UAV 4D Grid-Based Trajectory Planning using SWIM Data and Interval Trees

Stephan Heinemann
Department of Computer Science
University of Victoria
Victoria, Canada
Email: stephan@rigiresearch.com

Hausi A. Müller
Department of Computer Science
University of Victoria
Victoria, Canada
Email: hausi@uvic.ca

Afzal Suleman
Department of Mechanical Engineering
University of Victoria
Victoria, Canada
Email: suleman@uvic.ca

***Abstract*—Grid-based planning using heuristic path planning algorithms is one of the classic approaches to find a cost-optimal path for a moving agent operating in an non-uniform cost environment. We show how applicable costs can be modelled as cost interval trees and be aggregated by voxels of the planning grid. The interval trees allow for efficient lookup of valid, and possibly overlapping, intervals representing transition costs in a dynamically changing environment. In the domain of airborne mobile agents, the FAA NextGen and Eurocontrol SESAR initiatives aim at sharing relevant data between aviation stakeholders in real-time via the System-Wide Information Management (SWIM) infrastructure. Temporality-enabled SWIM data can be embedded nicely into planning grids using cost-interval trees. Furthermore, air-data-interval trees may improve the planning results by taking into account the changing capabilities and performance limitations of the agent. The enhanced grid with its SWIM data embeddings is amenable to the entire family of A*-based algorithms including their online, dynamic, anytime and any-angle extensions. As a multi-resolution grid it can be dynamically refined or coarsened depending on the available data and planning constraints of the agent.**

***Keywords*-motion planning; grid-based planning; heuristic planning; trajectory planning**

I. Introduction

The transportation sector, and in particular the aviation domain, increasingly depends on the availability of real-time data to derive the necessary information that enables further growth while ensuring flexible, economic and safe operations. The format and distribution of relevant data for the planning and execution of airborne missions has evolved from human-readable publications on paper to machine-readable subscriptions of relevant topics using a service-oriented software architecture.

The FAA NextGen programs[1], and in particular the efforts of the System-Wide Information Management (SWIM)[2] community, as well as the Eurocontrol SESAR[3] initiative aim at standardising the format and distribution of aeronautical, flight, weather, airport and maintenance data. The improved availability, variety, frequency and validity of relevant data enables the derivation of valuable information to support and make better decisions online, i.e., during an airborne mission.

It is obvious that higher levels of automation are required to support both manned and unmanned air operations with advanced filtering, fusing, learning and decision support (making) capabilities to manage the sheer amount of real-time data. Artificial intelligence solutions for trajectory planning and learning can be integrated into self-adaptive frameworks that monitor, analyse, plan and execute an airborne mission continuously.

Interesting aspects including interactions with humans in the loop such as crews, air traffic control, and dispatch (mission control), the available communications infrastructure and situation-aware levels of autonomy eventually have to be addressed by future airborne decision support systems.

This article discusses selected aspects of the trajectory planning feature of the Smart Autoflight Control System (SAFCS) framework[4] which is being developed at the Rigi Research[5] and Center for Aerospace Research[6] labs at the University of Victoria.

Section II describes one of the supported planning environment types of the SAFCS, namely grid-based environments, and how SWIM data is embedded into these environments using geometric embeddings and interval trees. Interval trees enable the efficient insertion and lookup of cost intervals in planning grids and, hence, facilitate trajectory planning and decision support algorithms.

Section III explains how aircraft (agent) capabilities influence the planning and decision support algorithms. It is emphasised that both the environment and the capabilities are to be considered dynamic and have to be taken into account to establish a feasible radius of action.

Sections IV and V provide an overview of grid-based trajectory planning with interval trees based on the environment and capabilities model presented before. The role of selected cost and risk policies for planning decisions and resulting trajectories is discussed.

Results of the initial evaluation of the SAFCS framework are presented in Section VI and plans for future work and the way ahead in Section VII lead to the conclusion of this article in Section VIII.

[1] https://www.faa.gov/nextgen/programs/
[2] https://www.faa.gov/nextgen/programs/swim
[3] https://www.eurocontrol.int/sesar-research
[4] http://search.maven.org/#search|ga|1|cfar
[5] http://www.rigiresearch.com/people/stephan-heinemann
[6] http://www.uvic-cfar.com

II. Environment

Trajectory planning can be performed based on different environment models. Environments can be modelled to be continuous or discrete in time and space. Their features can be considered static or dynamic; known, partially known or unknown in advance. They can be seen as benign or adversarial with respect to an agent performing tasks in them. Depending on the complexity of the environment model, different types of algorithms can be applied performing differently with respect to the quality of a solution and in terms of the time required to obtain that solution.

A. Planning Grids

Planning grids are one type of environment supported by the SAFCS framework. They represent an environment model which is discrete in space. A planning grid (or mesh) consists of discrete voxels where planning is performed using the neighborhood relation between vortices or center points of each voxel. Nash and Koenig [25] summarise different types of planning grids in the context of any-angle planning and the Theta* family of algorithms.

A cubic planning grid is based on cubic voxels as shown in Figure 1. The figure also shows how grids can be refined to become multi-resolution grids of arbitrary depth. Notice how the foremost cube is refined into eight child-cubes.

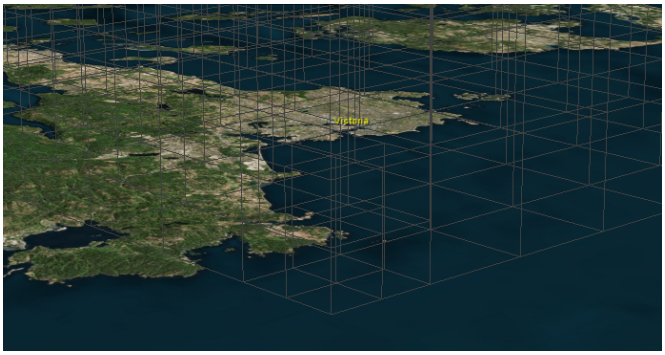

Fig. 1. Cubic Planning Grid

Depending on the required quality of a solution and the time available to find that solution, a multi-resolution planning grid can also be refined and coarsened dynamically. Ferguson, Stentz and Carsten [11], [5], [14] discuss approaches to grid-based planning in such an environment for the D* family of algorithms.

B. Embeddings

Usually, non-uniform cost planning grids associate a certain cost with each of their voxels. In a dynamic environment these costs may furthermore change over time possibly requiring a revision of a previously established trajectory.

Costs in the aviation domain can be associated with many different factors including aeronautical, flight, weather, airport and maintenance data. The format and distribution of this data is currently being standardised and tested. XML schemas have been defined to describe all of the above data, i.e., Aeronautical Information Exchange Model (AIXM[7]), Flight Information Exchange Model (FIXM[8]), (ICAO) Weather Information Exchange Model (WXXM[9], IWXXM), Airport Mapping Exchange Model (AMXM[10]), and Maintenance Management Information Exchange Model (MMIXM). The available exchange models are developed under the umbrella of the System-Wide Information Management (SWIM) program which enables stakeholders to subscribe and publish data using an implementation of the Java Messaging System (JMS) specification[11]. We have made Java XML bindings available to the community which allow for the convenient marshalling and unmarshalling of SWIM data supporting many of the available extensions[12].

To provide some examples, the airspace structure including active, inactive, or temporarily active airspaces as well as available ground navigation facilities are part of AIXM. Planned flights and trajectories including very detailed data about the aircraft involved are part of FIXM. Fronts, clouds, precipitation, obscuration or any significant weather and threats such as thunderstorms, icing and turbulence are part of WXXM/IWXXM. The layout of an aerodrome including its runways, taxiways, ramps and gates are part of AMXM. Maintenance schedules and facilities are part of MMIXM.

Representing the environment as a cubic multi-resolution planning grid with dynamically adapted resolution, the above data needs to be translated into costs of each voxel. Furthermore, data usually has a validity duration, e.g., an airspace gets activated at a specified time and deactivated later, a runway or taxiway is suddenly closed and then reopened, a front passage and the associated turbulence and precipitation is expected to happen within the forecasted time interval.

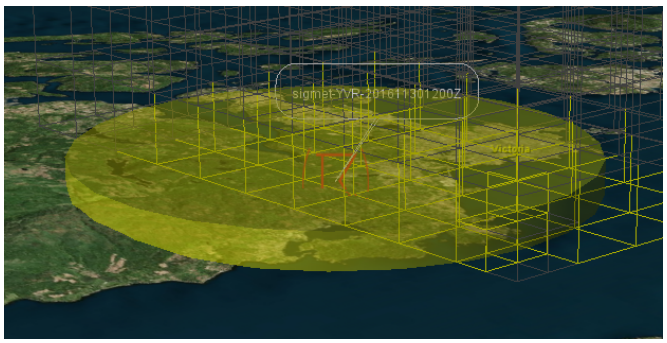

Fig. 2. Cubic Planning Grid Embedding

SWIM data is consequently defined in both time and space. The space component may employ geometric extents such as cylinders, boxes, cones, or simply individual positions. As explained above, there is usually a validity period associated

[7] www.aixm.aero
[8] www.fixm.aero
[9] www.wxxm.aero
[10] www.amxm.aero
[11] http://dev.solace.com/tech/jms-api/
[12] http://www.aixm.aero/page/open-source-projects

with the data as well to account for the time component. Geometric extents may intersect with voxels of the grid and intersections can be computed using the available collision detection algorithms as described by Ericson [7]. The SAFCS framework is however based on the NASA Worldwind SDK[13] which already supports sophisticated intersection tests between geometric extents.

Embeddings of SWIM data into multi-resolution cubic grids furthermore have to be performed recursively depending on the depth of the grid. It is obvious that a relatively small intersection could cause a comparably large voxel to be affected by data. Depending on the planning constraints such as quality and time, dynamic refinement or coarsening, that is, the adaptation of the planning environment may lead to better results. Figure 2 shows an embedding example of IWXXM data describing a thunderstorm into the voxels of a planning grid. Embeddings determine which voxels are affected by SWIM data. They may overlap in space and time, and even invalidate (cancel) previously published data.

C. Cost Intervals

Time intervals can be efficiently represented as balanced binary trees, e.g., augmented trees as described by Cormen et al. [6]. Interval trees allow for an $\mathcal{O}(\log n + m)$ lookup of any stored interval that coincides with another specified time interval or instance (an empty interval) where n is the number of stored intervals and m the number of matching reported intervals (output sensitive). Temporality-enabled SWIM data can, hence, be translated into costs which are then inserted into cost interval trees.

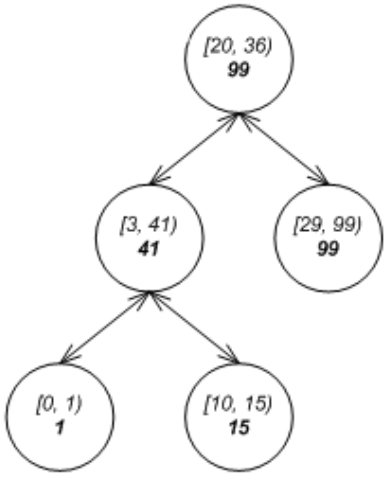

Fig. 3. Interval Tree

Each voxel of the planning grid aggregates such an interval tree which stores the applicable cost intervals of any subscribed SWIM topic. Planning algorithms may then determine the cost of a voxel during the planning phase by looking up the cost intervals during the estimated duration of passage through the voxel, and combining the latter using a cost function as further detailed in Section IV.

Figure 3 depicts an augmented tree implementing intervals[14]. This binary tree is always kept in balance during any modifications such as insertion and removal, never possessing a depth difference of more than one between any of its branches.

The key of any node in this interval tree corresponds to the lower bound of an interval. In the above example, the intervals are closed for lower bounds and open for upper bounds. If a new interval has to be added, then it is recursively inserted into the left sub-tree if the interval's lower bound is less than or equals the key (lower bound) of the root of the current tree. Otherwise it is recursively inserted into the right sub-tree. The annotation (bold number) of an interval tree node represents the maximum upper bound of all intervals stored in itself and its sub-trees.

The SAFCS planning grids employ cost interval trees. Cost intervals are implemented[15] as extensions of time intervals using Universal Time Coordinated (UTC) instances for lower and upper bounds. All aviation data is usually referenced with respect to UTC describing validity periods. Hence, a cost interval represents a certain cost associated with a SWIM data item which is valid during its time interval. Each planning grid voxel aggregates such a cost interval tree accounting for both space and time of applicable costs.

D. Air-Data Intervals

Aircraft performance is significantly affected by air data. The air density, that is, the density of the fluid through which the aircraft moves, influences the amount of aerodynamic forces created by its wings, stabilisers, and primary and secondary control surfaces (aerodynamic performance). It also influences the power and thrust generated by the powerplant of the aircraft (engine performance). Air density in turn is affected by air temperature, pressure and humidity. Furthermore, vertical and horizontal movements of the air such as winds, convection and subsidence affect the performance of the aircraft over the ground (range, radius of action).

WXXM/IWXXM data can be used to create air data intervals similar to cost intervals described above. These air data intervals may then be employed by planning algorithms to determine estimated climb, cruise and descent performances during the passage of a planning voxel. Air data intervals have not yet been implemented and manually configurable (static) aircraft capabilities are used for now.

III. Capabilities

The environment in which an agent operates is only one side of the coin with respect to trajectory planning – the (dynamic) capabilities of the agent (which may be affected by the dynamic environment) is the other.

[13] https://worldwind.arc.nasa.gov

[14] https://en.wikipedia.org/wiki/Interval_tree

[15] https://github.com/stephanheinemann/worldwind/tree/master/src/main/java/com/cfar/swim/worldwind/planning/CostInterval.java

Aircraft capabilities include climb, cruise and descent performance. They comprise energy (fuel) available and consumption, and the resulting endurance given a selected performance configuration. Capabilities also include equipment which might be required to operate in a certain environment, e.g., de-icing equipment.

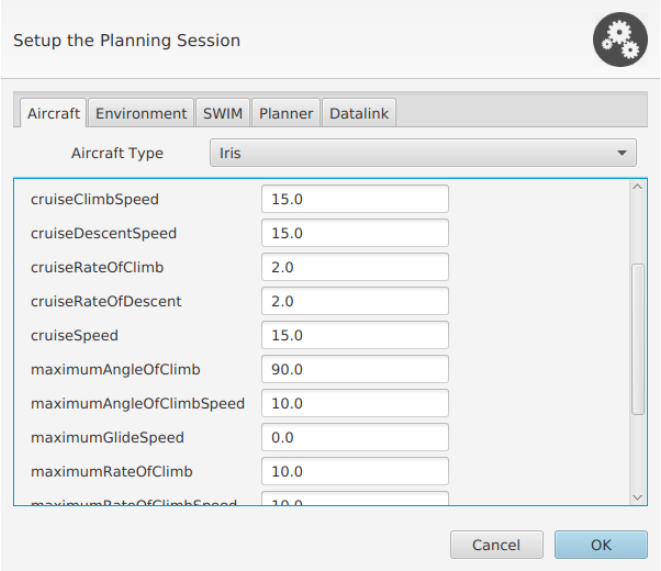

Fig. 4. Capabilities

The SAFCS currently uses manually configurable (static) aircraft capabilities to determine aircraft performance, range and endurance within an environment. The performance limitations of the aircraft constrain the feasible trajectories (non-holonomic constraints). Hence, it is possible that a feasible trajectory cannot be computed if the performance limitations of the aircraft (within an environment) are exceeded. Figure 4 shows an example of capabilities for a small Unmanned Aerial Vehicle (UAV) used for initial flight testing with the SAFCS.

IV. Planning

Classical Grid Search (GCS) and grid-based planning algorithms constitute an important branch of the motion planning discipline. A very common and successfully applied family of planning algorithms is the one based on the A* algorithm employing a heuristic search. Ferguson, Likhachev and Stentz [9] give an essential overview over heuristic path planning.

The A* algorithm and the concept of considering both known (currently computed) and estimated cost to guide a search towards a solution without necessarily examining the entire state space is the key concept for many extensions that are being researched until today. Only the most promising states (or nodes) in a graph representing the configuration space are expanded. The employed key function $f(s) = g(s) + h(s)$ adds the currently computed cost $g(s)$ of being in state s to an estimated cost $h(s)$ (the heuristic function) from that state towards a goal state. The idea is often realised using a priority queue `open` implementing a priority order defined by this key function.

An excerpt of the basic forward A* implementation[16] computing trajectories in a given environment is shown in Listing 1. This implementation has been derived from the one presented by Nash and Koenig [25].

```
public class ForwardAStarPlanner extends AbstractPlanner {
    private PriorityQueue<Waypoint> open =
        new PriorityQueue<Waypoint>();
    private Set<Waypoint> closed =
        new HashSet<Waypoint>();
    private Waypoint start = null;
    private Waypoint goal = null;
    private LinkedList<Waypoint> plan =
        new LinkedList<Waypoint>();

    public ForwardAStarPlanner(
        Aircraft aircraft,
        Environment environment) {
        super(aircraft, environment);
    }
    //...
    public Trajectory plan(Position origin,
                           Position destination,
                           ZonedDateTime etd) {
        this.open.clear();
        this.closed.clear();
        this.plan.clear();
        this.start = new Waypoint(origin);
        this.start.setG(0);
        this.start.setH(this.getEnvironment()
            .getNormalizedDistance(origin, destination));
        this.start.setEto(etd);
        this.goal = new Waypoint(destination);
        this.goal.setH(0);
        Set<PrecisionPosition> goalRegion =
            this.getEnvironment()
            .getAdjacentWaypoints(destination)
            .stream()
            .map(PrecisionPosition::new)
            .collect(Collectors.toSet());
        this.open.add(this.start);

        while (null != this.open.peek()) {
            Waypoint source = this.open.poll();
            this.setWaypoint(source);
            if (source.equals(this.goal)) {
                return this.computeTrajectory(source);
            }
            this.closed.add(source);
            Set<Position> neighbors = this.getEnvironment()
                .getNeighbors(source);
            if (neighbors.isEmpty()) {
                neighbors = this.getEnvironment()
                    .getAdjacentWaypoints(source);
            }
            if (goalRegion.contains(
                source.getPrecisionPosition())) {
                neighbors.add(destination);
            }

            for (Position neighbor : neighbors) {
                Waypoint target = new Waypoint(neighbor);
                if (!closed.contains(target)) {
                    if (open.contains(target)) {
                        Waypoint visited =
                        open.stream()
                        .filter(s -> s.equals(target))
                        .findFirst().get();
                        this.updateWaypoint(source, visited);
                    } else {
                        this.updateWaypoint(source, target);
                    }
                }
            }
        }
        return new Trajectory();
    }}
```

Listing 1. Basic Forward A* Planner (1)

[16]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/ai/astar/ForwardAStarPlanner.java

Notice how any planner such as the basic forward A* planner is constructed from an `Aircraft`[17] featuring `Capabilities`[18] and an `Environment`[19]. The aircraft operates in the environment and the planner has to support both the aircraft and the environment to compute feasible trajectories considering the aircraft capabilities. The concrete environment is a `PlanningGrid`[20] as discussed in Section II-A above. Assume that the planning grid is configured to use its cubic voxel corners with a 26 corner neighborhood relationship. Furthermore, notice the difference between `Position` and `Waypoint`[21]. A position in space is defined by its latitude, longitude and altitude. A waypoint extends a position adding additional attributes such as a waypoint designator, the computed and estimated costs along a computed trajectory, and the estimated and actual times associated with a waypoint.

The planner computes a `Trajectory`[22] which consists of waypoints and is, hence, a 4D extension of a simple 3D `Path` of positions. It requires an estimated time of departure `etd` to compute a trajectory between a given `origin` and `destination`.

Lines 20 to 36 of Listing 1 initialise the data structures and waypoints. The employed heuristic function is the normalised distance of the environment and is a *consistent* heuristic. In case of a planning grid, this distance uses the diameter of the largest (top-level) cubic voxels as a normaliser. The heuristic function and its relation to the cost function is further detailed in Section IV-A below providing the rational for consistency.

Although both the origin and the destination are required to lie within the planning environment, neither has to coincide with any of the voxel corners within the grid. Hence, the smallest voxels containing both positions provide the connecting corners, that is, the adjacent waypoints of the start and goal regions, respectively (lines 30 and 48).

The algorithm proceeds as usual: The highest priority waypoint, as defined by its key function, is expanded towards its neighborhood (lines 39 and 45). Since a consistent heuristic is employed, each waypoint is only expanded at most once (lines 44 and 58). The costs of the newly examined or revisited neighbors are then updated (lines 59 to 67). The expansion is repeated until the goal has been reached (lines 41 to 43) and the found trajectory can be returned.

The actual cost computation along the established trajectory is realised within the methods `updateWaypoint` and `computeCost` as shown in Listing 2. Updating a waypoint `target` includes the assessment of whether or not it can be reached from one of its `source` neighbors at a lower cost (lines 3 to 5) than already found until this point. If this is the case, the priority queue is updated accordingly (lines 6 to 15).

The `computeCost` method computes the estimated cost for the movement between a `source` and a `target` waypoint. The aircraft capabilities determine if and how this trajectory step can be flown. As mentioned in Section II-D above, air data intervals are currently not taken into account to determine aircraft performance depending on the environment and are assumed to be static (lines 22 and 23).

```
1  protected void updateWaypoint(Waypoint source,
2                                Waypoint target) {
3      double gOld = target.getG();
4      this.computeCost(source, target);
5      if (target.getG() < gOld) {
6          if (!this.open.contains(target)) {
7              target.setH(this.getEnvironment()
8                  .getNormalizedDistance(target, this.goal));
9              this.open.add(target);
10         } else {
11             // priority queue requires re-insertion
12             // of modified object
13             this.open.remove(target);
14             this.open.add(target);
15         }
16     }
17 }
18
19 protected void computeCost(Waypoint source,
20                            Waypoint target) {
21     Path leg = new Path(source, target);
22     Capabilities capabilities = this.getAircraft()
23         .getCapabilities();
24     Globe globe = this.getEnvironment().getGlobe();
25     ZonedDateTime end = capabilities
26         .getEstimatedTime(leg, globe, source.getEto());
27
28     double cost = this.getEnvironment().getStepCost(
29             source, target,
30             source.getEto(), end,
31             this.getCostPolicy(), this.getRiskPolicy());
32
33     if ((source.getG() + cost) < target.getG()) {
34         target.setParent(source);
35         target.setG(source.getG() + cost);
36         target.setEto(end);
37     }
38 }
```

Listing 2. Basic Forward A* Planner (2)

The computed passage duration from source to target waypoint (line 25) according to the aircraft capabilities are finally used to determine the cost of the passage (line 28 to 31). The `getStepCost` method of an environment such as the planning grid is the actual core of the computation. It looks up all relevant cost intervals which are stored in the cost interval trees (as described in Section II-C) of the voxels which are adjacent to the performed movement step. The applicable cost intervals are then combined using a cost function with a specified cost policy and risk policy to determine the final step costs.

A. Cost Function

The performance of a heuristic algorithm such as any A*-based search algorithm, depends significantly on the quality of the employed heuristic. A good heuristic relaxes the actual problem as little as possible to obtain a lower bound for the actual total cost of a state. A heuristic is said to be *admissible* if it does not overestimate the actual costs, or more formally (where c^* is the optimal cost function):

[17]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/aircraft/Aircraft.java

[18]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/aircraft/Capabilities.java

[19]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/planning/Environment.java

[20]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/planning/PlanningGrid.java

[21]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/planning/Waypoint.java

[22]https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/planning/Trajectory.java

$$\text{h is admissible} \iff \forall s, s_{goal} \in S \bullet h(s) \leq c^*(s, s_{goal})$$

A* is guaranteed to be optimal if an admissible heuristic is employed. An example for a very simple heuristic often used in basic path planning is the Euclidean distance from a state to the goal state.

A heuristic can furthermore be *consistent*, that is, the actual cost to an arbitrary direct successor state is never overestimated, or more formally:

$$\text{h is consistent} \iff \forall s, s' \in S, s' \in succ(s) \bullet h(s) \leq c(s, s') + h(s')$$

A consistent heuristic ensures that each state is expanded at most once during a search which further improves the algorithmic performance.

```
1  public double getStepCost(
2          Position origin, Position destination,
3          ZonedDateTime start, ZonedDateTime end,
4          CostPolicy costPolicy, RiskPolicy riskPolicy) {
5
6      double stepCost = 0d;
7
8      // compute participating cells
9      Set<? extends PlanningGrid> segmentCells =
10         this.lookupCells(origin);
11     segmentCells.retainAll(this.lookupCells(destination));
12
13     // an invalid step results in infinite costs
14     if (segmentCells.isEmpty()) {
15         return Double.POSITIVE_INFINITY;
16     }
17
18     List<Double> costs = new ArrayList<Double>();
19     double distance =
20         this.getNormalizedDistance(origin, destination);
21
22     // compute cost of each adjacent cell
23     for (PlanningGrid segmentCell : segmentCells) {
24         // add all (weighted) cost of the cell
25         double cellCost = segmentCell.getCost(start, end);
26         // boost cell cost if local risk is not acceptable
27         if (riskPolicy.satisfies(cellCost - 1)) {
28             costs.add(distance * cellCost);
29         } else {
30             costs.add(Double.POSITIVE_INFINITY);
31         }
32     }
33
34     // apply cost policy for final cost
35     switch (costPolicy) {
36     case MINIMUM:
37         stepCost = costs.stream()
38             .mapToDouble(Double::doubleValue)
39             .min().getAsDouble();
40         break;
41     case MAXIMUM:
42         stepCost = costs.stream()
43             .mapToDouble(Double::doubleValue)
44             .max().getAsDouble();
45         break;
46     case AVERAGE:
47         stepCost = costs.stream()
48             .mapToDouble(Double::doubleValue)
49             .average().getAsDouble();
50         break;
51     }
52
53     return stepCost;
54 }
```

Listing 3. Cost Function (1)

The heuristic function of a SAFCS environment is the normalised Euclidean distance using an environment-specific normaliser such as the longest (top-level) cubic diameter of the planning grid voxels. The step cost is derived by multiplying the normalised distance of the step with the applicable costs (≥ 1) as shown in Listing 3 for a planning grid. Consider that instead of distances, (normalised) passage times could also serve as the basis for the computation of estimated costs. It might even be more appropriate to use the duration being exposed to a certain cost (risk) than the distance which is to be covered through that space.

The affected grid voxels (cells) are found first (lines 9 to 11). This operation is very efficient since range checks can be used to lookup the affected cells. The computational complexity obviously increases with a finer resolution of the planning grid but the same range checks can be performed recursively. Each affected voxel aggregates its own cost interval tree and the individual costs can be collected (lines 23 to 32).

```
1  public double getCost(
2      ZonedDateTime start,
3      ZonedDateTime end) {
4
5      double cost = 1d;
6      Set<String> costIntervalIds = new HashSet<String>();
7
8      // add all (weighted) cost of the cell
9      List<Interval<ChronoZonedDateTime<?>>> intervals =
10         this.getCostIntervals(start, end);
11     for (Interval<ChronoZonedDateTime<?>>
12         interval : intervals) {
13         if (interval instanceof CostInterval) {
14             CostInterval costInterval =
15                 (CostInterval) interval;
16             if (!costIntervalIds
17                 .contains(costInterval.getId())) {
18                 costIntervalIds.add(costInterval.getId());
19                 if ((interval instanceof
20                     WeightedCostInterval)) {
21                     cost += ((WeightedCostInterval)
22                         interval).getWeightedCost();
23                 } else {
24                     cost += costInterval.getCost();
25                 }
26             }
27         }
28     }
29 }
```

Listing 4. Cost Function (2)

The `getCost` method shown in Listing 4 accumulates the actual weighted costs of the overlapping intervals which are active during the query (voxel passage) interval from `start` to `end` (lines 9 and 10). Since interpolated data items describing the same SWIM data may have been embedded before, it is important only to add embeddings with different identifiers (lines 13 to 27). Associating properly weighted costs to the available SWIM data is one of the key challanges and opportunities for tuning when setting up the planning infrastructure.

The combination of the applicable collected costs depends on the specified cost and risk policies. Both are important input parameters that describe the character of an airborne mission and have to be set for a planner (or otherwise the default policies are being used).

B. Cost Policy

The cost policy[23] of a planner expresses how optimistic or pessimistic the planner assesses its environment. In case of a planning grid, a step through the environment may be affected by a number of voxels of possibly different resolutions. These voxels aggregate their own cost interval trees and consequently may incur different costs for the same time interval.

The planner may choose to assume the minimum, maximum or average costs depending on the nature of the airborne mission (lines 34 to 51) or the intentions of an autonomic manager. Different cost policies in turn lead to different trajectories.

C. Risk Policy

The risk policy[24] of a planner determines how much local risk the planner is willing to take. Consider the example of a far away but safe bridge to cross a dangerous creek. The direct passage through the creek may incur a very high cost for a short distance whereas the bridge detour may result in a higher total costs for a long distance. Depending on the risk policy, the planner may choose to accept the very high local cost to minimise the total cost of the computed trajectory. A more cautious policy may instruct the planner to avoid the higher local risk instead.

The SAFCS planners support different local risk policies (cost thresholds) which need to be satisfied to consider or otherwise disregard a costly step. The available risk policies with increasing risk are `AVOIDANCE`, `PROBE`, `SAFETY`, `EFFECTIVENESS`, and `IGNORANCE`. A risk policy has to be satisfied or otherwise the step costs are boosted to an unacceptable level (lines 27 to 31).

V. Planning Extensions

Many extensions within the A* family of algorithms address different planning requirements. Algorithms have been developed that feature *dynamic* (*incremental*, *repairing*), *anytime*, *online* (*interleaved*) and *any-angle* (obeying kinodynamic constraints or allowing for post-processing) properties.

A dynamic planner is able to cope efficiently with a dynamic state (agent environment and capabilities). Only invalid parts (or at least a reduced set) of previously computed solutions are disregarded and re-computed. The ability to repair inconsistencies dynamically instead of computing solutions from scratch is an important advantage when dealing with high dimensionality and limited deliberation time. Dynamic planners implicitly deal with imperfect or incomplete knowledge. The concept has been addressed by Koenig and Likhachev [18] in their D* and D* Lite variants of A*. They formalise the concept of an *inconsistent* state which can be expanded and leads to a cost update during the expansion step. Inconsistency can manifest itself in a state being either *over-consistent* or *under-consistent*. Over-consistency implies that the currently computed cost of a state can be improved. Hence, all states in the `open` queue of the basic A* can be considered to be over-consistent. An under-consistent state in contrast possesses a better cost value than the updated cost an expansion would yield. This situation occurs if costs change (increase) dynamically. D* and D* Lite repair these inconsistencies without disregarding the current solution completely.

An anytime planner returns a first and feasible solution as quickly as possible. This solution is potentially highly suboptimal but sub-optimality can ideally be bounded. As long as deliberation limitations permit, the planner incrementally improves its solutions and ideally decreases the sub-optimality bound with each increment. Thus, an anytime planner resembles nicely the human concept of *bounded rationality*: The more deliberation time available, the better the solution obtained. Any computation of an algorithm is also subject to *bounded reactivity* which implies that instantaneous failure is always possible and can only be contained in the best case. The extensions described by Likhachev, Ferguson, Gordon, Stentz and Thrun [22], [21] describe an anytime dynamic variant of A*. By combining an incremental deflation of an initially inflated (inadmissible) heuristic (Weighted A*) with lifelong planning (Koenig, Likhachev and Furcy [19]) and dynamic repairing techniques (Likhachev, Gordon and Thrun [23]), they created the first AD* variant of the algorithm.

An online planner allows to interleave planning and execution. While the agent performs planned actions, the planner uses the next expected state as a starting point for a revised plan. This approach can be agent-centered if the available sensor range or deliberation time is limited. The extensions mentioned above are usually implemented as online algorithms proceeding backwards from the goal to the next position of the agent.

When employing classical grid search the resulting path may be unnecessary conservative and not implicitly consider any non-holonomic or kinodynamic constraints applicable to the mobile agent. Any-angle algorithms are based on approximate cell decomposition such as grids but compute paths that are not constrained to the planning grid (and adjacent grid nodes). These algorithms avoid the computation of a more expensive roadmap such as a visibility graph or Voronoi diagram. Although efficient algorithms exist (e.g., sweep or beach line paradigm algorithms such as Fortune's algorithm) to compute the latter in configuration spaces with lower dimensions, comparably efficient algorithms are unavailable for higher dimensions.

Ferguson and Stentz [10], [11] developed Field D* – an any-angle path planning algorithm operating on 2D planning grids. Field D* uses interpolation of grid node costs in order to estimate costs for arbitrary points on the edges of the grid. The resulting paths are consequently not restricted to the grid edges but can be any-angle. Their Field D* algorithm presented is based on D* Lite and is, thus, an incremental algorithm that repairs changing costs and is suitable for dynamically changing environments. Field D* has subsequently been extended to 3D grids by Carsten, Ferguson and Stentz [5] and multi-

[23] https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/planning/CostPolicy.java

[24] https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/planning/RiskPolicy.java

resolution grids by Ferguson and Stentz [14].

The second stream of any-angle algorithms applies some form of pulling strings. These algorithms eliminate intermediate nodes within a computed path and, thus, connect non-neighboring nodes directly (which looks like pulling a string that connected the nodes indirectly before the operation). Nash and Koenig [25] developed their Theta* family of algorithms which allow to connect the parent of the currently expanded node with all of its successors if they are visible from each other. Hence, it requires an implicit (more costly) visibility test and the resulting paths do not need to follow the grid edges. Theta* has already been implemented[25] and tested based on planning grids with cost interval trees. The extension will be discussed in a separate article.

Nash, Koenig and Tovey [26] address the issues of reducing the number of visibility checks and cost estimation in 3D with Lazy Theta*. Lazy Theta* reduces the number of visibility checks by assuming (collision-free) visibility between the parent (currently best predecessor) and the successors of an expanded node first, and instead performing visibility checks only for expanded vertices (correcting any visibility issues if any). The approach trades off a significant number of visibility checks with a higher number of expanded nodes while improving the algorithmic performance.

Other extensions cope with the problem of uncertain environments such as probabilistic planning with clear preferences on missing information by Likhachev and Stentz [24].

The presented environment and capabilities model should be amenable to many of the extensions to the A* family (and other non-deterministic planning algorithms) which leads to the larger scope of the SAFCS framework discussed in Section VII below.

VI. Evaluation

Members of the A* family of algorithms have already been extensively evaluated by the referenced authors with with respect to the applied heuristics, the required number of expansions, free-space heading changes, the sub-optimality of solutions etc.. The contribution of this article lies in the embedding of continuous time interval SWIM data into the voxels of a multi-resolution planning grid. The associated cost intervals are discretised by considering the passage time interval of the agent through individual voxels. Consequently, the effects of voxels aggregating cost intervals should be discussed further.

A. Performance

Employing an interval (annotated) tree to store temporality-enabled SWIM data results in an $\mathcal{O}(n)$ space complexity since there is one node for each stored cost interval. However, instead of maintaining one cost interval tree which could reference all affected voxels for each interval, a choice was made to maintain (potentially) smaller trees for each voxel. The number of these interval trees increases with the number of voxels and could become a limiting factor with very high-resolution grids. It is very likely that many cost intervals are stored redundantly in neighboring voxels.

The presented approach however trades off the additional space complexity for an improved runtime complexity during the lookup operation. The lookup of applicable cost intervals happens at a much higher frequency during each expansion step that a planner performs. Performing frequent lookups with $\mathcal{O}(\log n + m)$ time complexity benefits from keeping the trees and consequently n smaller. Considering that the planning infrastructure is not required to be deployed in the airborne system but instead may reside anywhere in the cloud justifies that decision.

Available SWIM data may grow quickly and cover large areas of space and time. Past cost intervals could be retained for later analysis or otherwise removed to maintain an acceptable planning performance. Today flight data recorders and cockpit voice recorders are being used to investigate aviation incidents and accidents. Being able to replay past situations and resulting decisions might be an important aspect to consider.

Another potential performance issue arises from using cubic grids as a planning environment on a sphere model representing our planet. The planning grids have been implemented based on hierarchical cubes which are geometric extents in the NASA Worldwind SDK. These extents are constructed using Cartesian coordinates. Frequent coordinate transformations are currently performed between the planning positions in spherical coordinates and the Cartesian coordinate system. A much better solution would be to perform only one transformation before and after the invocation of the planning algorithm. The entire planning algorithm would then be based on Cartesian coordinates.

A last issue which should be mentioned are numerical inaccuracies during the computation of embeddings, affected voxels and neighbors. A precision-type hierarchy has been implemented[26] to expand positions and points and include potentially affected areas of the search space. This may obviously lead to a more conservative planning environment and should probably be analysed and tuned carefully.

B. Simulations

Several simulations have been performed using artificially created IWXXM data and some videos have been published[27] demonstrating selected features of the SAFCS framework. A JavaFX based user interface has been developed[28] to enable the configuration of airborne missions and establish a datalink to a small unmanned aerial vehicle. The datalink feature enables actual flight tests and the revision of planned trajectories during the flight.

The first tests of the infrastructure seem promising. Cost-optimal trajectories such as the one shown in Figure 5 are

[25] https://github.com/stephanheinemann/worldwind/blob/master/src/main/java/com/cfar/swim/worldwind/ai/thetastar/ThetaStarPlanner.java

[26] https://github.com/stephanheinemann/worldwind/tree/master/src/main/java/com/cfar/swim/worldwind/geom/precision

[27] http://www.rigiresearch.com/people/stephan-heinemann

[28] https://github.com/stephanheinemann/worldwind-ui

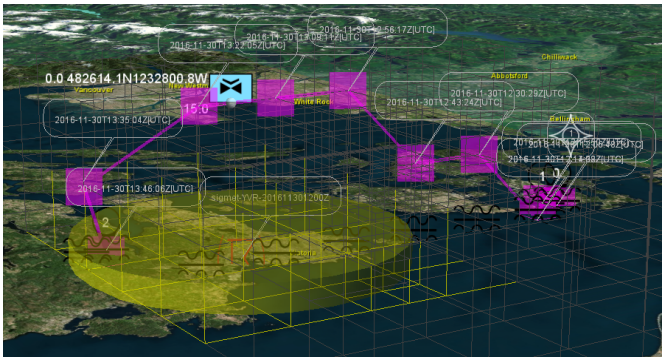

Fig. 5. Planning Scenario

computed within reasonable times. However, more data is required and will be collected with respect to the amount of SWIM data (cost interval tree sizes) and the resolution of the environment. In particular, the AD* extension discussed in Section V is going to be an interesting candidate when stressing the system with scenarios of limited deliberation times.

VII. Future Work

The presented planning infrastructure and algorithms are part of the larger SAFCS framework which shall enable higher levels of decision making and support functionalities for airborne missions. Many of the available A* extensions mentioned under Section V above shall be implemented and evaluated with situational SWIM data and different types of roadmaps. These roadmaps are not limited to planning grids but also include published VFR and IFR roadmaps available via AIXM. Non-deterministic algorithms such as Probabilistic Roadmaps (PRM) (Kavraki et al. [16]) and Rapidly-Exploring Random Trees (RRT) (LaValle and Kuffner [20], [15]) with their available dynamic and anytime extensions (Belgith et al. [4], [3], Ferguson, Kalra and Stentz [8], [12], [13]) shall also be evaluated.

The SAFCS shall be studied as an autonomic system based on the MAPE-K architecture described by Kephart and Chess [17]. Employing this architecture, a pool of planning environments and algorithms could be controlled and assessed online while continuously improving a solution for a particular planning problem. Both the environment and the algorithms can be selected and tuned (e.g., environment resolution and algorithmic tuning parameters) by an autonomic manager collecting statistics about planning performance in certain situations. The envisioned research shall lead to a self-adapting planning infrastructure which selects and tunes environments and algorithms to achieve the best possible performance for certain scenarios.

Topics to be addressed furthermore include the implementation of business rules (e.g., certain rules of the air), the communication with human actors in the loop (e.g., embedded airspaces with controlled access), and the assessment of the criticality of the situation resulting in a suitable level of autonomy and interaction.

VIII. Conclusion

In this article we have presented how aviation data available via the FAA System-Wide Information Management (SWIM) infrastructure can be embedded into grid-based planning environments using interval trees. Interval trees can be employed to store costs associated with temporality-enabled SWIM data and aggregated by voxels of a planning grid. We have discussed how the environment, the aircraft capabilities, and selected cost and risk policies influence the computed trajectories. We have provided a brief overview over the online, dynamic, anytime and any-angle extensions within the A* family of algorithms and their applicability within the larger context of the Smart Autoflight Control System framework. Classical grid search is going to be one approach in a pool of suitable environments and algorithms this framework will be able to select from.

References

[1] *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, October 9-15, 2006, Beijing, China*. IEEE, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4058334

[2] *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*. IEEE, 2006.

[3] K. Belghith, F. Kabanza, and L. Hartman, "Randomized path planning with preferences in highly complex dynamic environments," *Robotica*, vol. 31, no. 8, pp. 1195–1208, 2013. [Online]. Available: http://dx.doi.org/10.1017/S0263574713000428

[4] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou, "Anytime dynamic path-planning with flexible probabilistic roadmaps," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*. IEEE, 2006, pp. 2372–2377. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2006.1642057

[5] J. Carsten, D. Ferguson, and A. Stentz, "3d field D: improved path planning and replanning in three dimensions," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, October 9-15, 2006, Beijing, China*. IEEE, 2006, pp. 3381–3386. [Online]. Available: http://dx.doi.org/10.1109/IROS.2006.282516

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3. ed.)*. MIT Press, 2009. [Online]. Available: http://mitpress.mit.edu/books/introduction-algorithms

[7] C. Ericson, *Real-Time Collision Detection*. Boca Raton, FL, USA: CRC Press, Inc., 2004.

[8] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, May 15-19, 2006, Orlando, Florida, USA*. IEEE, 2006, pp. 1243–1248. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2006.1641879

[9] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, pp. 9–18, 2005.

[10] D. Ferguson and A. Stentz, "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-19*, 2005.

[11] ——, "Field d*: An interpolation-based path planner and replanner," in *Robotics Research: Results of the 12th International Symposium, ISRR 2005, October 12-15, 2005, San Francisco, CA, USA*, ser. Springer Tracts in Advanced Robotics, S. Thrun, R. A. Brooks, and H. F. Durrant-Whyte, Eds., vol. 28. Springer, 2005, pp. 239–253. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-48113-3_22

[12] ——, "Anytime rrts," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, October 9-15, 2006, Beijing, China*. IEEE, 2006, pp. 5369–5375. [Online]. Available: http://dx.doi.org/10.1109/IROS.2006.282100

[13] ——, "Anytime, dynamic planning in high-dimensional search spaces," in *2007 IEEE International Conference on Robotics and Automation, ICRA 2007, 10-14 April 2007, Roma, Italy*. IEEE, 2007, pp. 1310–1315. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2007.363166

[14] D. I. Ferguson and A. Stentz, "Multi-resolution field D," in *Intelligent Autonomous Systems 9 - IAS-9, Proceedings of the 9th International Conference on Intelligent Autonomous Systems, University of Tokyo, Tokyo, Japan, March 7-9, 2006*, T. Arai, R. Pfeifer, T. R. Balch, and H. Yokoi, Eds. IOS Press, 2006, pp. 65–74.

[15] J. J. K. Jr. and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, April 24-28, 2000, San Francisco, CA, USA*. IEEE, 2000, pp. 995–1001. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2000.844730

[16] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE T. Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. [Online]. Available: http://dx.doi.org/10.1109/70.508439

[17] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[18] S. Koenig and M. Likhachev, "D*lite," in *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, R. Dechter and R. S. Sutton, Eds. AAAI Press / The MIT Press, 2002, pp. 476–483. [Online]. Available: http://www.aaai.org/Library/AAAI/2002/aaai02-072.php

[19] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A," *Artif. Intell.*, vol. 155, no. 1-2, pp. 93–146, 2004. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2003.12.001

[20] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, K. Donald, B.; Lynch and e. Rus, D., Eds., 2001, pp. 293–308.

[21] M. Likhachev, D. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artif. Intell.*, vol. 172, no. 14, pp. 1613–1643, 2008. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2007.11.009

[22] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm," in *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, S. Biundo, K. L. Myers, and K. Rajan, Eds. AAAI, 2005, pp. 262–271. [Online]. Available: http://www.aaai.org/Library/ICAPS/2005/icaps05-027.php

[23] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2003, pp. 767–774. [Online]. Available: http://papers.nips.cc/paper/2382-ara-anytime-a-with-provable-bounds-on-sub-optimality

[24] M. Likhachev and A. Stentz, "Probabilistic planning with clear preferences on missing information," *Artif. Intell.*, vol. 173, no. 5-6, pp. 696–721, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2008.10.014

[25] A. Nash and S. Koenig, "Any-angle path planning," *AI Magazine*, vol. 34, no. 4, p. 9, 2013. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2512

[26] A. Nash, S. Koenig, and C. A. Tovey, "Lazy theta*: Any-angle path planning and path length analysis in 3d," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, M. Fox and D. Poole, Eds. AAAI Press, 2010. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1930