Trajectory Planning on Grids: Considering Speed Limit Constraints

Lukáš CHRPA [a],
[a] *Agent Technology Center*
Faculty of Electrical Engineering
Czech Technical University in Prague
E-mail: chrpa@agents.felk.cvut.cz

Abstract. Trajectory (path) planning is a well known and thoroughly studied field of automated planning. It is usually used in computer games, robotics or autonomous agent simulations. Grids are often used for regular discretization of continuous space. Many methods exist for trajectory (path) planning on grids, we address the well known A* algorithm and the state-of-the-art Theta* algorithm. Theta* algorithm, as opposed to A*, provides 'any-angle' paths that look more realistic. In this paper, we provide an extension of both these algorithms to enable support for speed limit constraints. We experimentally evaluate and thoroughly discuss how the extensions affect the planning process showing reasonability and justification of our approach.

Keywords. Trajectory planning, A*, Theta*, Speed limit constraints

1. Introduction

Automated planning [5] belongs to one of the important research fields of Artificial Intelligence (AI). Trajectory (path) planning, a subgroup of AI planning, is a crucial part of systems controlling autonomous (mobile) agents or robots. There exist many advanced techniques for path (trajectory) planning, however, the techniques do not take into account agent (or robot) dynamics such as speed limit constraints. In some systems (discussed below) speed is computed as a post-processing step after the path is found. Such an approach, however, does not work if the found path cannot be accommodated with speed (to provide a trajectory) satisfying the given model of dynamics.

One of the well known techniques for path planning in continuous space is based on Visibility Graphs [8], that are graphs, where nodes represent the start and goal points as well as the corners of all obstacles and edges connect the nodes if and only if there is a line-of-sight between these nodes. In this case, path planning is about finding a path on the Visibility Graph from start to goal node. The shortest paths on Visibility Graphs are also the shortest paths in the continuous space. However, path planning on Visibility Graphs becomes significantly slower when the number of obstacles increases.

Continuous space can be discretized by regular geometric tessellations (usually square or hexagonal) - grids where each cell is either blocked or free. In other words we deal with path planning on grids [14]. A found path is represented as an ordered sequence

of cells. Path planning on grids is generally faster than on Visibility Graphs [9]. A basic and well known technique for path planning on grids uses the A* algorithm [6]. However, paths found by A* look unrealistic, since the paths connect only the centers of adjacent (or diagonally adjacent) cells in the grid. A simple approach for making paths more realistic rests in a post-processing step [1], where it is iteratively checked whether the current cell can be directly connected by a line (without crossing any blocked cell) with a cell lying on the path one position before the preceding one. This allows us to connect (by lines) more distant cells (not only adjacent cells). Theta* [9] (a variant of A*) incorporates this idea directly during the search. Theta* will be discussed more thoroughly later. Recent works in this area use Rapidly exploring Random Trees (RRT) [3] (most recently [12]), where the search is combined with randomized techniques. Contrary to A* or Theta*, RRT produces paths in shorter time, but the quality of paths (measured by their lengths) is often much worse.

As indicated before in the text, path planning has many practical applications. It is advisable to mention AgentFly [10], a system for Air Traffic Control where path planning is about chaining (irregular) maneuvers. Another state-of-the-art system deals with path planning for autonomous helicopters [13] where the path planning algorithm is based on Probabilistic Roadmaps [7]. The algorithm results in a sequence of way-points interpolated by cubic curves (splines). However, in both these systems (and many others) there is a certain need for handling even simple dynamic constraints, for instance, the speed limits for turn maneuvers etc. Both mentioned systems handle these constraints in post-processing steps, but this kind of approach will fail if previously planned paths are invalid in terms of the speed limit constraints.

In this paper, we will investigate the possibilities of trajectory planning with speed limit constraints. The idea was introduced in a planning method (based on A*) for helicopters [4]. We followed this idea by extending the A* and Theta* (path planning) algorithms with speed limit constraints. We also provide an experimental comparison of how the extension influences the planning process and the results will be thoroughly discussed in this paper.

This paper is organized as follows. The next section introduces the preliminaries, including the well known A* algorithm and the state-of-the-art Theta* algorithm. Then, we present how to extend both these algorithms to consider speed limit constraints. Then, we provide and thoroughly discuss experimental results.

2. Preliminaries

Path planning on grids is well known and can be considered common knowledge. In this section we introduce notions and concepts that are used in this paper to help the readers to better understand the paper.

According to the robotics theory we distinguish between the terms **trajectory** and **path** such that a trajectory includes dynamics such as speed while a path has only a spatial dimension.

A **cell** stands for an atomic part of a grid. A **node** is a well known notion from the graph theory. Thus, it is useful to represent grids by undirected graphs in such a way that every cell is represented by a corresponding node with edges between nodes representing adjacent cells. It obviously gives us a parallel between the graph theory and grid trajectory (path) planning.

A path (or a trajectory if dynamics is considered) is represented by an ordered sequence of nodes (it can also be understood as a parallel to the graph theory). If a node s lies on a path p, then a **parent(s)** stands for a node that lies on the path p directly before the node s. **Succ(s)** stands for a set of successors of s, i.e., $succ(s) = \{u \mid \text{there is an edge from } s \text{ to } u\}$.

*2.1. A**

A* [6] is a well known algorithm for an informed search in the state space. Because we believe that the A* algorithm belongs to common knowledge we do not provide details about it. The key point of A* rests in selecting nodes for expansion. We select such a node s whose sum $g(s)+h(s)$ is minimum. $g(s)$ stands for a real distance from an initial node to s. $h(s)$ stands for a heuristic estimation of distance from s to goal node. If h is admissible (does not overestimate distances), then the A* provides optimal solutions.

For path planning h can be defined as a Cartesian distance between the current node and the goal node (e.g., the cell positions the nodes represent) or especially in grid path planning Manhattan (for square grids) or Vancouver (for hexagonal grids) distances [14]. Obviously, if we use Cartesian distance, then the heuristic is admissible. If we use Manhattan or Vancouver distance, then the heuristic is admissible if and only if we do not allow diagonal moves.

*2.2. Theta**

Theta* [9] is a variant of A*. Although A* can be used in a general manner, Theta* focuses only on path planning. As mentioned in the Introduction, Theta* allows connecting (by lines) more distant (non-adjacent) cells which makes the found path more optimal and realistic looking. Inspired by [1], Theta* performs the following line-of-sight check (for grids we can apply a variant of the well known Bresenham's line algorithm [2]) while node s is being explored:

$$\textbf{If } \text{line\_of\_sight}(\text{s}, \text{parent}(\text{parent}(\text{s})) \textbf{ then } \text{parent}(\text{s}) := \text{parent}(\text{parent}(\text{s}))$$

To ensure admissibility of the heuristic we have to use the Cartesian distance instead of the Manhattan or Vancouver distance. Obviously, Theta* can produce paths shorter than (or the same as) A*, but as authors claim in [9], Theta* does not provide optimal plans with respect to continuous space.

3. Planning with Speed Limit Constraints

Introducing speed into the planning process has one key point - providing plans considering (even simple) dynamics of autonomous vehicles (or robots). The plans are in fact trajectories, so we are dealing with trajectory planning. For instance, it is well known that the minimum turn radius depends on the current vehicle speed, more specifically the minimum turn radius grows quadratically with speed. This can, of course, be generalized such that a certain maneuver or terrain requires a certain speed limit constraint. Obviously, we must not forget to take into account vehicles' speed limit constraints as well as their acceleration and deceleration limits.

Speed limit constraints as well as acceleration and deceleration ranges can be determined from vehicle dynamic properties, environmental properties (e.g., terrain type, slope etc.) and maneuver properties (e.g., turn radius). This kind of determination can be provided by a model of (vehicle) dynamics. Obviously, every vehicle should have its own model of dynamics. The model of dynamics must be capable of providing for every pair of (neighboring) nodes s, s' i) an interval of speed values $spdlim(s, s')$ which refers to the speed limit constraint required at the beginning of the maneuver (e.g., Turn maneuver) the vehicle must perform when going from s to s' and ii) an interval of the speed values $spdreach(s, v, s')$ (v stands for the current speed in s) which refers to a range of speeds that can be achieved after the vehicle performed the maneuver from s to s'. For now we consider the model of dynamics to be a black-box procedure.

Section 2 discusses the parallel between path planning on grids and graph theory. For trajectory planning we have to take into account the model of dynamics which modifies the graph. Besides the current position (cell) we also take into account the current direction (heading) and current speed. Obviously, there exists a unique node for every combination of a cell (position), direction (heading) and speed value. Then, the number of nodes we must consider (in the worst case) is $\#nodes = |cells| * |directions| * |speed\ values|$. Edges can be defined in such a way that the (directed) edge between nodes a and b exists if one of the following conditions holds:

- a and b represent adjacent cells, having the same directions and for any speed value v_b in b and speed value v_a in a holds that $v_a \in spdlim(a, b)$ and $v_b \in spdreach(a, v_a, b)$.
- a and b represent the same cell, the same speed value v such that $v \in spdlim(a, b)$ same)

Obviously, it is not necessary to construct the whole graph at the beginning, we only expand nodes selected in each iteration of A* (according to minimum $g(s) + h(s)$). Clearly, by applying A* we can easily obtain a solution. This approach might work well if $|directions| * |speed\ values| \ll |cells|$. However, if we use Theta* – the 'any-angle' path (trajectory) planning method, then we can easily find out that the number of directions can be very high (but finite on finite grids). Similarly, the number of possible speed values can be very high (or infinite). It can be easily seen that the number of nodes can be much higher than in the basic case.

The number of (explored) nodes can be reduced if we consider intervals of speed values (hereinafter speed intervals) rather than single speed values. It actually corresponds to sets of the nodes defined in previous paragraphs. Giving an informal insight, in every node speed intervals represent a range of speed values that can be reached (from the initial node) with respect to all constraints defined in a corresponding model of dynamics. Obviously, these intervals allow us to process (many) more speed values in a single step.

To be more precise, since the speed intervals are incorporated (A* and Theta* algorithms are augmented), some aspects of the algorithms are modified. The aspects of the algorithms are discussed in the following paragraphs. We must extend the definition of $spdreach$ to support speed intervals, i.e., if s and s' are (neighboring) nodes and $spd(s)$ stands for a speed interval in s, then $spdreach(s, spd(s), s') = \bigcup_{v \in spd(s)} spdreach(s, v, s')$.

In the basic model (without dynamics), $\mathbf{succ(s)}$ returns the nodes representing all the cells adjacent to s. Considering our model (with speed limit constraints), there is a

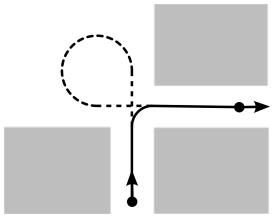

Figure 1. The solid-lined trajectory is considered when a vehicle is moving slowly enough to perform a 'normal' turn maneuver. The dash-lined trajectory must be considered when a vehicle is moving too fast to perform a 'normal' turn maneuver which is replaced by an 'ear'.

restriction such that the speed interval in s ($spd(s)$) must not be disjoint with the speed limit constraint applicable while transiting to the successive node s' ($spdlim(s, s')$), formally: $spd(s) \cap spdlim(s, s') = \emptyset \Rightarrow s' \notin succ(s)$ (Regarding Theta* this check is made if a line-of-sight check between s' and $parent(s)$ fails). Of course we must update speed intervals for every (valid) successor s' of s (e.g., $spd(s') = spdreach(s, spd(s), s')$).

In the basic model, **Previously visited nodes** are specified only by cells represented by the nodes, so we only need to check if we have visited the corresponding cell. In our model, we distinguish the nodes also by the direction (heading) and the speed interval (necessary for ensuring completeness). Here, we need to check if we have visited such a node that has the same position and direction and its speed interval is a superset (or equal).

Besides the **line-of-sight** check between $parent(s)$ and s', we must also check speed limit constraints occurring when s' becomes a successor of $parent(s)$ (e.g., $spd(parent(s)) \cap spdlim(parent(s), s') \neq \emptyset$). If either check fails, then $parent(s)$ and s' cannot be directly connected, i.e., the line-of-sight check fails. Otherwise we must not forget to compute $spd(s')$ (e.g., $spd(s') = spdreach(parent(s), spd(parent(s)), s')$).

Taking a closer look we can easily find out that the planning process (augmented A* and Theta*) is sound, i.e., every solution we find is valid with respect to the given model of dynamics. Completeness of the planning process rests in (as indicated before) distinguishing nodes not only by their position (cell), but also by their direction (heading) and speed value (or speed interval). The necessity of such node distinguishing is illustrated in Fig. 1, where the dash-lined trajectory is the only option if the speed is not low enough to perform a turn maneuver depicted by the solid-lined trajectory. It can be seen that the solid-lined trajectory passes one cell twice (with a different direction).

On the other hand the number of nodes considered during the planning process may be extremely high (just keep in mind the number of possible directions while using Theta*) which may significantly slow down the planning process. To keep the number of considered nodes within 'reasonable' bounds we can distinguish the nodes only by their position (cell) as it is in the basic model (see Section 2). It affects only the detection of previously visited nodes, i.e., we check only if we have visited any node referring to the corresponding cell. We expect that in most cases the number of explored nodes will be at most slightly higher which (as we believe) will lead to a relatively insignificant increase of running time (e.g., time needed by the planner to find a solution). However, generally it can be done at the cost of 'sacrificing' completeness (this issue is discussed in Section 4).

The last point we would like to focus on is plan (trajectory) extraction. The output we get from A* and Theta* is an ordered sequence of nodes. From this sequence we

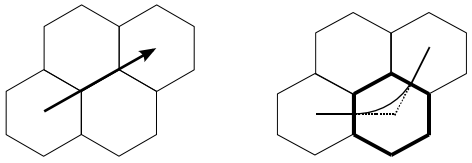

Figure 2. A sample 'diagonal' movement (left hand side). A sample turn maneuver (right hand side).

can obtain an ordered set of points, where each point is provided with a corresponding direction and speed interval. These points may not only directly refer to cells, especially if Theta* is used, but it is necessary to compute (the rest of) the speed intervals. Now, each point has a speed interval corresponding to the part of the trajectory going from the initial point to the current point. It means that there might be disproportions in the speed intervals, because speed limitations that might occur later are not taken into account. For instance, a vehicle can go between 10 and 100km/h at some point. Right after that, the vehicle performs a turn maneuver, where the speed is limited to 50km/h. It is clear that the vehicle cannot slow down from 100km/h immediately. Obviously, we have to back-propagate the speed intervals with respect to the whole trajectory. The idea rests in updating the speed intervals from the end of the trajectory to the beginning (e.g., from this we can get the necessary slow-downs or speed-ups to fit the speed limits), for a deeper insight into the back-propagation, see [4]. The final speed intervals are obtained as intersections of the initially computed and back-propagated speed intervals. Having the final speed intervals in every point we can simply extract the exact speed values as maximum values of the intervals. Then, we can simply compute the time-stamps.

4. Evaluation

4.1. Implementation

For experimental purposes we implemented both the A* and Theta* algorithms augmented to handle speed limit constraints. We used a hexagonal grid. As adjacent cells we also consider cells placed 'diagonally' (see Fig. 2 on the left hand side). 'Diagonal' moves make paths (or trajectories) found by the A* algorithm more realistic looking. As a heuristic estimation we use Cartesian Distance (between the cells represented by the current and the goal node).

Found paths (trajectories) are in fact polygonal chains connecting the hexagon centers. For our purpose it is necessary to compute exact turn radiuses in (hexagonal) cells, where the direction is changed. Of course, an infinite number of possible turn maneuvers exist even if the direction change is given. We decided to keep the turn maneuver within the cell in which the direction changes but we want to keep the turn radius as big as possible. It is depicted in Fig. 2. The turn radius r is then computed as $r = \frac{d}{2} \cot \frac{\alpha}{2}$ (d stands for the distance between adjacent cells (not 'diagonal'), α stands for the angle of direction change). The computation of turn radii is essential for determining speed limit constraints (e.g, $spdlim(s, s')$). Our (simple) model of dynamics considers the physical law according to which an increase in speed results in a quadratic increase of the minimum turn radius (i.e., $\Delta r = (\Delta v)^2$). Our (simple) model of dynamics allows only to adjust speed during moving straight (on lines) which is based on uniformly accelerated

	Solved %	Timeout exceeded %	Time	Trajectory length	Nodes
$\sim 10\%$ cells blocked					
A* (lite)	96.4	0.0	0.978	1.001	0.945
A* (full)	46.6	53.4	87.166	1.005	14.689
Theta* (lite)	99.6	0.3	1.137	1.000	0.999
Theta* (full)	2.0	98.0	8.342	1.000	5.516
$\sim 20\%$ cells blocked					
A* (lite)	95.2	0.0	1.028	1.002	0.941
A* (full)	50.1	49.9	86.485	1.007	15.643
Theta* (lite)	99.5	0.0	1.101	1.001	0.999
Theta* (full)	1.6	98.4	1.733	1.000	1.526
$\sim 30\%$ cells blocked					
A* (lite)	95.0	0.0	1.060	1.005	0.946
A* (full)	46.8	53.2	95.674	1.009	15.418
Theta* (lite)	98.7	0.0	1.093	1.001	1.000
Theta* (full)	1.4	98.6	2.807	1.000	11.336

Table 1. The experimental results. All the values are normalized against the corresponding basic versions.

motion (maximum acceleration and deceleration are given). Speed cannot be changed while turning. We do not consider other limitations (like terrain, etc.), but on the other hand it may not be so difficult to potentially extend the implementation in this way.

4.2. Experiments

For experimental purposes we implemented (as described in the last subsection) and evaluated i) **basic versions** of A* and Theta* (no dynamics considered), ii) **full versions** of augmented A* and Theta* (nodes distinguished by their position, direction and speed interval) and iii) **lite versions** of augmented A* and Theta* (nodes distinguished only by their position). The implementation is coded in JAVA SE 1.6 and not optimized for performance (can be possibly improved). All the benchmarks were performed on Core2Duo 1.86 GHz, 2 GB RAM, Windows 7. An area of size 1000x1000 units is considered for the experiments, where a distance of the adjacent hexagonal cells (not 'diagonal') is 10 (units). It gives us a grid of size about 100x100 cells. Maximum speed was set to 10, maximum acceleration was set to 0.5, maximum deceleration was set to -0.5, minimum turn speed was set to 0.1 and, finally, maximum turn speed for turn angle $\frac{\pi}{6}$ was set to 5. Regarding units we might consider meters for the distances, m/s for the speeds and m/s^2 for the acceleration and deceleration. For the benchmarks we randomly generated 100 points and then we plan trajectories between each other (it gave us up to 4950 planning problems – some of them are (trivially) unsolvable, because some points lie in the blocked cells). We have three 'maps' ($\sim 10\%$, $\sim 20\%$ and $\sim 30\%$ cells blocked). We also randomly generated initial speed and maximum goal speed for every point. For every planning problem we set a timeout of 1s (e.g., 1000ms).

Table 1 presents our experimental results. All values are normalized against the corresponding basic versions (by the used algorithm and 'map'), e.g., we take into account only problems solved by the corresponding basic versions (e.g., the lite version augmented A* is normalized against the basic version of A*). Column (Solved %) refers to the percentage of solved problems (within the timeout). Column (Timeout exceeded %)

refers to the percentage of problems where the solution time exceeded the given timeout (1s). Column (Time), is computed as $\frac{\sum_i t_{aug_i}}{\sum_i t_{bas_i}}$ (we take into account only running times of such problems successfully solved by both augmented (t_{aug}) and corresponding basic versions (t_{bas})). Columns (Trajectory length) and (Nodes) are computed analogically.

Note that only a very few problems were solved by the full version of Theta* (all three 'maps'). Consequently, the corresponding values in the table, especially time and nodes, are very inaccurate.

4.3. Discussion

Comparison of basic versions of A* and Theta* is discussed in [9]. Obviously, it comes as no surprise that found paths were shorter when Theta* was applied. On the other hand, time spent by Theta* was sometimes slightly higher. In this paper, our purpose is to compare the lite and full versions of augmented A* and Theta* algorithm against the basic versions.

Comparing the basic and lite versions (of augmented A* and Theta*) we found out several interesting points. The running time was generally a bit higher in the lite versions which was caused mainly by overheads required for handling speed limit constraints. In fact, the lite version of A* performed (according to our results) a bit (at most 6%) slower than the basic version, but sometimes even a bit faster. It is caused by the fact that the number of opened nodes was about $5-6\%$ lower in the lite version of A*. Since in the lite versions we distinguish the nodes only by their position (cell) and speed limit constraints might disallow some maneuvers, then it is obvious why the number of opened nodes can decrease. The lite version of Theta* performed (according to our results) about $10-13\%$ slower than the basic version. The reason for the twice as high slow-down compared to the A* results lies in the fact that Theta* allows 'any-angle' movement which requires to compute speed limits for an arbitrary turn angle. In the A* case we only have several directions which can be followed (12 in our implementation), thus we have a limited number of turn maneuvers and the speed limits for them can be cached. Paths found by the lite versions were the same as in the basic model or (very) slightly longer. Clearly, if the paths were the same, then the basic models found paths with respecting given speed limit constraints (i.e., we can assign speed values in a post-processing step), otherwise they did not. As mentioned before the lite versions are not complete, i.e., they might not find a solution even if it exists. Our experiments showed that in the A* case (lite version) about 5% of tasks were not solved (i.e., stated as unsolvable within the time limit) and in the Theta* case (lite version) just about 1% of the tasks.

Full versions of augmented A* and Theta* are, contrary to the lite versions, complete approaches. However, the experiments confirmed what we stated in Section 3. Using the full version of A* we were able to solve about 50% of the problems, otherwise the given time limit expired. The number of opened nodes was about 15x larger and the time spent on solving was about 90x larger (compared with the basic version). Besides the overheads due to speed limits computation this slow-down was caused by the fact that while testing if we have visited some node before, we have to check not only the cell position but also the direction and speed interval. Regarding the full version of Theta* we were only able to solve at most 2% of the problems, otherwise the given time limit expired. It actually means that we were able to solve only a few simplest problems. Thus, the other values such as time and opened nodes are very inaccurate.

Taking a careful look at the lite versions we can see that for almost every problem its unsolvability was 'proved' within the time limit. Since nodes are distinguished only by their positions (like in the basic versions), we consider only the nodes with the lowest $g(s)+h(s)$ regardless of the other attributes (direction and speed interval). As illustrated in Fig. 1, sometimes a certain position (cell) must be considered (re-entered or re-opened) more than once. Secondly, the experiments showed that if we are unable to slow down from the initial to goal speed (or vice versa) within a short distance, then the selected node s (having minimum $g(s) + h(s)$) in the goal position (cell) can have inappropriate speed interval (e.g., the speed interval is disjoint to the given goal interval). In that case we are unable to find a solution in the lite versions, because we cannot re-open the node s. We believe that if we modify the heuristic h in such a way that we penalize such situations (i.e., if we cannot slow down or speed up from the current node to the goal), then we would be able to solve such kinds of problems even by the lite versions.

We showed that in the most cases we can use the lite versions for trajectory (path) planning with speed limit constraints at the cost of only slightly worse performance. However, as we have shown it does not work in some (special) cases. Using the full version of A* might be helpful, however, it might be appropriate only for problems where we need to explore only small areas. However, in general we cannot ensure to solve every problem, because the lite versions might fail to find a solution due to incompleteness and the full versions might fail to find a solution in a reasonable time. The question is how we can identify a class of such (special) problems. However, identification whether a problem belongs to this class or not does not seem to be straightforward and requires further investigation.

The experiments were done in 2D space, but we preliminarily tested the behavior of augmented A* and Theta* algorithms also in 3D space. Preliminary results showed that the performance of the lite versions against the basic versions in 3D is sightly worse than in 2D. It is caused by slightly bigger overheads in the computations of speed limits for certain maneuvers (we have to take into account pitch angles, for instance). Another possibility rests in optimizing the time (not only the distance) needed to move from the starting point to the goal point. To do this it is necessary to modify the computation of g and h to consider time (e.g., $g(s)$ stands for time needed to move from the start to s and $h(s)$ stands for heuristic estimation of the time needed to move from s to the goal). It will obviously increase the computation time for both g and h. On the other hand we believe that the potential slow-down of the planning process will still be within reasonable bounds.

Besides the issues we discuss in the last paragraphs (such as incompleteness of lite versions) we should investigate further possibilities of improving the trajectory planning process. First, studying the ideas behind the Accelerated A* algorithm [11] could help to reduce the number of considered nodes in all the presented modifications of A* and Theta*. Second, we may inspire from techniques based upon RRTs [3,12] which could also help us to reduce the number of considered nodes.

5. Conclusions

In this paper we presented an extension of the well known A* and the state-of-the-art Theta* algorithms for path planning on grids. The extension incorporates handling with

speed limit constraints, i.e., found trajectories respecting the given model of dynamics. The experimental evaluation, where we introduced two versions of augmented A* and Theta*, namely, the lite versions (the nodes distinguished only by their positions) and the full versions (the nodes also distinguished by directions and speed intervals). The experimental results we got showed that the lite versions did not have much worse performance than the basic versions even though the lite versions brought additional expressiveness. However, we did not successfully solve all the test problems by the lite versions even though the problems were solvable. Fortunately, the number of problems incorrectly marked as unsolvable were quite low (up to 2% for the lite version of Theta*). On the other hand, this percentage could increase in environments with high density of obstacles (blocked cells) and/or it is also dependent on the particular model of dynamics. Identification of such problems as mentioned before remains an open problem and should be investigated in future. Despite the incompleteness the results showed the reasonability of using the lite versions (especially the Theta* one) in trajectory planning, where we have to deal with dynamics such as speed limit constraints.

Acknowledgements We would like to thank Dušan Pavlíček for careful proofreading of the paper. This work was supported by Czech Ministry of Education, Youth and Sports under Grant MSM6840770038, by U.S. Army Grant W911NF-81-1-0521 and by SAAB under no. 1600039230.

References

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

[2] J. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[3] P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *Proceedings of ICRA*, pages 267–272, 2002.

[4] L. Chrpa and A. Komenda. Smoothed hex-grid trajectory planning using helicopter dynamics. In *Proceedings of International Conference on Agents and Artificial Intelligence (ICAART)*, volume 1, pages 629–632, 2011.

[5] M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice*. Morgan Kaufmann Publishers, 2004.

[6] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] L. E. Kavraki, P. Svestka, L. E. Kavraki, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[8] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commununications of the ACM*, 22(10):560–570, 1979.

[9] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *Proceedings of AAAI*, pages 1177–1183, 2007.

[10] D. Šišlák, M. Pěchouček, P. Volf, D. Pavlíček, J. Samek, V. Mařík, and P. Losiewicz. *Defense Industry Applications of Autonomous Agents and Multi-Agent Systems*, chapter AGENTFLY: Towards Multi-Agent Technology in Free Flight Air Traffic Control, pages 73–97. Birkhauser Verlag, 2008.

[11] D. Šišlák, P. Volf, and M. Pěchouček. Accelerated a* path planning. In *Proceedings of AAMAS (2)*, pages 1133–1134, 2009.

[12] W. Wang, X. Xu, Y. Li, J. Song, and H. He. Triple rrts: An effective method for path planning in narrow passages. *Advanced Robotics*, 24(7):943–962, 2010.

[13] M. Wzorek and P. Doherty. Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *Proceedings of ICAPS*, pages 438–441, 2006.

[14] P. Yap. Grid-based path-finding. In *Proceedings of Canadian Conference on AI*, pages 44–55, 2002.