(i_{first});$ 
{22}    $t_{last} \leftarrow \text{CalculateTime}(i_{last});$ 
{23}   if  $o.t_o < t_{first}$  OR  $o.t_f > t_{last}$  then
{24}     return False;
{25}   end
{26}   return True;
{27} end

```

Algorithm 2: The arc-obstacle intersection test

by distance d . This is done by rotating each of the faces of the obstacle's mesh outwards (away from its center of mass) around the origin of the coordinate system (the center of the Earth). Next, we calculate the point's signed distance s_i from each of the vertical planes of the inflated obstacle:

$$s_i = \vec{p} \cdot \vec{n}_i,$$

where \vec{n}_i is the normal vector of the i -th geometric plane pointing out from the (inflated) obstacle and \vec{p} is the vector pointing from the center of the Earth to the point being tested.

A positive value of s_i means that the point lies outside of the obstacle, while a negative value means the point lies inside. Therefore, the point only passes the test if all of the calculated signed distances s_i are less or equal to zero.

F. DeriveTestedSubset Function

The output of the DeriveTestedSubset function is a subset of obstacles \mathcal{O}_{tested} that are used in the next planner invocation. The subset of obstacles \mathcal{O}_{tested} is constructed as an intersection of a set of all obstacles in the environment \mathcal{O} and the inclusion space \mathcal{IS} . Formally,

$$\text{DeriveTestedSubset}(\mathcal{O}, \alpha, c_I, c_G) = \{o \in \mathcal{O}; o \cap \mathcal{IS} \neq \emptyset\}.$$

IV. EXPERIMENTS

In this section the properties of AA* [7] are compared with the novel IAA* algorithm.

A. Scenario Domain

The provided algorithm was tested on The United States National Airspace System (US NAS) domain, see [12]. In the domain we model the air traffic in the US NAS using real data provided by Federal Aviation Administration (FAA). We use a spherical model of Earth using Earth-centered Earth-fixed coordinates (GPS). The provided data set consists of:

- Description of flights starting and landing in the USA, specifically from 30th Aug 2007 09:00 AM to 31st Aug 2007 08:59 AM. For each flight, the data file defines an initial configuration point c_I , the goal configuration point c_G and the velocity of the airplane. For each flight there are also several mid-waypoints the airplane has to pass when flying from c_I to c_G .
- Definition of the Special Use Airspaces (SUA). The SUAs are modeled as three-dimensional shapes derived from polygons projected on a spherical surface and extruded either towards or away from the center of the Earth (see Section II).

B. Scenario setup

We simulate one percent of the daily traffic (369 flights) in the domain. The set of simulated flights has been selected randomly from the set of all flights starting and landing in the USA from 30th Aug 2007 09:00 AM to 31st Aug 2007 08:59 AM described above. For each flight the initial configuration c_I and the goal configuration c_G are passed to the planning algorithm (we do not consider the mid-waypoints) and the algorithm plans the path in real time. We do not consider take-offs or landings of the airplanes – the airplanes are simply spawned in the air at the defined flight level with the defined direction vector and velocity, and they start flying to the destination. The measured data is the total time it takes the planner to find the plan. This data is then summarized over all 369 flights. The parameters in different experiment setups are α determining which zones are going to be tested in the first iteration of the algorithm and the *minimal planning step* which sets the minimal step of the AA* planning algorithm. The experiment has been repeated three times and the two tables present the average values.

C. Results

The total planning times summarized over all 369 airplanes are provided in Table I. The numbers of zones used in the latest iteration of the planning algorithm summarized over all 369 airplanes are provided in Table II. From the two tables it is apparent that the IAA* significantly outperforms the original AA* algorithm for all values of the minimal planning step and for all three selected values of α . This is apparently caused by the number of tested obstacles reduced by the IAA* algorithm. The average speedup ratio of the IAA* algorithm using the $r_{sur} = 0.1$ computed over all

		minimal planning step				
α		1 000	5 000	10 000	20 000	50 000
0.1		85 475	25 316	15 876	11 952	4 927
0.3		74 814	29 420	21 938	17 292	9 794
0.5		194 829	76 536	43 590	49 598	26 185
AA*		283 734	181 243	92 700	85 493	45 216

TABLE I

SUM OF PLANNING TIMES IN MS OVER MEASURED FLIGHTS

		minimal planning step				
α		1 000	5 000	10 000	20 000	50 000
0.1		1 129	1 156	1 076	1 170	1 169
0.3		2 794	2 779	2 657	2 822	2 818
0.5		4 071	4 172	4 008	4 240	26 185

TABLE II

SUM OF OBSTACLES USED FOR THE TEST

measured minimal planning step values is 6.52 compared to AA*. The speedup ratio $r_{sur}=0.3$ is 5.65 and for $r_{sur}=0.5$ it is 1.87. The average speedup is computed as an average of individual speedups.

The important property of the IAA* algorithm is the quality of the plan. The final plan was in all test cases exactly the same as the plan produced by the AA* algorithm.

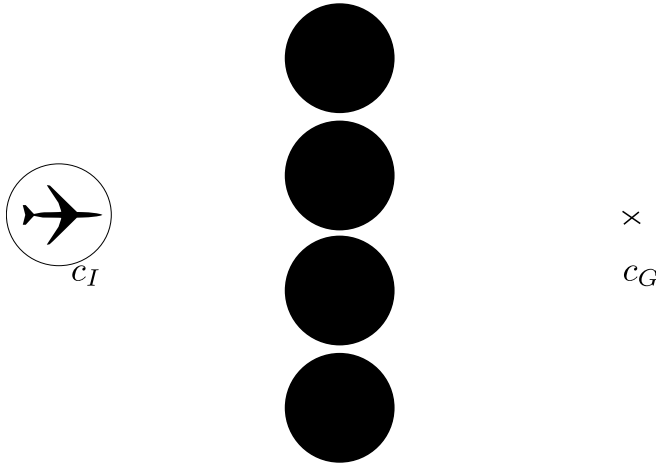


Fig. 5. Example of a domain where AA* finds the solution faster than IAA*. The black circles represent the obstacles. The Figure displays only the center of the setup to illustrate the placement of the obstacles easily. It is assumed that there exists a valid plan from c_I to c_G although the possible path is not displayed.

The results are very much domain dependent. The placement and size of the obstacles plays a crucial role and a domain where AA* provides better results than IAA* may be found. For example consider the obstacle placement in the Figure 5. In this domain, the IAA* algorithm selects several obstacles around the great circle connecting c_I and c_G based on the parameter α . These obstacles are put into the set \mathcal{O}_{tested} . The solution found by IAA* in each iteration violates exactly one obstacle which is added to \mathcal{O}_{tested} , therefore the algorithm needs to iterate many times to find the solution.

V. CONCLUSION

In the paper we have introduced a novel improvement of the *Accelerated A** planning algorithm called the *Iterative Accelerated A**. The algorithm utilizes the concept of iterated planning. It uses a limited subset of domain obstacles in the first iteration and if the generated plan intersects with any other obstacle that has not been considered, this obstacle is added to the test set and the next iteration of the algorithm is executed. This approach significantly reduces the number of obstacles and thus the number of computationally expensive intersection tests of the flight plan and obstacles. Such reduction of intersection tests implies reduction of the algorithm's execution time in domains with relatively sparse obstacles.

The IAA* algorithm has been tested extensively in the air traffic control domain on real data provided by FAA. In contrast with the widely used random techniques the IAA* algorithm keeps the properties of the AA* algorithm – it searches efficiently for the shortest possible path. The plan produced by IAA* algorithm is the same as the one produced by the AA* algorithm. Moreover, the final path produced by the algorithm is smooth. The algorithm may be further improved by distribution of the computation on multiple computation units.

REFERENCES

- [1] S. Carpin and G. Pillonetto, "Motion planning using adaptive random walks," *IEEE Transactions on Robotics and Automation*, vol. 21, no. 1, pp. 129–136, 2005.
- [2] S. M. LaValle and J. J. Kuffner, "Rapidly exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. MA, USA: A K Peters, Wellesley, 2001, pp. 293–308.
- [3] J. Barraquand and J.-C. Latombe, "A monte-carlo algorithm for path planning with many degrees of freedom," in *Proceedings of IEEE International Conference on Robotics and Automation*, 1990, pp. 1712–1717.
- [4] S. Carpin and G. Pillonetto, "Merging the adaptive random walks planner with the randomized potential field planner," in *Proceedings of the IEEE International Workshop on Robot Motion and Control*, 2005, pp. 151–156.
- [5] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *Proceedings of IEEE Conference Decision and Control*, 2005.
- [6] D. C. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.
- [7] D. Šišlák, P. Volf, and M. Pěchouček, "Accelerated A* path planning," in *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. New York: ACM Press, May 2009.
- [8] Y. K. Hwang and N. Ahuja, "Gross motion planning: A survey," *Advanced Robotics Redundancy and Optimization. ACM Computing Surveys*, vol. 24, no. 3, September 1992.
- [9] *User Manual for the Base of Aircraft Data (BADA) REVISION 3.7*, European Organisation for the Safety of Air Navigation EUROCONTROL, March 2009.
- [10] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, no. 79, pp. 497–516, 1957.
- [11] P. Souères and J.-D. Boissonnat, "Optimal trajectories for nonholonomic mobile robots," in *Robot Motion Planning and Control*, J.-P. Laumond, Ed. Berlin: Springer-Verlag, 1998, pp. 93–169.
- [12] M. S. Nolan, *Fundamentals of Air Traffic Control*, 4th ed. Belmont, CA, USA: Thomson Brooks/Cole, 2004.