3D Path Planning for Autonomous Aerial Vehicles in Constrained Spaces

Schøler, Flemming

Publication date:
2012

Document Version
Early version, also known as pre-print

Flemming Schøler

3D Path Planning
for Autonomous Aerial Vehicles in Constrained Spaces

3D Path Planning for Autonomous Aerial Vehicles in Constrained Spaces
Ph.D. thesis

ISBN: 978-87-92328-73-1
January 2012

Contents

Preface

This thesis is submitted as partly fulfillment of the requirements for the Doctor of Philosophy at the Section of Automation and Control, Department of Electronic Systems, Aalborg University, Denmark. The work has been carried out in the period September 2008 to January 2012 under the supervision of Associate Professor Anders la Cour-Harbo and Assistant Professor Morten Bisgaard.

Anders has been the ideal main supervisor. His engagement, enthusiasm and encouragement were an aid in innumerable ways. I also greatly appreciate Morten for his theoretical and technical advice.

I was visiting Naval Postgraduate School during the summer and autumn of 2010. I would like to thank Professor Michael Ross and Dr. Mark Karpenko for their persistence in making this stay possible and ensuring that I had a pleasant and enjoyable time.

I thank my family, friends, and colleagues for all the support, and especially my fellow PhD-students for our greatly valued discussions.

Finally, I dedicate this thesis to Diane for her love and patience during these three years.

Aalborg University, January 2012
Flemming Schøler

Abstract

3D Path Planning for Autonomous Aerial Vehicles in Constrained Spaces

Determining how an autonomous Unmanned Aircraft System (UAS) should reach a goal position amidst obstacles is a challenging and difficult problem. This thesis treats the subject of path planning and trajectory generation for UAS, while utilizing the ability to move in all three spatial dimensions. The primary contributions of this thesis are an approximate path planner and a geodesic path planner. Both planners are model independent and operate on the surface of the configuration space to identify a length minimizing path.

The approximate planner determines an approximated shortest path by building and searching a visibility graph. This planner maintains this visibility graph to enable fast multi-query searches as well as replanning when changes occur in the work space. As paths obtained from a visibility graph are not continuously differentiable, a trajectory generation method is developed that uses the path to find a collision free trajectory that is more appropriate for flight.

The geodesic planner relates to wavefront-type planning, and identifies continuously differentiable geodesic paths as parametric equations determined by surface primitives given from the configuration space. Consequently, this planner uses a more analytical approach since it relies on combinations of optimal curves.

Both planners operate on an explicit description of the configuration space in a work space containing 3D obstacles. A method was developed that generates convex configuration space obstacles from any point clouds or geometric meshes in work space.

Two approaches are used for generating a trajectory from an existing path. The first generator use Dubins curves to find a collision free continuously differentiable trajectory. The second generator relies on formulating and solving an optimal control problem (OCP) using a Legendre pseudospectral method. The main contributions of this approach are the formulation of distance functions that constrains the trajectory. This allows finding trajectories that follows minimal length paths while optimizing the trajectory according to a performance index.

The methods and algorithms developed in this thesis are implemented in a planning application and validated through simulated flight using a helicopter model.

Synopsis

3D ruteplanlægning for autonome luftfartøjer i afgrænset rum

At afgøre hvordan et ubemandet luftfartøjs system (UAS) skal nå en ønsket position blandt forhindringer er et vanskeligt, men kritisk, problem. Denne afhandling afdækker ruteplanlægning og trajektoriegenerering for et UAS hvor dets evne til at bevæge sig i alle tre rumlige dimensioner udnyttes. De primære bidrag fra denne afhandling er hhv. en approksimativ og en geodætisk ruteplanlægger. Begge planlæggere er modeluafhængige og opererer på overfladen af konfigurationsrummet for at finde den korteste rute.

Den approksimative planlægger bestemmer en tilnærmelsesvis korteste rute ved at opbygge og søge i en synlighedsgraf. Planlæggeren vedligeholder denne synlighedsgraf for at gøre efterfølgende forespørgsler samt genplanlægning efter ændringer i det fysiske rum hurtigere. Metoder til trajektoriegenerering udvikles også da ruter fundet i en synlighedsgraf ikke er kontinuerligt differentiable og derfor svre at flge prcist. Disse metoder bruger eksisterende ruter til at finde kollisionsfri trajektorier, som er mere hensigtsmæssige til flyvning.

Den geodætiske planlægger relaterer sig til bølgefront-planlægning, men den finder parametriske ligninger for de kontinuerligt differentiable geodæter. Geodæterne bestemmes af geometriske primitiver givet ud fra konfigurationsrummet. Denne fremgangsmåde er mere analytisk da den baserer sig på en kombination af optimal kurver.

Begge planlæggere opererer på en eksplicit beskrivelse af konfigurationsrummet, der genereres efter et fysiske rum indeholdende 3D forhindringer. En metode blev udviklet, som genererer konvekse forhindringer i konfigurationsrummet fra punktskyer og polyeder i det fysiske rum.

To metoder anvendes til at generere et trajektorie fra en eksisterende rute. Den første metode anvender Dubins kurver til at finde et kollisionsfri kontinuerligt differentiable trajektorie. Den anden metode afhænger af at formulere og løse et optimalt kontrol problem (OCP) gennem en Legendre pseudospectral metode. Det vigtigste bidrag i denne tilgang er formuleringen af afstandsfunktioner, der begrænser trajektoriet. Dette gør det muligt at finde trajektorier, som følger den korteste rute, samtidig med at trajektoriet optimeres efter en omkostningsfunktion.

Metoder og algoritmer udviklet i denne afhandling er implementeret i software og valideret gennem simuleret flyvning ved hjælp af en helikoptermodel.

1 | Introduction

1.1 Background and Motivation

UAS are unpiloted aircrafts that are either remotely controlled or autonomous. They are considered an indispensable platform for many applications. Until recently, UAS were primarily supporting military operations. This is changing rapidly. UAS promise new ways to increase efficiency, decrease costs, enhance safety and even save lives.

Search and rescue operations can benefit from UAS that may assist in finding missing or injured persons, see i.e. [1, 2]. The UAS provide aerial imagery of a search area with the advantage of quick systematically coverage of large areas, access of hard-to-reach or dangerous areas, and lower cost than manned aircrafts. Once the persons are located, the UAS can guide rescue workers to the victims. This helps focus the efforts on the rescue operation instead of the search operation, which can be substantial. Being unmanned they can also operate in dangerous environments such as more strenuous weather conditions, or environments contaminated by chemical, biological, or nuclear materials.

Aerial photography, mapping and inspection are other examples of tasks where a small aircraft is beneficial, since it is able to operate much closer to surfaces and in smaller and more constrained spaces than conventional aircrafts. This enables the UAS to build high resolution elevation or 3D maps. UAS make a good alternative when installations and buildings need to be inspected. UAS could also inspect high voltage electrical lines in remote locations, or structures such as bridges, dams, wind turbines, offshore oil-rigs or other structures that are not easily accessible.

Emergency assistance and law enforcement are amongst other potential civil uses. An advantage is that a UAS equipped with cameras and other equipment can send back information to assist in getting an overview, and such recorded data can subsequently be used for evaluation. They are also not stopped by roads that are blocked from damage, debris, or traffic which might occur after natural disaster such as a tsunami, earthquakes or a hurricane. They can also assist in more frequently occurring situations, where a main advantage is that small fast aircraft can take off instantly and arrive at the scene at initial critical time before other vehicles. The can provide information critical to the manned part of the operation as it is on route to, for instance, a building fire or a robbery. In the former case it can help give an overview of amount of relief required, alert residents, and look for people in need through windows. A UAS equipped with an infrared camera can see through smoke. In the latter example it can be beneficial to at arrive at the scene silently to get video evidence of perpetrators, and also follow them. Video can in both examples be used for evidence as to whom or what started the fire or how the robbery was

carried out.

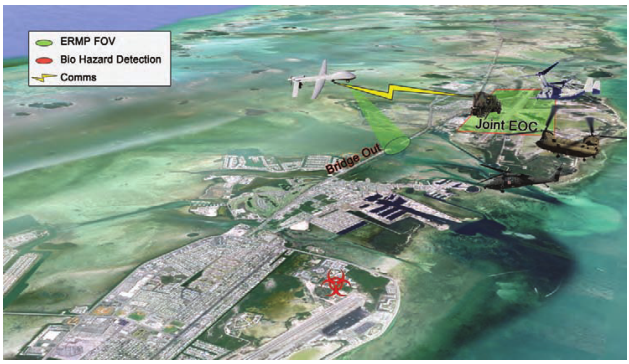


Figure 1.1: UAS used for remote sensing and search and rescue after a hurricane strike. From U.S. Army Roadmap for Unmanned Aircraft Systems 2010-2035 [3].

The U.S. Army Roadmap for Unmanned Aircraft Systems 2010-2035 [3] includes plans to use UAS for operations that address the consequences of natural disasters and man-made accidents. Such a system must have various capabilities that enable it to handle situations similar to that illustrated in Figure 1.1. A highway bridge and a biohazard storage facility are damaged after a hurricane strike leaving people stranded in the affected area. Because of the risks of a biohazard leak an unmanned operation is used in favor of manned operations. An extended range multi purpose (ERMP) UAS is used for providing incident awareness and assessment, for enabling search and rescue, and for communications in the area. Its operation is managed by an emergency operations center (EOC) that is established outside the area. The aircraft can transmit video back to the operations center, as well as data from equipment that performs remote sensing of biohazardious material. The UAS initially covers the affected area, allowing for immediate action on 3 the biohazard and search and rescue efforts. It also performs communications relay missions to facilitate command between maritime effort, the operations center, staging areas, and other UAS operations. After these operations, the UAS remains on station to provide situational awareness.

Most current operational UAS are either remotely controlled by a human operator or operating autonomously in an obstacle-free environment. There is a need of improving the autonomy of UAS to facilitate future capabilities. Autonomy in robotics is commonly defined as the ability to make decisions without human intervention. This discipline aims at teaching robots intelligent behavior. However, the market for autonomy for UAS is still undeveloped and remains a bottleneck for UAS developments.

1.2 Objective

There are many aspects to consider when developing a UAS. This covers mechanics, electronics, software, operation procedures, contingency planning, safety, permits, mission planning and numerous other aspects. Some aspects are shown in the UAS block in Figure 1.2. Improvements to future UAS could be largely driven by the advances

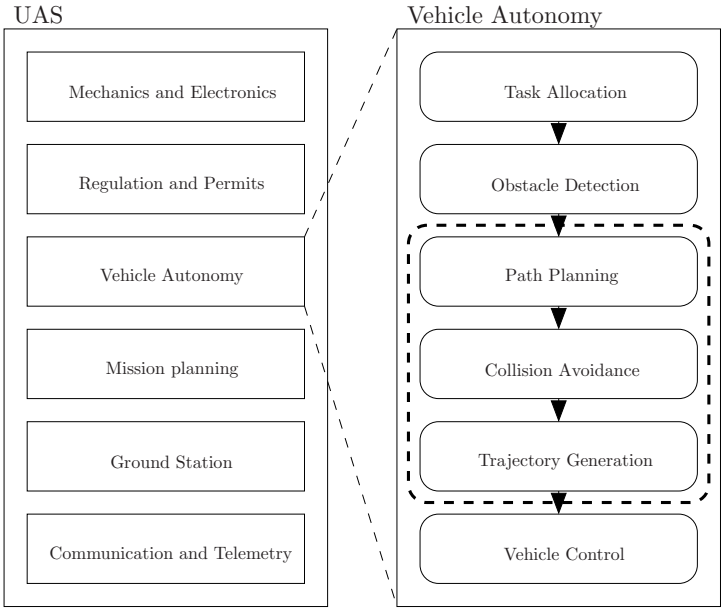


Figure 1.2: A UAS consists of numerous components. Some of the more important ones are shown above. One main component is vehicle autonomy. That is the ability of the aircraft to operate without human intervention and continuous control.

made in autonomy [4]. Consequently, there is much focus on the improving planning and control architecture [5], including task allocation, path planning, and vehicle control. However, dealing with all the aspects simultaneously is very complex, especially since different parts of the planning process must react to changes happening at different rates and times. A widely used strategy is to decouple the overall architecture into a hierarchy of less complex problems [6–10]. Such decoupling can i.e. lead to a high level task allocation layer [11–13] that decides on a goal, a path planning layer [14–16] that determines waypoints to visited to reach the goal while avoiding obstacles, a trajectory generating layer [16, 17] that interconnects waypoints with dynamically feasible paths, and a low level vehicle control layer [18–20] that allows the UAS to track a path under disturbances such as wind. This architecture should also have layers to handle collision detection and avoidance during flight. This is split in an obstacle detection and collision avoidance layer in the figure. Higher layers operates on data known in advance and that are changing infrequently, and lower levels handles disturbances, collision detection and avoidance of new obstacles, and operate on vehicle states such as positions and velocities. In this thesis, the task allocation layer is reduced to a single or a list of sequential goal, complete information about obstacles in the environment is assumed, and an aircraft with an existing low level control layer is used. The contributions of this thesis are in the path planning layer, the collision avoidance layer, and the trajectory generation layer. The collision avoidance layer is used to do faster replanning. The trajectory generation layer is used to verify the feasibility of paths. The aim is to develop algorithms that can be used

Figure 1.3: The modified Bergen Industrial Twin.

to plan a feasible path for a small aircraft.

1.2.1 Vehicle Control

The baseline helicopter used for the research is the modified Bergen Industrial Twin seen in Figure 1.3, which is an off-the-shelf radio controlled (RC) helicopter with a 52 cc, 8 HP engine. It has a bounding sphere radius of 1.7 meters. The helicopter has no significant nonholonomic constraints. Consequently, it has the ability to fly in any direction and this is virtually unaffected by its orientation. A rigid body model and a vehicle control system have been derived and implemented in Simulink by Bisgaard [21]. A feedback controller has been designed using linear quadratic regulation (LQR) with the model linearized in hover. The model is the AAU Helisim model[22], a first principles model of the Bergen industrial twin helicopter. It includes second order actuator dynamics and second order flapping dynamics for main rotor and stabilizer bar. The model is simplified to 12 rigid body states with steady state solutions for flapping and actuator dynamics. The states are position and Euler angles and body velocities and angular rates. The model has four control signals, and has been verified against the actual helicopter.

1.2.2 Path Planning

Path planning is a fundamental challenge in robotics. The classical challenge can be formulates as: Given a three-dimensional rigid body with a set of configuration parameters and a known set of stationary obstacles, find a collision-free path from a start configuration to a goal configuration. This formulation is known as the ”piano mover’s problem” [23]. Path planning is relevant to a number of fields including control theory, computational and differential geometry, mechanics, and computer science, and has been the subject of substantial research. Path planning and trajectory generation problems are two distinct parts of robotics that are intimately related. A clear difference exists between a path and a trajectory. A path is without time considerations and a trajectory is. In this work, a path is an ordered set of 3D points, while a (state) trajectory is an set of state that are associated with a time. Some methods, such as the solution to the optimal control problem, also generate a control trajectory associated with the state trajectory.

The presented research deals with the subject of path planning for autonomous aerial vehicles in constrained spaces. While the focus is on aerial vehicles, the developed methods are general for rigid bodies translating in a known constrained 3D space, and most approaches are model-independent, although an agile vehicle capable of fast acceleration and with no significant nonholonomic constraints is assumed. Other uses of such planning could include undersea vehicles. Constraints will mostly be due to fixed obstacles in the form of polyhedrons of any shape and orientation, although in some approaches other constraints are also considered in a dynamic environment.

The basic problem is to find a collision free path inside some space (denoted the work space), so there is a need to identify the locations of the vehicle where no part of it intersects the obstacles. These representations are known as the configurations, and the configuration space is the space of all allowed configurations of the vehicle. The dimension of the configuration space is equal to the number of independent variables in the representation of the configuration, also known as the degrees of freedom (DOF). The UAS has six degrees of freedom: three to represent the position and three to represent the orientation. It is fair to exclude orientation by representing the shape of the vehicle with its bounding sphere in mass-of-mass as there are no significant nonholonomic constraints. The problem is now to find a curve in a 3D configuration space that connects the start and goal points and avoids all configuration space obstacles that arise due to obstacles in the work space.

1.3 Previous Work

Path planning is an active research area, and there are many approaches and algorithms to solve such problems. These are traditionally classified into classes of methods, such as roadmap methods, potential field methods, approximate and exact cell decomposition [24–26]. More recent approaches such as rapidly-exploring random trees (RRTs) and wavefront planning methods are based on the classic approaches. The most appropriate method for a given planning problem depends on several factors. This includes the configuration space, its number of dimensions and how constraints are described. Many methods are limited to two-dimensional spaces with the difficulty increasing rapidly as the number of dimensions increase. A review follows of the classical path planning paradigms and some more recent methods that offspring from the classical ideas. There is a focus on methods that are suitable for the problem presented here.

1.3.1 Potential Fields

The potential field method [27], pictured in Figure 1.4, makes use of a potential field in the configuration space to solve the path planning problem numerically. In the simplest approach vehicle and obstacle are given positive repulsing potential and the goal a negative attracting potential. At each step the vehicle follows the negative gradient towards a minimum, until it reaches a critical point where the gradient vanishes. While this happens at any local minimum, maximum, or saddle point, the use of numerical methods generally means that local maxima and saddlepoints are unstable and only locally minima are reached. The implication is that progression stops at the first reached local minimum, weather or not this is the actual goal.

It is possible to define a potential function with only one minimum [28]. The idea is to make the attracting potential at the goal overwhelm the repulsing potential at the obstacles. This essentially makes the potential field take form towards a steep bowl centered at the goal. As a result, critical points gravitate toward the goal, and local minima turn into unstable saddle points. Unfortunately such potential function can reach arbitrarily large values far from the goal, making it difficult to compute. Modifying the potential field function to flatten such areas makes implementation of a gradient descent approach quite difficult because of numerical errors [25]. Another problem with potential fields is the lack of control of the exact motions of the vehicle, and it is difficult to guarantee that the resulting path is optimal in any particular sense or even collision free path.

An analytical definition of the problem is possible[29], but difficult for general environments. An example is shown in Figure 1.5.

The numerical potential field methods, also known as wavefront planning methods [30, 31], partitions the configuration space into cells in a grid. A numerical potential field with only one local minimum is built on the grid. The planner initializes the grid cells with obstacles with ones and free cells with zero. The cell containing goal is initialized with a two. In the first step, all zero-valued cells neighboring the goal are set to three. Next, all zero-valued cells adjacent to threes are set to four. This procedure continues and essentially grows a wavefront from the goal. The procedure terminates when the wavefront reaches the cell containing start. The planner then uses gradient descent from the start cell to find a path. For an example see Figure 1.6.

Randomized planning strategies [32–34] attempt to escape local minima by initiating

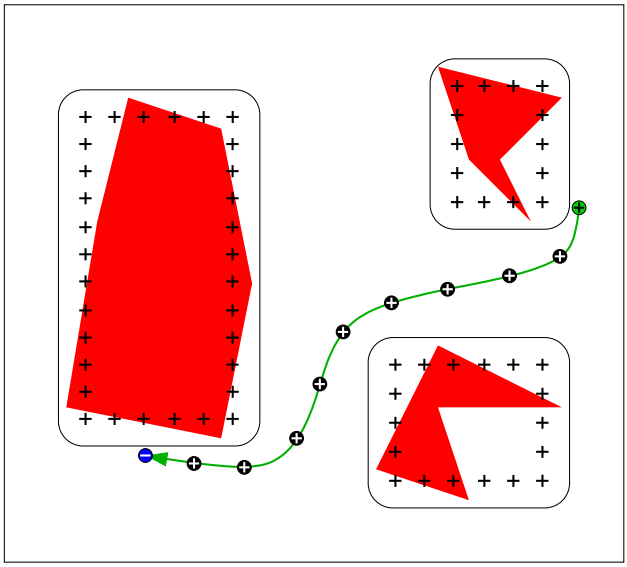


Figure 1.4: The negative potential at the blue goal attracts the vehicle while the positive potentials at the green start and red obstacles repel it. This results in the green path around the obstacles and to the goal. The black nodes show position at each step of the method.

a series of random walks when stuck. This often escapes the local minimum. After an escape the negative gradient is followed again. This randomization strategy is the basis of probabilistic roadmaps [35].

1.3.2 Probabilistic Roadmaps

Roadmaps [36] are data structures that aim at capturing the connectivity of the configuration space. The main idea is to first construct a map that can subsequently be searched for a collision free path. With Probabilistic RoadMaps (PRM) [35] such a map is constructed by sampling points in the configuration space. If a point is outside the configuration space obstacles, they are added to the map (a graph) as nodes. Configurations near each other are then linked by connecting their nodes with an edge if there is a collision free path between them. Once the map is sufficiently dense, a graph search algorithm is used for identifying a path in the graph. Classical methods include Dijkstra's algorithm [37] or A* [38], which achieves better time performance using heuristics. There are many variants on the basic PRM method that vary the sampling strategy and connection strategy to achieve faster performance. See e.g. [39] for more details.

1.3.3 Rapidly-Exploring Random Trees

Path planning with kinematic constraints is PSPACE-Hard [40], but have relative fast average-time performance. PRMs sample many points and then identify which samples in proximity are reachable in order to create a connected graph for multi-query searches. However it can be quite difficult to find connections between samples if significant dy-

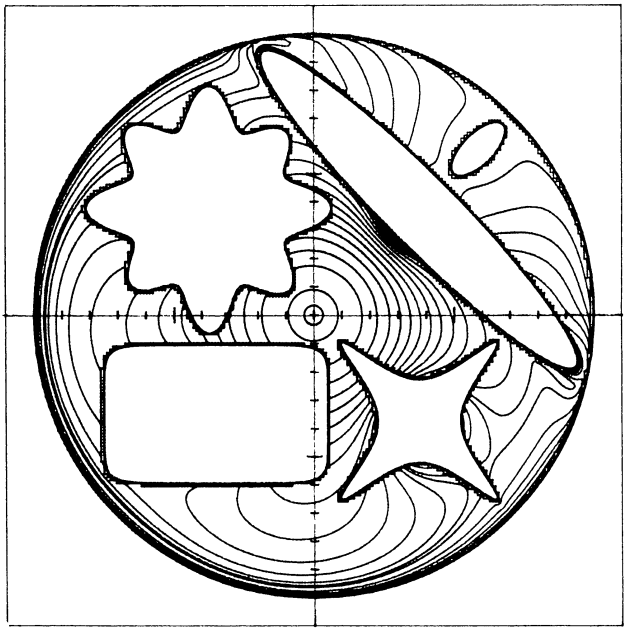


Figure 1.5: Potential function with unique minimum at the destination located in origin. From ”The construction of analytic diffeomorphisms for exact robot navigation on star worlds” [29].

namic constraints exist. In addition, the number of necessary samples increases exponentially with dimensionality of the problem. One notable approach is using Rapidly-exploring Random Trees (RRT) [41–43]. RRTs can be seen as an extension of PRM to include vehicle dynamics. With RRTs, a tree is iteratively expanded by applying control inputs that drive the system slightly toward randomly-selected points. Rapidly-exploring Random Trees (RRTs) are single-query methods geared towards fast path planning. The RRT approach creates a tree in the CS from nodes that represent reachable collision free configurations, and links that represent feasible dynamical transitions between the nearest nodes. The samples are usually found from a uniform distribution inside a bounded area of the CS with some bias towards the goal, and Euclidean length is usually used as metric. The root node of the tree is the initial configuration and it grows until the goal configuration is reached. The RRT algorithm is efficient because it introduces a Voronoi bias, that encourages it to explore unexplored regions.

1.3.4 Voronoi Diagrams

A path can be found with a Voronoi diagram that keeps equal distance to the nearest obstacles at all times. This approach gives the maximum clearance roadmap [26].

The Voronoi diagram method partitions the free CS is into generalized Voronoi regions, where points in the interior of a region are closer to a particular obstacle than to any other obstacles. The regions are bounded by points that are equidistant to the nearest

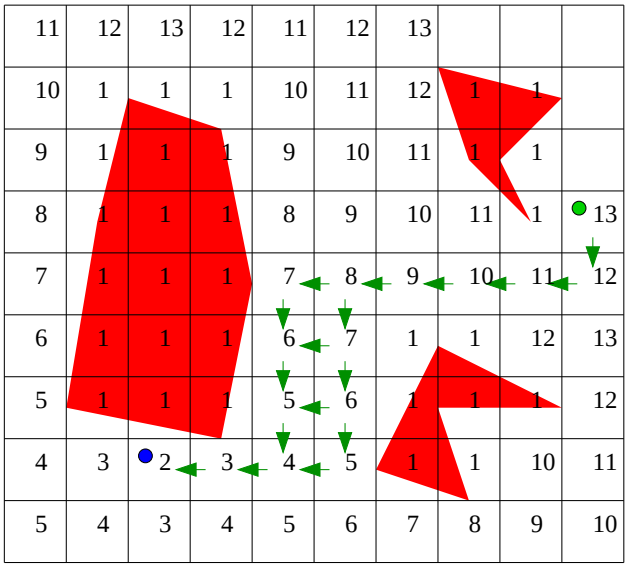


Figure 1.6: The wavefront planning method builds a roadmap while continuously propagating a wavefront from the blue goal cell to zero-cells directly above, beneath, or on the sides. A path can be found once the start cell is reached by following a green descending gradient to a neighboring cell until the goal cell is reached. There can be multiple paths that visit an equal number of cells.

CS obstacles. A generalized Voronoi region is the set of such equidistant points. In 2D, the points are equidistant to two obstacles so the Voronoi regions are convex polygons and the Generalized Voronoi Diagram (GVD) forms a 'two-equidistant surface'. In 3D, the regions are bounded by points that are equidistant to the three nearest CS obstacles and the Voronoi regions are polytopes (convex polyhedrons). Since a two-equidistant surface pierces obstacles, it is restricted to the set of points that is both equidistant to two particular obstacles and have them as their closest obstacles. This restricted structure is the two-equidistant face.

In 2D the GVD is a one-dimensional union of edges with endpoints that are either equidistant to three or more obstacles, or intersect the nearest obstacles. The GVD is connected, meaning that there exist a path between any two points in the GVD if and only if there exists a path between these two points in the CS. Path planning is achieved by moving away from the closest obstacle until reaching the GVD, then along the double equidistant GVD to the vicinity of the goal, and then from the GVD to the goal.

In 3D the GVD is two-dimensional and the path planning is still difficult. In 2D two two-equidistant faces intersect and form a one-dimensional manifold. The union of these one-dimensional structures is termed the Generalized Voronoi Graph (GVG) [44, 45]. The GVG and the GVD coincide in 2D. The GVG edges in 3D are the set of points equidistant to three obstacles with these three obstacles being closest. However in 3D the GVG is not a roadmap because it is not always connected. However the two-dimensional GVD is connected and this connectivity can be used to "patch together" the GVG. Connect-

ing different components is tedious and challenging [25]. For algorithms that compute Voronoi diagrams, see [46, 47].

1.3.5 Visibility Graph

The standard visibility graph (VG) is defined in a 2D polygonal CS [14, 48]. This graph can be searched to yield a minimum length path. The nodes of the graph include the start location, the goal location, and the vertices of the configuration space obstacles on the convex hull. The graph edges are straight-line segments that connect pairs of nodes, if they are in line-of-sight of each other. The reduced visibility graph [24] reduces the size of such a graph, by generating it from only supporting and separating line segments. For convex obstacles these are segments of lines that only touch obstacles at the node vertices. Segments on the hull of an obstacle are also permitted. A reduced VG can be seen in Figure 1.7. VGs can be used to determine the Euclidean shortest path of a vehicle moving amongst polygons. Although the notion of Euclidean shortest path is straightforward both in 2D and 3D space, exact algorithms exist only in the 2D case. While it is clear that the shortest path is through the surface of the polytope in both cases, these methods unfortunately do not extend nicely to 3D. The main problem in 3D is that the shortest path does not in general traverse only vertices of the convex polytope, as in the 2D case, but also points on the edges of the polytopes. It is usually not apparent where on the edges the path should cross in order to achieve minimal length. In fact, since it was shown by [49] that the 3D shortest-path problem in a multi-polyhedral environment is NP-hard, the only practical approach is to use an approximative method, such as [50] or [14] where no edge is longer than a specified maximum length.

A special variant is the problem of computing a shortest path between two points on the surface of a single polytope. For this problem approximation algorithms exist [51–53], and an exact method was presented by [54] that exploits the property that a shortest path on a polyhedron unfolds into a straight line. Another approach was described by [55] with an implementation made public available by [56]. This non-Dijkstra based algorithm finds the shortest path from one source point to all vertices on a polyhedral surface. Still, another approach is the fast marching method, which is an optimal time numerical method for solving the Eikonal equation on a Cartesian grid [57, 58]. The central idea is to advance a wavefront starting at an initial grid point though neighboring grid points continuously until it first reaches the final grid point. It is related to Dijkstra's algorithm [37] used in finding shortest paths between nodes linked in a network. However, the fast marching method is effective for surfaces because it is not limited to path planning along links. In [59] the method is extended from Cartesian grids to triangulated manifolds.

1.3.6 Vertical Cell Decompositions

Vertical cell decomposition (or trapezoidal decomposition) reduces the path planning problem to a graph search problem. It operates on CS obstacles that are polytopes by partitioning the free CS into a set of regions called cells. The term k-cell refers refer to a k-dimensional cell.

An example of how a roadmap and path is found using decomposition in 2D is shown in Figure 1.8. The first step is the decomposition, where a ray is shot upwards and downwards at every vertex of every CS obstacle, until a CS obstacle or bound is hit. These

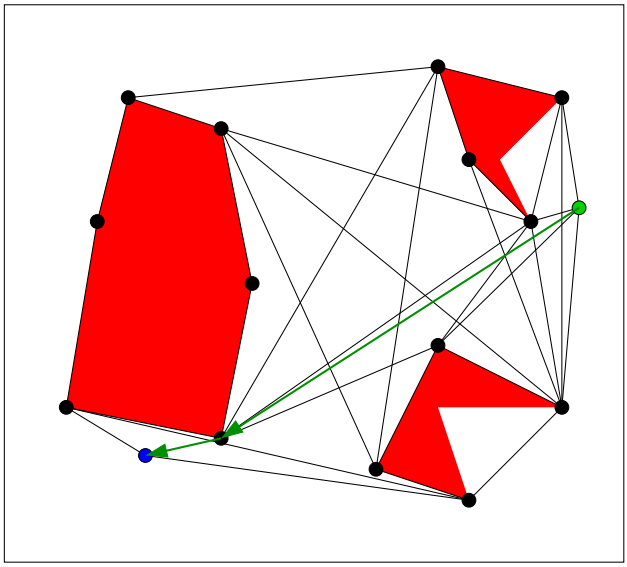


Figure 1.7: The (reduced) visibility graph method uses supporting and separating line segments to construct a 2D roadmap, which is searched to find the shortest path. A roadmap graph is build that avoids the red CS obstacles, although it touches their surface. It is build by connecting graph nodes at vertices that are mutually in line-of-sight and located on a supporting or separating line. The green arrows show the shortest path between the green start and the blue goal. This is obtained by searching the roadmap graph.

subdivide the CS into a set of 2-cells and 1-cells. Each 2-cell is either a trapezoid that has vertical sides or a triangle with one vertical side. Each 1-cell is a vertical segment that serves as the border between two 2-cells. The roadmap graph is constructed from each 2-cell and 1-cell by sampling a point inside each cell, such as the cell centroid, and used as a graph node. The graph edges are obtained by connecting the sample point of each 2-cell with the sample point of each neighboring 1-cell. The green start and blue goal points can be connected to the graph by adding an edge between the points and the sample points of their cells, or the start and goal points can be used directly as sample points as shown in the Figure.

Vertical decomposition can be done in 3D by sweeping the yz plane along the x-axis, which produces convex 3-cells, 2-cells, and 1-cells. A 3-cell is a polyhedron with up to six facets. The cross section of a 3-cell for a fixed x value is a 2-cell, so the 2D vertical decomposition at cross section plane indicates the 3-cells. The idea is to initially find these 3-cells at one extrema of x, and then keep track of how the 3-cells change as the plane is swept along the x-axis. The algorithm first sorts all x values of all vertices of all CS obstacles, since the 3-cells can only change here. The center figure in Figure 1.9 shows a case where a vertex of a CS obstacle is met. The obstacle lies to right of the sweep plane. The algorithm proceeds by first building the 2D vertical decomposition at the first x-value. At each subsequent x-values the 2D vertical decomposition are updated according to the changes.

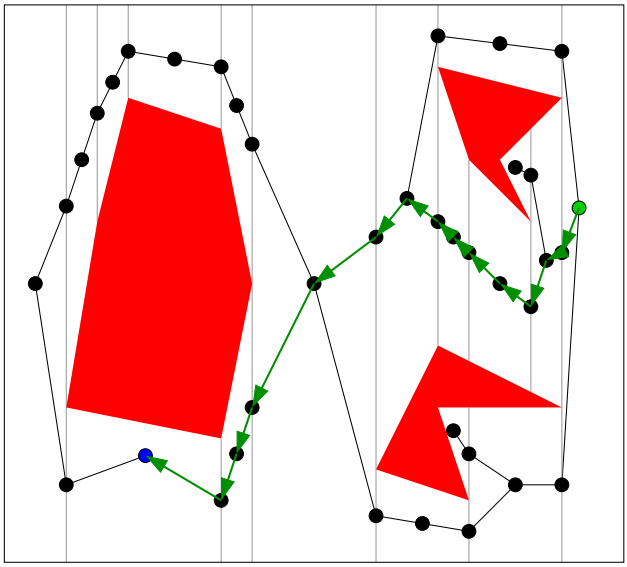


Figure 1.8: The vertical cell decomposition method uses the cells to construct a roadmap, which is searched to yield a path. The CS is decomposed of the gray vertical segments through the vertices of the CS obstacles. A roadmap graph is build that avoids the red CS obstacles by connecting the black nodes of neighboring cells. The green arrows show a path between the green start and the blue goal, found by searching the roadmap.

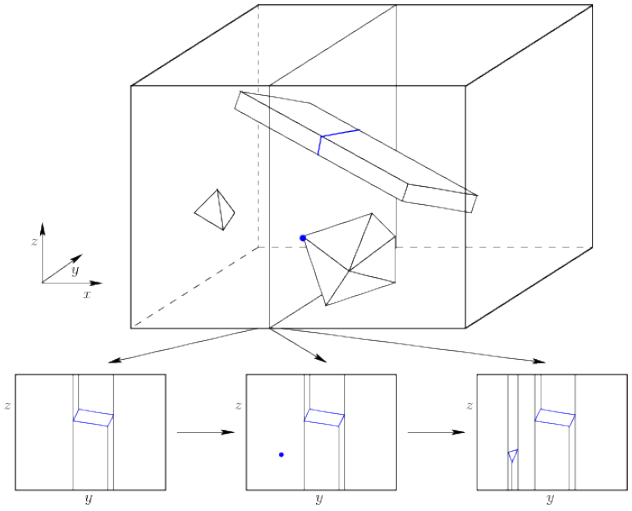


Figure 1.9: The sweeping idea can be applied in 3D. A cross section through the blue vertex of a CS obstacle. The smaller rectangles show a slice of the CS before, at, and after the cross section. From ”Principles of Robot Motion: Theory, Algorithms, and Implementations” [25].

In 3D, the roadmap graph is constructed from each 3-cell and 2-cell. Similar to the 2D case, a sample point such as the cell centroid is selected inside each cell and used as a graph node. The graph edges are now obtained by connecting the sample point of each 3-cell with the sample point of each neighboring 2-cell. Start and goal points can be connected to the graph as before.

1.3.7 State of the Art

Because of the difficulty in representing the CS obstacles for more than two dimensions, the focus in the field of path planning has shifted to methods that completely avoid an explicit representation. This has led to a range of methods that are based on sampling, such as the PRMs from Section 1.3.2 and RRTs from Section 1.3.3. These methods tend to have trouble finding paths through narrow passages, but they have successively solved challenging planning problems with in spaces with many dimensions. Sampling based methods, especially RRTs, are now considered state-of-the-art in path planning.

Many recent improvements to RRTs focus on including dynamical constraints and making the metric used to calculate distance dependent on dynamics. Some [60–62] use time or energy as metric instead of Euclidean distance. Unfortunately, computing such costs is impractical for most higher-order systems since it is equivalent to optimally solving the original motion planning problem [26]. Instead there is a focus on using sub-optimal heuristic functions that are tuned to a particular problem in combination with linearized models to approximate dynamics [63]. The Reachability-Guided RRT [64] improves rate of exploration by emphasizing reachable nodes that exhibit the greatest contribution towards exploring the state space.

RRT algorithms have been developed that adaptively learns a suitable metric during planning. This can be according to how well the nodes and their children are consist with constraints from Cheng [65], or the history of failed expansion attempts on nodes [63]. The dynamic-domain RRT [66, 67] learns about obstacles in the environment from intersected samples and reduces the likelihood of sampling in areas near obstacles. The RRT-Blossom algorithm from Kalisiak [68] expands a node from performing the set of all possible actions, then prunes all expanded node that are closer to other nodes than the parent.

This thesis focuses on path planning for a helicopter with fast dynamics and no significant holonomic constraints. This means that it is reasonable to prioritize a faster path planning and replanning method to a model-based approach. A method is desired where the paths are near-optimal in terms of length, where the amount of spacing between obstacles is insignificant to the difficulty of planning, and where it can be explicitly specified how near to obstacles an aircraft will fly. Consequently, the approach of the original RRT and the recent developments seem unfit for path planning for the Bergen helicopter.

The shortest path is a series of connected curves between two points. Each curve is either on the surface of the CS obstacle or it is a straight line segment that extends between a point and an obstacle, between two different obstacles, or between different parts of the same obstacle. Since all curve end-points are located on the CS obstacle surface, the PRM approach of sampling in the entire CS is inefficient. It is a more efficient strategy to use the Visibility Graphs described in Section 1.3.5 that limit the solution space to these surfaces. This means that the path planning problem does not become harder when the CS is expanded unless the obstacle surface area is also increased. However it necessitates an

explicit representation of the CS obstacles, and a proper extension to three-dimensional space.

This thesis presents an approximative method and a geodesic method for path planning. The first method can be seen as an attempt to bridge visibility graphs and sampling based methods by designing a method that builds a visibility graph based on vertices that are carefully sampled on the CS surfaces to improve possibility of finding a nearly shortest path with relative few vertices. The geodesic method combines sampling with the 2D wavefront concept, to sample paths that are likely to be locally shortest. It then uses these paths to find the global optimum.

2 | Contributions

Section 3 describes the developed planning application that implements all developed methods on path planning and trajectory generation. Section 4 to Section 6 summarizes the contributions on path planning and trajectory generation in the papers. Each section briefly describes the method, results, and recommendations on future work for each paper. Section 7 concludes the thesis and discusses future works.

Approximate Path Planning

Paper A: Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard, "Configuration Space and Visibility Graph Generation from Geometric Workspaces for UAVs," *in Proc. of AIAA Guidance, Navigation, and Control Conference*, 2011.

Paper B: Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard, "Generating Approximative Minimum Length Paths in 3D for UAVs," *submitted to the IEEE Intelligent Vehicles Symposium*, 2012. *Originally accepted by CATEC Research, Development and Education on Unmanned Aerial Systems*, 2011.

Paper C: Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard, "Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning," *in Journal of Autonomous Robots*, 2012.

Geodesics Path Planning

Paper D: Flemming Schøler and Anders la Cour-Harbo, "3D path planning with geodesics on geometric work spaces," *submitted to Journal on Robotics and Autonomous Systems*, 2011.

Trajectory Generation

Paper E: Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard, "Collision free path generation in 3D with turning and pitch radius constraints for aerial vehicles," *in Proc. of AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2009.

Paper F: Flemming Schøler, Anders la Cour-Harbo, and Morten Bisgaard, "State-Control Trajectory Generation for Helicopter in Obstacle-Filled Environment using Optimal Control" *submitted to International Conference on Unmanned Aircraft Systems*, 2012.

3 | Planning Application

The path planning and trajectory generation methods are implemented in a joint planner application that allows setting up obstacle courses using the 3D interface seen in Figure 3.1. The course can be constrained by obstacles and limits on extreme positions, and

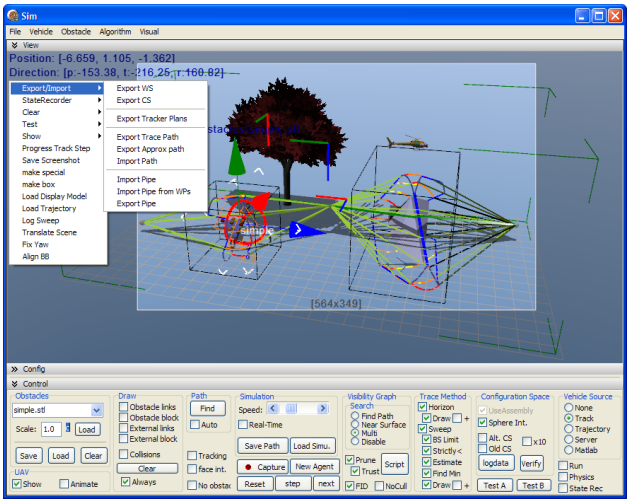


Figure 3.1: The planner GUI gives visual feedback from path planning and trajectory generation methods and flight.

for some methods also constrained by surfaces and other vehicles. Obstacles can be imported from point clouds or geometric meshes and given any position and orientation. The optimal trajectory generation method can use dynamic obstacles, vehicles, and surfaces. Trajectories for dynamic obstacles are generated with physics engine. Vehicle trajectories are generated with the AAU Helisim model. Surfaces can be described by time varying surface functions. They can also be represented by a digital elevation model (DEM). The planner application interface the external modules that are shown in Figure 3.2 and outlined below

- GLScene [69] is an open source set of founding classes for a generic 3D engine. It is used for visual feedback, video and pictures.

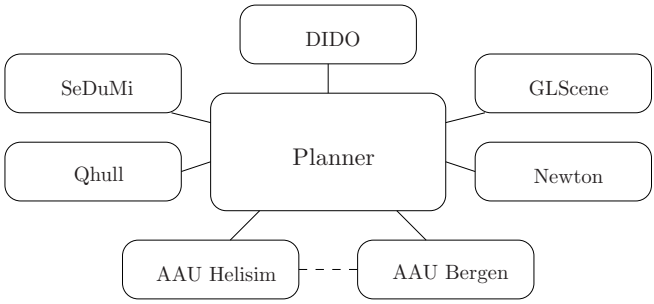


Figure 3.2: Modules interfaced by the planner application

- DIDO [70] is a proprietary software package to solve optimal control problems. It is used for optimal trajectory generation.
- SeDuMi [71] is an open source software package to solve optimization problems over symmetric cones. It is used for building bounding ellipsoids for WS obstacle with optimal trajectory generation method.
- Newton [72] is an open source dynamics engine for real-time simulation of physics environments. It is used in the optimal trajectory generation method for generating trajectories for dynamic obstacle.
- Qhull [73] is open source software for computing convex hulls. It is used for building WS obstacles for the path planners.
- AAU Helisim model [22] is an in-house helicopter model used for simulation and control. A binary version is publicly available.
- AAU helicopter is a modified Bergen Industrial Twin [21]. An interface to the planner has made, although no flight test have been done.

The application has a rich visualization module that provides visualization from all methods and flight. This visual interface improves understanding of the behavior of the path planning problem and of vehicle during flight. The visual interface has the option to make video and high resolution pictures. Such pictures are shown throughout the thesis.

The application is implemented in Object Pascal and runs both as a standalone application and can be invoked as a library from another application such as Matlab. The latter allows synchronization with the AAU Helisim model and controller in Matlab Simulink and means that real-time flight can be shown as it is simulated. Real-time flight can also be shown for the actual AAU Bergen helicopter. Flight can be replayed subsequently in the standalone application. An interface to the DIDO optimal solver allows visual evaluation during the trajectory optimization process.

4 | Approximated Path Planning

4.1 Paper A: Configuration Space and Visibility Graph Generation from Geometric Workspaces for UAVs

In Paper A, an approximated shortest path is found in a 3D environment amongst obstacles. The path is obtained by searching a visibility graph, which is generated in the configuration space. The paper presents a method for constructing the configuration space (CS) from an existing work space (WS), and for generating the visibility graph. The CS obstacles are built from each WS obstacle and are composed of patches of primitive surfaces. These primitives are spheres, cylinders, and planar facets. The CS obstacles are processed to generate a visibility graph for each obstacle. The visibility graph consists of nodes and links. The nodes correspond to vertices that are lifted above the surface of each configuration space obstacle to prevent links from intersecting the patches. The vertices are obtained by sampling the patches in a particular manner to keep the number of vertices at a minimum. The amount of necessary lift depends on a parameter that specifies interpolation density and an upper bound is given in the paper. Links are added between node pairs in the graph if their respective vertex counterparts have inter-visibility. Nodes and links are removed if they represent intersected vertices and edges. The remaining nodes and links are supplemented with addition links that interconnect different obstacles or via points. This structure represents is a visibility graph that can be searched to find a collision free path, as i.e. seen Figure 4.1.

The paper demonstrates that the presented method is able to give a path that is close to optimal in length. It is found that presented method might find a path that takes different route than the optimal path, but that the approximated path should still be nearly optimal. It is however seen that an approximated path tends to continue too far along the surface before moving towards a via point or another obstacle. Additional nodes and links could be added in place of some of the nodes and links that are removed due to intersections. This is because the intersection between two obstacles forms a ridge where a local minimum is likely. For a static configuration space, the visibility graph remains largely constant when planning between different via points. This can be used to perform faster planning on subsequent searches. An efficient implementation of this requires a data structure where interpolated vertices are aware of their relation to other nodes and obstacles.

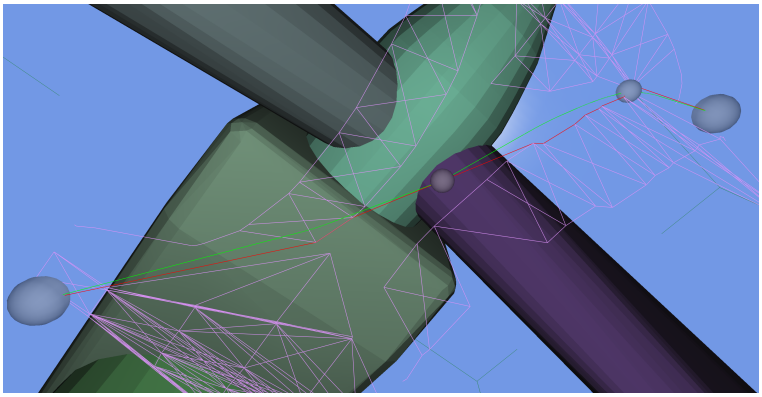


Figure 4.1: The red approximated path and the green geodesic path between two goal locations. A part of the link set is shown in purple for each obstacle.

4.2 Paper B: Generating Approximative Minimum Length Paths in 3D for UAVs

Paper B continues the method in Paper A with more focus on performing fast planning in the case of multiple via points and a single obstacle. The paper compares path length of the optimal and approximated path for new examples. The data structure presented in Paper A was implemented in the planner application to allow better pruning and faster planning. An example of the generated VG and resulting path is shown in Figure 4.2. The VG is composed of a single obstacle visibility graph and a supporting visibility graph. The obstacle visibility graph links node-pairs near the obstacle surface. The supporting visibility graph connects the via points to the CS obstacle by supporting lines. The advantage of separating the obstacle and via point graphs is that while both graphs must be generated when creating the initial full VG, only the supporting visibility graph is created when either the obstacle is moved or rotated or either via point is moved. The full VG is generated such that it contains only links that are relevant when computing the Euclidean shortest path between via points.

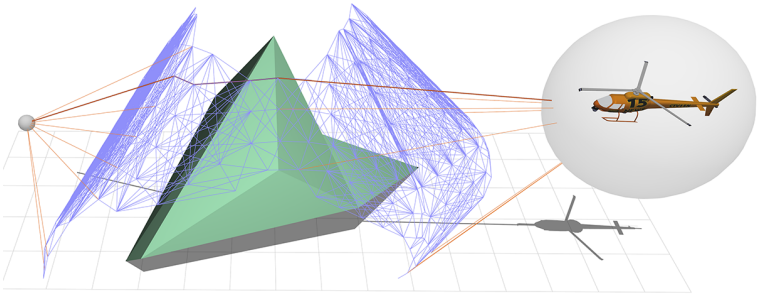


Figure 4.2: The red approximated path between two gray spheres. The path is found around a single obstacle. The blue and orange segments represent the visibility graph.

Paper B finds good performance in terms of accuracy versus computational time. Results also show a good consistency between optimal and approximated paths. The optimal and approximated paths can differ in route, but have nearly same length. The distance between interpolated vertices can be reduced to improve the solution, while it can be increased to generate a smaller graph. However, there is a limit on how small the graph can become for a given obstacle model. Since all points on the WS sphere are part of its convex hull, a dome will add many smaller patches to the CS, which results in a higher density of nodes. A polygon reduction algorithm can be applied on the model before generating the workspace to improve performance.

4.3 Paper C: Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning

Paper C extends the method in Paper B to enable fast planning for multiple obstacles and via points in a changing environment. The paper considers how a visibility graph should be maintained in a changing environment. An example of a VG for multiple obstacles is shown in Figure 4.3. This was done by implementing intersection tests for multiple obstacles from Paper A along with the data structure for faster planning used in Paper B. The graph is now composed of the obstacle visibility graphs and two supporting visibility

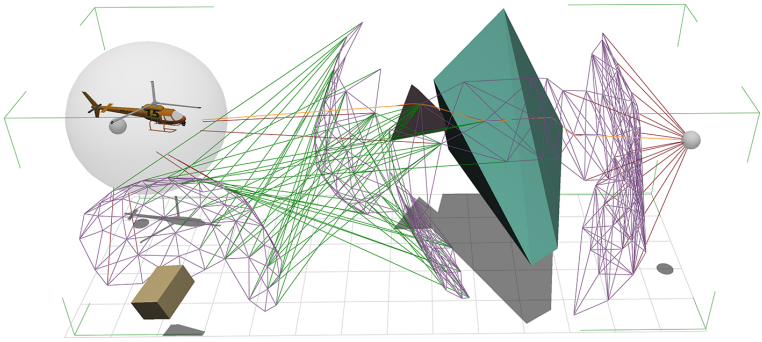


Figure 4.3: The orange approximated path between two gray spheres. The path is found amidst three obstacles. The red, purple, and green segments represent the visibility graph.

graphs. The new additional supporting graph links nodes of different CS obstacles. This is the green graph in Figure 4.3.

Results show that generated paths are near optimal with respect to length and how near intrinsically straight they are. The paths tend to optimal as node spacing is reduced, but this also increases computational requirements rapidly. The method is not suitable for a very high number of obstacles or when obstacles are overly detailed. Replanning and planning with several via points can be done significantly faster by maintaining the VG according to changes in the environment. This means building supporting graphs between pairs of obstacles, and grouping obstacles that form rigid obstacles. A potentially better path can be missed, where nodes at intersection between CS obstacles and bounds are 'missing' due to intersections. Since shortest paths follow these ridges of intersection

between CS obstacles, future works should consider how these could be added to the visibility graph.

5 | Geodesic Path Planning

5.1 Paper D: 3D Path Planning with Geodesics on Geometric Work Spaces

Paper D presents a method for finding the shortest path for a sphere moving around a single polyhedron in a 3D work space. As in Paper A–C, a CS obstacle is build from the Minkowski sum of a WS obstacle and a sphere. The CS obstacles have a continuously differentiable surface that is composed of patches of spheres, cylinders, and planar facets. Shortest paths around such obstacles consist of combinations of tangents and geodesics. These paths will consequently be continuously differentiable. The paper introduces the notion of 'horizons' that are certain curves, which any potential shortest path will intersect. Examples of such horizons can be seen in Figure 5.1. The horizons are used in combination with certain properties of geodesics, which significantly reduces the solution space in which a shortest path must be found. The method reduces the 3D path planning problem to a one dimensional strictly quasiconvex optimization problem. If any continuously differentiable and locally shortest path suffices then numerical optimization methods, such as steepest descend, will converge very quickly. If the shortest path is needed, points in this dimension can be sampled to identify each convex region. The number of regions is closely related to the number of locally shortest paths, and the one containing the shortest path is identified from the sampling. Convex optimization is subsequently used to find the shortest paths in this region.

Results show that the approach is very fast for a single obstacle. It also functions well in combination with approximation based methods to produce piecewise optimal continuously differentiable path. Future works consider better support for multiple obstacles. It is expected that the method scales very well to environments with additional obstacles. It is also expected that a similar could be applied on CS polyhedrons. The reduced complexity of this problem would make it very well suited for multiple obstacles

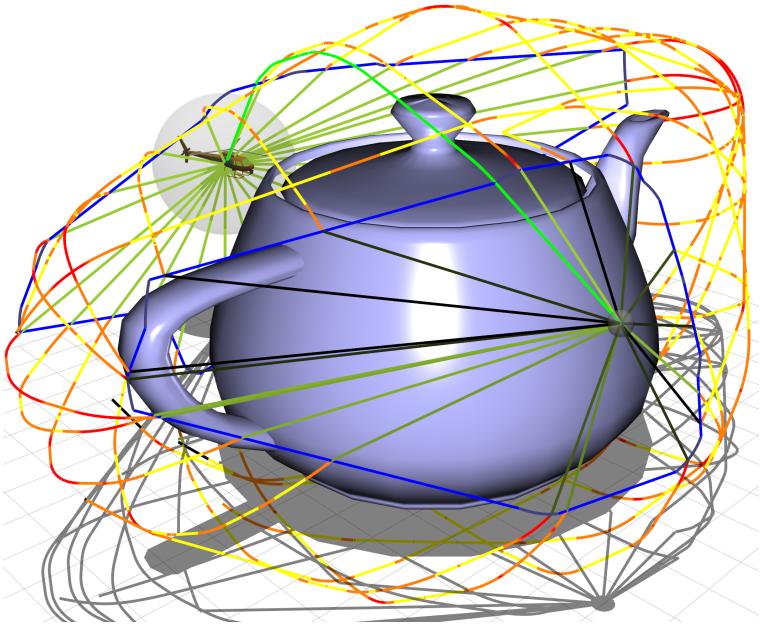


Figure 5.1: Traces between two goal locations around an obstacle. The blue lines are horizons. The green line shows the shortest path.

6 | Trajectory Generation

6.1 Paper E: Collision Free Path Generation in 3D with Turning and Pitch Radius Constraints for Aerial Vehicles

Paths generated from the methods described in Paper A–C consist of a continuous series of line-segments that connect the waypoints. Since these waypoints are generally located on CS obstacle surfaces, following good helmsman behavior or simple path smoothing methods can lead to collisions. Paper E presents a method that makes such paths easier to follow by generating a continuously differentiable collision-free trajectory using Dubins curves. After a waypoint is reached, the aircraft changes direction of flight to target the next waypoint. Although the vehicle moves in all three spatial dimensions, this change can be seen as a rotation in the plane. This plane is defined from the path segment to and from that waypoint. Three arcs are found in each plane that steers the vehicle around a corner of an obstacle, as shown in Figure 6.1.

This gives a continuously differentiable curve around the obstacle through a waypoint. A complete position trajectory is build by repeating these steps for all waypoints. The full state trajectory is generated by determining velocities along the position trajectory. It is used as reference to a full state linear-quadratic feedback controller that does asymptotic trajectory tracking.

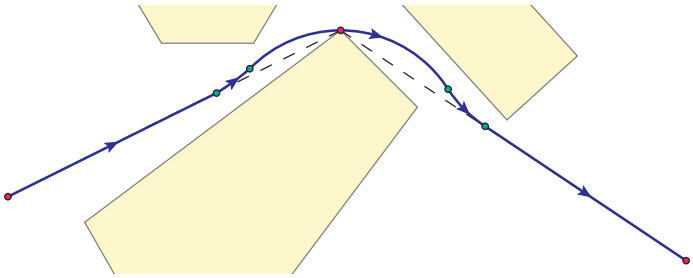


Figure 6.1: The blue trajectory is drawn on top the black dashed path.

Simulated flight shows that the method can be used to track a path very accurately at low speeds. Flight speed is reduced when there is insufficient space for turning. This can happen for waypoints that are located too near each other. Future works could consider multiple waypoints at once. This would ensure better overall placement of arcs and possible faster flight.

6.2 Paper F: State-Control Trajectory Generation for Helicopter in Obstacle-filled Environment using Optimal Control

Paper F presents a different approach to trajectory generation based on optimization. The approach is based on formulating an optimal control problem (OCP). The solution to this OCP is a state and control trajectory that is found using a Legendre pseudospectral method implemented in the software package DIDO. The state trajectory is used as reference to a feedback controller and the control trajectory for feedforward. The paper investigates how different requirements to the state trajectory can be imposed through path constraints using distance functions. Such functions measure distance to obstacles, surfaces, or a path that should be followed. This gives the solver the ability to evaluate how well a candidate trajectory conforms to constraints in the environment. This method can be used to generate a trajectory for an existing path, such as one found using the approaches described in Paper A–D. The altitude of the vehicle is also constrained relative to both ground level and mean-sea level. Other path constraints enable the helicopter to fly inside a pipe, between moving surfaces and obstacles, or amongst trees as seen in Figure 6.2.

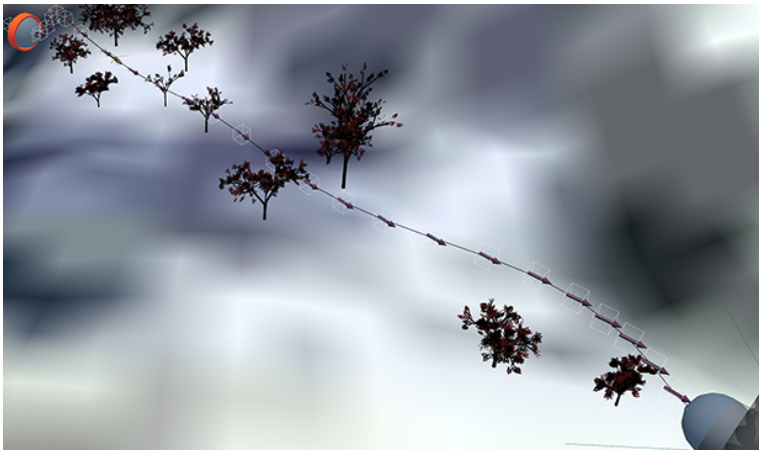


Figure 6.2: The generated trajectory avoids trees and terrain while maintaining low altitude.

The results show a range of different tasks completed satisfactory by introducing the appropriate path constraints in the OCP. It was shown that a state reference can be tracked quite well, although some stability issues could be seen with the tracking controller. This approach gives trajectories that can be tracked more accurately than in Paper E at higher speeds, but the trajectories cannot be generated in real time because of the computational time requirement. Both this issue and the stability issues could be subject to future works.

7 | Conclusions and Future Work

7.1 Conclusions

The topic of this thesis has been the development of new methodology to enable more autonomous UAS. Path planning and trajectory generation are two areas important to the development of such autonomous systems. Path planning tries to obtain a sequence of vehicle configurations (a path) between an initial configuration and a final configuration (goal) that avoids collision. Trajectory generation aims at obtaining a temporal history for the evolution of the configuration.

The focus in this thesis is mostly on the difficult challenge of three dimensional path planning. In this work, the complexity of this problem is reduced using considerations on shape and dynamics of the UAS, as well as properties of shortest paths.

In this thesis an algorithm was developed and implemented to build CS obstacles. Based on this, two fast and supplementary methods were developed. The first is global and use visibility graphs to find a near-shortest path. The second is based on geodesics and implemented as a local method. To the best of the authors' knowledge this is a new approach to path planning.

Two methods for trajectory generation were also considered. One method used Dubins arcs to find a collision free trajectory where the radius of these arcs determined velocity. While this approach was a significant improvement to just waypoint flight using good helmsman behavior, which would not ensure a collision free trajectory, it worked best for low speed flight. The second approach focused on improved speed. To do so, an optimal control problem was formulated. This had minimum time as the primary optimization parameter. Although this approach made it difficult to guarantee a collision free flight, the trajectory would follow the collision free path quite well in practice.

Current UAS do not consider obstacles. They fly at altitudes or in areas where no obstacles exist. The designed methodology could be part of the way to more autonomous UAS. The next logical step would be to recreate the simulated results on the real system.

7.2 Future Work

During this thesis a number of issues that requires further investigation and consideration has been brought to attention and the most important are summarized here as a suggestion for future work. These suggestions range from specific improvements to the developed methods to general suggestions for research that could be investigations.

- The approximated path planner operates by combining visibility graphs that cover different parts of the environment. When CS obstacles intersect each other or the environment bounds, affected graphs are pruned and no graph is put in place although it may contain a local minima.
- It remains an open problem to extend the geodesic path planning method to multiple obstacles. It seems very likely that this should be possible and the main challenge would be to identify horizons seen from a helix or arc path on another obstacle. Such a horizon would be one dimensional and consists of two points at most.
- The geodesic path planning concept is applicable to polyhedral robots. In this case a CS obstacle would be polyhedral and consist solely of planar patches. This makes the challenge of extending the method to several obstacles less intricate, since the one dimensional two point horizons would be seen from points on the edges of the CS obstacles.

References

[1] L. Lin, M. Roscheck, M. A. Goodrich, and B. S. Morse. *Supporting Wilderness Search and Rescue with Integrated Intelligence: Autonomy and Information at the Right Time and the Right Place*. Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10), 2010.

[2] P. Rudol and P. Doherty. *Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery*. IEEE Aerospace Conference, 2008.

[3] M. E. Dempsey. *U.S. Army Roadmap for Unmanned Aircraft Systems 2010-2035*. U.S. Army, 2010.

[4] D. H. Shim, H. Chung, H. J. Kim, and S. Sastry. *Autonomous Exploration in Unknown Urban Environments for Unmanned Aerial Vehicles*. Proceedings of the AIAA Guidance, Navigation and Control Conference, San Francisco, 2005.

[5] T. Samad and G. Balas. *Software-Enabled Control: Information Technology for Dynamical Systems*. Wiley-IEEE Press, 2003.

[6] P. R. Chanler and M. Pachter. *Hierarchical Control for Autonomous Teams*. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, Montreal, 2001.

[7] A. Richards, J. Bellingham, M. Tillerson, and J. How. *Coordination and Control of Multiple UAVs*. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, Monterey, CA, Aug, 2002.

[8] W. Findeisen, F. N. Bailey, M. Bryds, K. Malinowski, and A. Wozniak. *Control and coordination in hierarchical systems, volume 9*. A Wiley Interscience Publication, London, 1980.

[9] Luis B. Gutirrez, George Vachtsevanos, and Bonnie Heck. *A Hierarchical/Intelligent Control Architecture for Unmanned Aerial Vehicles*. In Proceedings of the 11th Mediterranean Conf. on Control and Automation, 2003.

[10] D. Jung. *Hierarchical Path Planning and Control of a Small Fixed-wing UAV: Theory and Experimental Validation*. Georgia Institute of Technology, 2007.

[11] B. Bethke, M. Valenti, and J. P. How. *UAV Task Assignment*. IEEE Robotics and Automation Magazine, 2008.

[12] M. Alighanbari. *Task Assignment Algorithms for Teams of UAVs in Dynamic Environments*. Masters thesis, MIT, 2004.

[13] L. F. Bertuccelli, H.-L. Choi, P. Cho, and J. P. How. *Real-time Multi-UAV Task Assignment in Dynamic and Uncertain Environments*. AIAA Guidance Navigation and Control Conference, Chicago, 2009.

[14] T. Lozano-Perez and M. A. Wesley. *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles*. Communications of the ACM, 22, 1979.

[15] M. Jun. *Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments*. In: S. Butenko, R. Murphey and P. Pardalos, Eds., Cooperative Control: Models, Applicarions and Algorithms, Kluwer, pp. 95-111, 2003.

[16] P. O. Pettersson and P. Doherty. *Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Aerial Vehicle*. The Workshop on Connecting Planning and Theory with Practice, ICAPS, 2004.

[17] E.-M. Kan, M.-H. Lim, S.-P. Yeo, J.-S. Ho, and Z. Shao. *Contour Based Path Planning with B-Spline Trajectory Generation for Unmanned Aerial Vehicles (UAVs) over Hostile Terrain*. Journal of Intelligent Learning Systems and Applications, 2011.

[18] S. Sawant, A. Davari, and J. Wang. *Trajectory tracking of UAV using robust inventory control techniques*. Proceedings of the Thirty-Seventh Southeastern Symposium on Systems Theory, 2005.

[19] H. Wang and J. Gao. *Control system design for UAV trajectory tracking*. Sixth International Symposium on Instrumentation and Control Technology: Sensors, Automatic Measurement, Control, and Computer Simulation, 2006.

[20] D. H. Shim, H. J. Kim, and S. Sastry. *Hierarchical Control System Synthesis for Rotorcraft-based Unmanned Aerial Vehicles*. Proceedings of the AIAA Guidance, Navigation and Control Conference, Denver, 2000.

[21] M. Bisgaard. *Modeling, Estimation, and Control of Helicopter Slung Load System*. Aalborg University, 2007. PhD Thesis.

[22] M. Bisgaard. AAU Helisim Model, June 2011.
http://uavlab.org.

[23] J. T. Schwartz and M. Sharir. *On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds*. Advances in Applied Mathematics, 4:298351, 1983.

[24] J.-C. Latombe. *Robot motion planning*. Springer, 1991.

[25] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[26] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[27] O. Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. IEEE International Conference on Robotics and Automation, 5(1), 90-8, 1985.

[28] D.E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, 11(4):412–442, 1990.

[29] E. Rimon and D.E. Koditschek. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. *Transactions of the American Mathematical Society*, 327(1):71–116, 1991.

[30] J. Barraquand, B. Langlois, and J.-C. Latombe. *Numerical potential field techniques for robot path planning*. IEEE Transactions on Man and Cybernetics, 22(2):224241, 1992.

[31] R. Jarvis. *Collision free trajectory planning using distance transforms*. Mech Eng Trans of the IE Aust, ME10:197191, 1985.

[32] J. Barraquand and J.-C. Latombe. *A Monte-Carlo algorithm for path planning with many degrees of freedom*. In Proceedings IEEE International Conference on Robotics and Automation, pages 1712-1717, 1990.

[33] J. Barraquand and J.-C. Latombe. *Robot motion planning: A distributed representation approach*. International Journal of Robotics Research 10 (6), 628649, 1991.

[34] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. *A Random Sampling Scheme for Path Planning*. The International Journal of Robotics Research 16, 759-774, 1997.

[35] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. IEEE Transactions on Robotics & Automation 12 (4) 566580, 1996.

[36] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[37] E. W. Dijkstra. *A note on two problems in connection with graphs*. Numerische Mathematik 1: 269-271, 1959.

[38] P. E. Hart, N. J. Nilsson, and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4: 100-107, 1968.

[39] R. Geraerts and M. H. Overmars. *A comparative study of probabilistic roadmap planners*. Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02), pp. 4357, 2002.

[40] J. H. Reif. *Complexity of the movers problem and generalizations*. Proceedings IEEE Symposium on Foundations of Computer Science, pp. 421-427, 1979.

[41] S. LaValle and J. Kuffner. *Rapidly-exploring random trees: Progress and prospects*. In Proceedings of the Workshop on the Algorithmic Foundations of Robotics, 2000.

[42] S.M Lavalle. *Rapidly-exploring random trees: A new tool for path planning*. Computer Science Dept, Iowa State University, Tech. Rep. TR: 9811, 1998.

[43] S. LaValle and J. Kuffner Jr. *Randomized kinodynamic planning*. The International Journal of Robotics Research, 2001.

[44] H. Choset and J. Burdick. *Sensor-Based Exploration: Incremental Construction of the Hierarchical Generalized Voronoi Graph*. International Journal of Robotics Research, 19(2):126148, 2000.

[45] H. Choset and J. Burdick. *Sensor based motion planning: The hierarchical generalized Voronoi graph*. International Journal of Robotics Research, 19(2):96125, 2000.

[46] F. Aurenhammer. *Voronoi diagrams A survey of a fundamental geometric structure*. ACM Computing Surveys, 23:345405, 1991.

[47] M. Sharir. *Motion Planning*. Handbook of Discrete and Computational Geometry, CRC Press, 733–754, 1997.

[48] N. J. Nilsson. *A mobile automaton: An application of artificial intelligence techniques*. In 1st International Conference on Artificial Intelligence, pages 509-520, 1969.

[49] J. Canny and J. H. Reif. *New lower bound techniques for robot motion planning problems*. In Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci., 1987.

[50] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inf. Process. Lett.*, pages 259–263, 1985.

[51] S. Har-Peled. *Constructing approximate shortest path maps in three dimensions*. SIAM J. Comput., 2005.

[52] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan. *Approximate shortest paths on a convex polytope in three dimensions*. J. ACM, 1997.

[53] K. R. Varadarajan and P. Agarwal. *Approximating shortest paths on an nonconvex polyhedron*. In Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci., Miami Beach, Florida, 1997.

[54] M. Sharir and A. Schorr. *On shortest paths in polyhedral spaces*. SIAM J. Comput., 1986.

[55] J. Chen and Y. Han. *Shortest paths on a polyhedron*. SCG '90 Proceedings of the sixth annual symposium on Computational geometry, 1990.

[56] B. Kaneva and J. O'Rourke. *An Implementation of Chen & Han's Shortest Paths Algorithm*. Proc. of the 12th Canadian Conference on Computational Geometry, New Brunswick, 2000.

[57] S. Cao and S. A. Greenhalgh. *Finite-difference solution of the eikonal equation using an efficient, first-arrival wavefront tracking scheme*. Geophysics, 59, no. 4, 632-643, 1994.

[58] J.A. Sethian. *Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, Monograph on Applied and Computational Mathematics, 1999.

[59] R. Kimmel and J. A. Sethian. *Computing geodesic paths on manifolds*. Proc. Nat. Acad. Sci. 95, 1998.

[60] E. Frazzoli, M. A. Dahleh, and E. Feron. *Real-Time Motion Planning For Agile Autonomous Vehicles*. Journal of Guidance, Control, and Dynamics, 2002.

[61] R. Tedrake. *LQR-Trees: Feedback motion planning on sparse randomized trees*. In Proceedings of Robotics: Science and Systems (RSS), page 8, 2009.

[62] E. Glassman and R. Tedrake. *LQR-based heuristics for rapidly exploring state space*. IEEE International Conference on Robotics and Automation, 2010.

[63] Frederick D. Kim, Roberto Celi, and Mark B. Tischler. *Forward Flight Trim and Frequency Response Validation of a Helicopter Simulation Model*. Journal of Aircraft, Vol 30, No 6, pp 854-863, 1993.

[64] A. C. Shkolnik. *Sample-Based Motion Planning in High-Dimensional and Differentially-Constrained Systems*. Massachusetts Institute of Technology, 2010. PhD Thesis.

[65] P. Cheng. *Sampling-based motion planning with differential constraints*. University of Illinois at Urbana-Champaign, 2005. PhD Thesis.

[66] A. Yershova, L. Jaillet, T. Simeon, and S.M. LaValle. *Dynamic-domain rrts: Efficient exploration by controlling the sampling domain*. International Conference on Robotics and Automation (ICRA), pages 38563861, 2005.

[67] A. Yershova. *Sampling and Searching Methods for Practical Motion Planning Algorithms*. University of Illinois at Urbana-Champaign, 2008. PhD Thesis.

[68] K. Kalisiak. *Toward More Efficient Motion Panning with Differential Constraints*. University of Toronto, 2008. PhD Thesis.

[69] E. Grange and M. Lischke. GLScene, January 2012. http://glscene.org.

[70] I. M. Ross. *User's Manual for DIDO: A MATLAB Application Package for Solving Optimal Control Problems, Tomlab Optimization*, Feb 2004.

[71] J.F. Sturm. *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*. Optimization Methods and Software, 11-12:625-653, 1999. http://sedumi.ie.lehigh.edu.

[72] J. Jerez and A. Suero. Newton physics engine, January 2012. http://newtondynamics.com.

[73] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. *The quickhull algorithm for convex hulls*. ACM Trans. Mathematical Software 22, 1996. http://qhull.org.

Paper A

Configuration Space and Visibility Graph Generation from Geometric Workspaces for UAVs

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

This paper was published in:
Proceedings of AIAA Guidance, Navigation, and Control Conference, 2011

Revised to correct typographical errors

Configuration Space and Visibility Graph Generation from Geometric Workspaces for UAVs

Flemming Schøler* and Anders la Cour-Harbo† and Morten Bisgaard‡
Aalborg University, 9220 Aalborg East, Denmark

We address the challenge of generating a visibility graph for 3D path planning for a small-scale helicopter through an environment with geometric obstacles. The visibility graph is based on approximating a number of continuously differentiable configuration space obstacles that are generated from the convex hull of their work space obstacles. An approximation to the optimal path can subsequently be found using an existing graph search algorithm. The presented approach is suitable for fully known environments with a large number of truly 3D (not merely "raised" 2D) obstacles. A test scenario for a small-scale helicopter inspecting a wind turbine is shown.

I. Introduction

UAS have become a preferred, indispensable, and increasingly used platform for many applications where manned operation is considered unnecessary, repetitive, or too dangerous.[1,2] Path planning and trajectory generation are fundamental area for UAS development[3] and is used to find a path through an environment under a set of constraints. Specifically, for tasks such as surveillance, inspection, search and rescue, aerial mapping, cinematography, etc., small-scale autonomous helicopters are increasingly being used. In many such tasks it is advantageous to operate in close proximity to the obstacles or to follow their surface. Methods for describing these obstacles as geometric figures from real-world measurements have also been improved recently[4,5] with 3D environment mapping data getting increasingly detailed and readily available including entire city maps. These recent advances now allow the use of 3D maps for high-level planning, including tasks such as inspecting a stopped wind turbine.

A. Background

We consider methods for path planning that can be used to operating a small-scale helicopter near the surface of obstacles. We distinguish the term "trajectory" from "path" in that a path is without time and trajectory includes time. In this paper we are mainly concerned with the geometric problem of path planning. We consider the general trajectory planning problem for a UAS as having two levels; 1) a path that avoids larger static obstacles, such as buildings, mountains, etc., and 2) a trajectory along the path that also avoids smaller, possibly mobile object and at the same time takes the dynamics of the vehicle into account by generating a feasible state trajectory and possibly a related control trajectory. The present work is a method for generating and optimal solution to the first level. And because that level is static the method presented also prioritize the ability to easily do multi-query searches and replanning. We look at feasibility of the path from a qualitative perspective by preferring path with higher order of differentiable. This means that we do not consider the structural and dynamical capabilities of the vehicle in greater details at this level. This and dynamic obstacles should be handled at a lower trajectory planning level, which generates full state and control trajectories, and typically also does single query searches because of the dynamic obstacles. Although we only look at top level path planning, we still employ a stepwise strategy that more efficiently can handle isolated changes in the environment as well as re-planning where only changes to via points change. This work is focused on path planning using visibility graphs that is based on the configuration

*Ph.D. Student, Institute of Electronic Systems, Fredrik Bajers Vej 7C.
†Associate Professor, Institute of Electronic Systems, Fredrik Bajers Vej 7C.
‡Assistant Professor, Institute of Electronic Systems, Fredrik Bajers Vej 7C.

space. The generalized visibility graph $VG(N, L)$ contains a node set N and a link set L of links between node pairs. The nodes are sampled from the edges of the configuration space, and will hence only approximate an optimal solution. The link set is directed and produced according to a custom optimality criterion. This means that other costs than Euclidean distance can be used and links can have two different costs according to the direction of travel. To obtain a path, the visibility graph should be searched using a graph search algorithm, for instance Dijkstra's algorithm[6] or A*.[7] However this part is not the focus of this research, and we will not go into details. Contrary to other popular methods such as the Rapidly-exploring random tree method, searching the tree will also reveal if no solution exists. Path planning using visibility graphs has been subject to much research. In comparison, the literature on how to practically, automatically and efficiently generate configuration spaces from a known general 3D work space polyhedron seems limited to building primitive solids or geometric representations and often an existing configuration space is assumed. The main contribution of this paper lay in an algorithm that build geometric configuration spaces and a visibility graph from a group of known workspace obstacles. Our algorithm works on a collection of meshes or point-cloud representations of the obstacles, and exploits knowledge of the obstacles to efficiently determining links for the graph. In order to meet that goal some generalizations must be made. The configuration space for a helicopter has usually six degrees-of-freedom, with three dimensions to describe position and three to describe attitude. However, by considering a bounding sphere for the helicopter that is centered in center-of-mass, the attitude will be irrelevant when determining a collision free path and the configuration space can be reduced to three degrees-of-freedom. Besides reducing the complexity of the problem (still a NP-complete), the configuration space obstacles will also get a continuously differentiable surface and any geodesic path will be continuously differentiable which will be easier to track. Generally the higher order of smoothness our path has the easier it should be to track. Another advantage is that one issue criticized about the visibility graph method can be easily avoided. This issue is that it is impractical to operate near obstacles since errors will cause collisions, and is avoided by introducing a safety margin that increases the vehicle bounding sphere.

The configuration space for a rotorcraft, or indeed any other airborne vehicle, usually has six degrees-of-freedom, with three dimensions to describe position and three to describe orientation. However, by instead planning a path for the bounding sphere of the helicopter, the attitude will be irrelevant for determining a collision free path and the configuration space can be reduced to three degrees-of-freedom. The configuration space obstacles are defined as the Minkowski sum of convex work space obstacles and the bounding sphere. They have a continuously differentiable surface and any shortest path (geodesic) on their surface will be continuously differentiable, which in turn will be easier to track for a vehicle with bounded dynamics. This is because a path that is continuously differentiable does not require instantaneous changes in velocity (although it does for acceleration, unless it is twice differentiable). Risks of collisions due to tracking errors can be avoided by increasing the vehicle bounding sphere accordingly.

Previous work on how to practically, automatically, and efficiently generate a configuration space from a 3D point-cloud in WS seems limited to building primitive solids or geometric representations, and in many path planning publications an existing configuration space is simply assumed. The main contribution of this paper lies in an algorithm that builds geometric configuration space obstacles suitable for easy generation of a VG from a group of workspace obstacles. Each workspace obstacle is made convex and is build from vertices feed to the algorithm. These vertices do not need to be convex and could be from, for instance, the mesh of geometric obstacle or a point-cloud from sampled data. The workspace obstacles are made convex since non-convex parts are not relevant when searching the visibility graph for a solution. When the path is non-trivial, meaning that there is no line-of-sight in configuration space between initial and desired configuration, the shortest path will at times follow the surface of one or more obstacles in configuration space. The shortest path lies on the convex hull of the surface.

B. Previous work

Visibility graphs for path planning has been used extensively even though finding an optimal solution to the general path planning problem in 3D is NP-complete.[8,9] The difficulty of the problem is that the shortest path around a polyhedral obstacle does not in general traverse only vertices of the polyhedron but also points its edges. The concept of adding additional vertices along these edges in configuration space, so that no edge is longer than a specified maximum length, is introduced by.[8] This method generally results in a good approximation to the optimum path. In Lozano[10] an algorithm is presented that generates a configuration space obstacle in 2D from a convex polygonal obstacle and a convex polygonal vehicle. This is done by sliding

the vehicle polygon around the obstacle in such a way that they are always in contact at one point. This approach would be adapted to our case by approximating the vehicle by for instance a geodesic dome. While the algorithm can be extended from 2D to 3D, this results in a sequence of algebraic equations that describe the configuration space obstacles. Another used approach for polyhedron is based on convex hulls. For a convex polyhedron vehicle and obstacle, a polyhedron describing the Minkowski sum can be calculated. This is done through vector addition of all points of the vehicle and obstacle, and then computing the convex hull of the resulting point set. Convex hulls can be computed using several algorithms. See e.g. O'Rourke[11] for a robust $O(n^2)$ three-dimensional implementation or the quickhull algorithm.[12] We suggest a similar approach, but substitute the convex polygonal of the vehicle with a sphere approximation. This leads to a 3 degrees of freedom configuration space, and a configuration space that tends towards continuously differentiable as the resolution of the approximation improves. A continuously differentiable work space allows finding paths that are easier to track, than in discontinues cases, because it does not require instantaneous changes in velocity (although it does for acceleration).

C. Present work

In this paper the focus is on inspection tasks where helicopter must stay close to obstacle surface with multi-query searches must be done due to i.e. multiple via points or replanning. The focus is also on cases where limited changes occur in the workspace. We consider the configuration space of an, at least conceptually, spherical vehicle moving amongst polyhedral obstacles, i.e. obstacles described by patches of planar surfaces. We generate the configuration space by translating or growing each surface patch by the radius of the sphere. Our approach will generate a convex configuration space surface for each obstacle. This surface goes towards continuously differentiable as resolution improves. We take an sequential approach by 1) construction a visibility graph for each obstacle, 2) then prune nodes and links from intersecting obstacles, 3) combine the visibility graphs, 4) finally include nodes and links from via points. The advantage of this sequential approach is that while all steps must be completed when creating the initial visibility graph, only a partial update of the visibility graph is required when obstacles are added or removed, even less when obstacles are just moved or rotated, and even further less when just via points are changed.

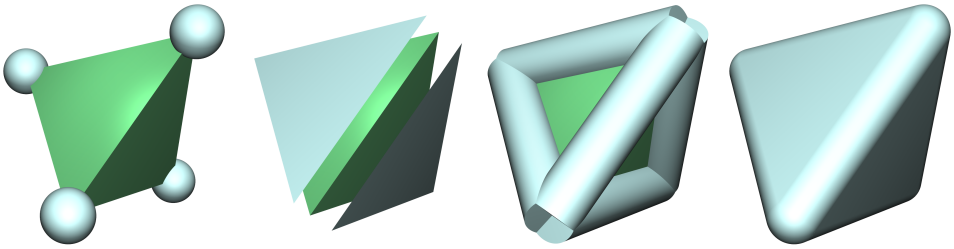
II. METHOD

The general problem of finding a path for a helicopter is a 6 degrees-of-freedom problem. Our approach generates a configuration space based on the Minkowski sum of the vehicle bounding sphere and the convex hull of work space obstacles. This approach makes the configuration space invariant to vehicle orientation, and thus reduces the problem to a 3 degrees-of-freedom problem. These obstacles are represented in configuration space as a set of primitives (planes, cylinders, and spheres) generated by the algorithm from the any work space obstacles.

A. Configuration space

The space of all possible positions, denoted the configuration space, is constrained by the obstacles. For each obstacle in work space, a configuration obstacle is created. The configuration space obstacle is the Minkowski sum of the vehicle bounding sphere and the convex hull of a work space obstacle polyhedron. From a geometric point of view, this Minkowski sum is obtained when vertices in the convex hull of the workspace are replaced with sphere patches, edges with cylinders, and faces are translated along their normal. An example of this is shown in Figure 1.

We decompose the configuration space obstacles by patches of primitive surfaces. These primitives are sphere, cylinder, and planar facets. Any two neighboring patches intersect at one common edge. This edge is either straight or an arc edges. It follows from the Minkowski sum operation that spherical patches only has cylindrical patches as neighbors. Cylindrical patches have two opposing sphere patches and two opposing facet patches as neighbors. Facet patches only have neighboring cylindrical patches. Because of this composition, all nodes for the visibility graph can be determined from the sphere and cylinder patches. These nodes origin from the interior of the sphere and cylinder patches, and from their common edge. Once the node set is formed for a patch, the link set can be easily determined knowing that the obstacle is convex. This means that links between nodes belonging to an obstacle are restricted to nodes of the same patch. The Sections B and C explain how the node set is formed based on the sphere and cylinder patches and how the



(a) Spheres at vertices (b) Facets are translated (c) Cylinders at edges (d) Full configuration space

Figure 1. Configuration space is formed by the Minkowski sum of a polyhedron workspace obstacle and a vehicle bounding sphere.

link set is found on each of the three patch types. In Section 1 the visibility graph is extended to include multiple obstacles.

B. Sphere patch

A sphere patch was added to the configuration space obstacle for each vertex in workspace obstacle. A sphere patch C is a simple closed geometric figure on the surface of a unit hemisphere that is formed by the arc segments on the great circles (the geodesics) between N cyclic ordered vertices v_i, $i = 0, ..., N-1$. The last vertex v_N is assumed to be the same as the first, i.e.: the patch is closed. The sphere patch is convex so the geodesic between any two vertices in C lies entirely in C. Each vertex in the patch is obtained by translating a workspace vertex along the neighboring facet normal in an ordered manner. So the patch polygon has the same number of nodes as number of facets that use the workspace vertex. The subject of ordering the vertices is fairly standard and will not be covered here. Each vertex is shared by two cylinder patches, one sphere patch and one planar patch. Each segment of the patch neighbor a different cylinder patch. The objective is now to determine where to add nodes and links for each patch, such that as few nodes as possible are used in the visibility graph, while at the same time limiting the volume of the configuration space obstacle. Since any two different points on the patch will not have line-of-sight, it is necessary to place them above the surface.

We reduce the spherical polygon that outlines C to a set of connected spherical triangles to easier determine where to add nodes. We let the spherical polygon be composed by N spherical triangles, where each triangle shares a shares an interior central vertex denoted v_c. Because of convexity, this interior vertex is given by the weighted average in Eq. 1 of the vertex set q and some non-negative weights w_i that sum to 1.

$$q = \sum_{i=0}^{N-1} w_i v_i \tag{1}$$

A central vertex candidate can be found simply by equating all weights to $1/N$, but a better choice is usually to find the central vertex that minimize spherical distance to the vertex set. This spherical weighted average is equal to the point q on the unit sphere S which minimize $f(q)$;

$$f(q) = \frac{1}{2} \sum_{i=0}^{N-1} w_i \mathrm{dist}_{\mathrm{geo}}(q, v_i) \tag{2}$$

where $\mathrm{dist}_{\mathrm{geo}}(q, v_i)$ is the geodesic distance from q to v_c on S. This problem is well-defined for vertices on a hemisphere. A fast iterative algorithm with quadratic convergence rate exists, see.[13]

The visibility graph, that is, the node set and link set for the sphere patches, must be located above/outside the patch, so that neighboring nodes will have line-of-sight (i.e. not intersecting the patch). In essence the visibility graph takes shape of a triangle mesh approximating the sphere patch. How far above the nodes

must be located are resolution dependent - the greater resolution, i.e. nodes, the less displacement is required. Here we base the calculation of this displacement r_n on the worst case:

$$r_n = \frac{r_{bs}}{\sqrt{\frac{1}{2}\cos(a_{\max})^2 + \frac{1}{2}}}$$

where

$$a_{\max} = \min(l_{\max}, r_{bs}\pi\sqrt{2})/r_n$$

is the angle between two neighboring nodes, $l_{\max}$ is the maximum allowed arc distance between neighboring nodes, and r_{bs} is the vehicle bounding sphere radius. Thus r_n can be found as the solution to the following minimization problem

$$\underset{r_n \in (r_{bs}, \sqrt{2}r_{bs})}{\operatorname{argmin}} \left| r_n - \frac{r_{bs}}{\sqrt{\frac{1}{2}\cos\big(\min(l_{\max}, r_{bs}\pi\sqrt{1/2})/r_n\big)^2 + \frac{1}{2}}} \right| \tag{3}$$

1. Node Set

In order to construct the node set for the obstacle, we first interpolate the edge endpoints of the spherical polygon to find a set of vertices v_a. We then interpolate the edges formed by these vertices and the central vertex to get a vertex set, which can be scaled by r_n to obtain the nodes. From the set of vertices v that consecutively forms the edges of one spherical polygon. The number of nodes $N_s(i)$ along an edge index i required for a certain resolution, determined by $l_{\max}$, is given by:

$$N_{\text{se}}(i) = \lceil r_n \arccos(v_i \cdot v_{i+1})/l_{\max} \rceil$$

where $l_{\max}$ is the maximum allowed arc distance between neighboring nodes. The vertex set $v_a(i,s)$ on the arc between v_i and v_{i+1}, and where $s = 0, ..., N_s(i) - 1$ is:

$$v_a(i,s) = \frac{\sin\left(\frac{s}{N_s(i)}\arccos(v_i \cdot v_{i+1})\right) v_{i+1} + \sin\left(\left(1 - \frac{s}{N_s(i)}\right)\arccos(v_i \cdot v_{i+1})\right) v_i}{\sqrt{1 - (v_i \cdot v_{i+1})^2}} .$$

Nodes are added by spherical linear interpolation between the central vertex v_c and the vertices in v_a for each edge. The required number of nodes is given by $N_t(i,s)$:

$$N_t(i,s) = \lceil r_n \arccos(v_a(i,s) \cdot v_c)/l_{\max} \rceil$$

where $l_{\max}$ is the maximum allowed distance between neighboring nodes. The node set $n_{sp}(i,s,t)$ is then given for integer values of $t = 0, ..., Nt(i) - 1$ as:

$$n_{sp}(i,s,t) = r_n \frac{\sin\left(\frac{t}{N_t(i)}\arccos(v_a(i,s) \cdot v_c)\right) v_c + \sin\left(\left(1 - \frac{t}{N_t(i)}\right)\arccos(v_a(i,s) \cdot v_c)\right) v_a(i,s)}{\sqrt{1 - (v_a(i,s) \cdot v_c)^2}} .$$

The interpolation functions above have no singularities since consecutive vertices are always spaced less than π for any obstacle that extend in three dimensions. No node is added at the central vertex v_c by the equations above. This node is given by $n_{sp.p} = r_n v_c$.

2. Link Set

The link set is found from the set of edges defining the convex hull of the node set above the sphere patch. The convex hull of a set of points S is the intersection of all convex sets containing S. For N nodes $p_1, ..., p_N$, the convex hull C is given by the expression:

$$C = \left\{ \sum_{i=1}^{N} \lambda_i p_i : \lambda_i \geq 0 \ \forall \ i, \sum_{i=1}^{N} \lambda_i = 1 \right\} .$$

Since a sphere patch is connected to its neighbors, it should be an open polygon, whereas the hull obtained by the mentioned algorithms is closed. One way of handling this is to add a dummy node is on the opposite hemisphere, e.g. $-v_c$ before calculating the hull. This node and connected edges can then be removed after the hull is found, which result in an open polygon that can be patched with its link set from the neighboring cylinder patches.

C. Cylinder patch

From the sphere patch a set of nodes was found at the arc edges of the cylinder patch. These arc edges are located at the top and bottom of the cylinder. A link is now added for each opposing node pair $v_i w_i$. We then connect opposing and neighboring nodes into rectangular facets. Additional nodes are inserted along opposing nodes, so that no nodes are spaced more than a maximum distance $l_{\max}$ in each facet of the path. Because the height of the cylinder is constant, the number of vertices along this height must be

$$N_{\mathrm{c}}(i) = \lceil ||v_i - w_i||/l_{\max} \rceil \ .$$

Each interpolation point for pair i is given by

$$n_{\mathrm{cp}}(i,j) = v_i + (v_i - w_i)j/N_{\mathrm{c}}(i)$$

for $j = 1, \ldots, N_{\mathrm{c}} - 1$ and for $i = 0, \ldots, N_{\mathrm{se}}(u)$, where u is the index of the corresponding sphere patch polygon edge. The link set is generated by connecting nodes of the planar patches after the rule describe in Figure 2, where the number of nodes added along an patch edge i is

$$N_{\mathrm{v}}(i) = \lceil ||v_i - v_{i+1}||/l_{\max} \rceil \ .$$

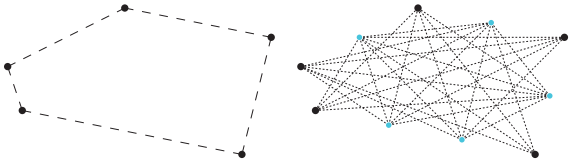


Figure 2. Each facet adds links to the visibility graph. Nodes are added at the vertices of the facet and along its edges, so that two neighboring nodes on an edge are not spaced more than a given threshold. Links are added between all pairs of nodes that are not located on the same edge unless the pair is the endpoints of that edge. This pruning is done since such links will not be part of an optimal path. The left picture shows the visibility graph constructed from the edges and vertices of the facet. The right picture shows where additional nodes and links are added. Edges and vertices along the facets are generally shared by neighboring facets but are only added to the graph once.

At this point all nodes for the obstacle visibility graph have been found, and only the links from of the planar patches remain. These links are again found by connecting nodes after the rule describe in Figure 2.

1. Composing a Visibility Graph

Once visibility graphs are generated for each obstacle, the via points are added, and node pairs from different obstacles are added to the graph if they have line-of-sight. To test line-of-sight, an intersection test is performed that tests against all obstacles for intersection, until either an intersection is found or all obstacles have been evaluated. Methods for collision testing are numerous. One method is to test the candidate link edge for intersection against all facets of an obstacle, which can be done by first determining the point of the segment/plane intersection and then determining if this point is inside the face polygon. Testing whether a point is inside a polygon is a common operation in computer graphics and several methods are compared by.[14] For cases with less than 10 edges per face, algorithms which partition the polygon into triangles are amongst the fastest. Not all triangles necessarily have to be tested. Creating a spatial search tree can significantly reduce the number of tests required and hence the computational load. This works by using a faster test to initially exclude groups of faces that do not intersect the segment.

III. Results

The implemented method for construction visibility graphs will be demonstrated by finding a path around a wind turbine. The path must visit a number of via points near the surface as shown in Figure 3. The wind turbine is a Vestas V52 model. The helicopter model is a Bergen Industrial Twin helicopter that has a bounding sphere radius of 1.7 m. This test case will take the helicopter through the six via points (vp) listed in Table 1.

For each part of the vps the full graph is build and searched. On subsequent searches the VG remain largely constant since no changes to the environment occur. The approximated shortest path p_a is found using a graph search algorithm on the full visibility graph, and is then compared to the optimal path p_g. The optimal path is composed of geodesics and tangents on the CS obstacle and is calculated using a minimization algorithm.

Table 1. Position of the six via points.

Via point	1	2	3	4	5	6
X (m)	-2.0	4.0	-3.5	8.5	-6.0	-3.5
Y (m)	-4.0	1.0	2.0	-2.0	-32.5	-4.0
Z (m)	1.0	35.0	68.0	69.5	53.0	67.0

The helicopter starts off at the base of the turbine (vp 1) and moves towards the nacelle (vp 3). Halfway up the tower a via point at the back of the tower is visited (vp 2). From the nacelle the path moves underneath the nose cone between two rotor blades to a position on the base of one the rotors (vp 4). This rotor is inspected by moving towards a point on its tip (vp 5) and then back to the base, this time one the back of the rotor (vp 6). In the sections below the entire path is split into three parts: Moving along the tower in vp 1-3, moving under the nose in vp 3-4, and moving around a rotor in vp 4-6. Instead of showing the full visibility graph, which is more difficult to illustrate in 3D, we will usually only show the combination of nodes and links that result in the approximated shortest path P_a.

The wind turbine model seen in Figure 3 is composed of six separate obstacles as indicated by the different colors. These are the tower, the gearbox house, and three rotors attached to the nose. In the sections below the entire path is split into three parts: Moving along the tower in vp 1-3, moving under the nose in vp 3-4, and moving around a rotor in vp 4-6.

Vp 1-3 The path starts off from the base of the tower and moves upwards towards a point half way up the tower before continuing to the gearbox house, see Figure 5. From the two plots in Figure 4 a good consistency between the geodesic and approximated path can been seen. The largest discrepancy can be seen in the top view near vp 3. Here, the geodesic path will leave the configuration space surface as soon as it "sees" vp 3, while the approximated path must stay near the surface until a node is met that has a visible link to the vp. As a result, an approximated path tends to continue too far along the surface before moving towards a vp or another obstacle. From the top view it can also be seen that the radius of the tower at the base is larger than at the top, since both paths intersects the y-axis at -4.2 m near the base while intersecting 3.6 m near the top. As a result, the geodesic path projected on the vertical plane is not a straight line, as it would, had the tower had constant radius. The largest difference between the paths occurs between vp 2 and vp 3 the side view. Figure 5(a) shows a close up of some of this path. Although nodes could have been picked that seem nearer to the geodesic path, this does not result in a globally shorter path. The approximated path is 0.05 % longer for this part.

Vp 3-4 At vp 3, the path continues between the two downwards pointing rotor blades to a position in front of the leftmost blade. The two plots in Figure 6 show a larger discrepancy between the paths, and the approximated path is 2.1 % longer. Because of limitations in the minimization method, used to find the optimal path, it can only operate on one obstacle at a time. The approximated path is used to add two intermediate via points on the approximated path between via point 3 and via point 4, such that each part of the geodesic path is only located on a single obstacle. Although this means that the geodesic path is only optimal between the intermediate via points, we find this sufficient for comparison. The location of the two added via points can be seen on Figure 7, as well as the relevant visibility graph link set for each obstacle. The links set interconnecting different obstacles are too many to show here. Links and nodes in the obstacle

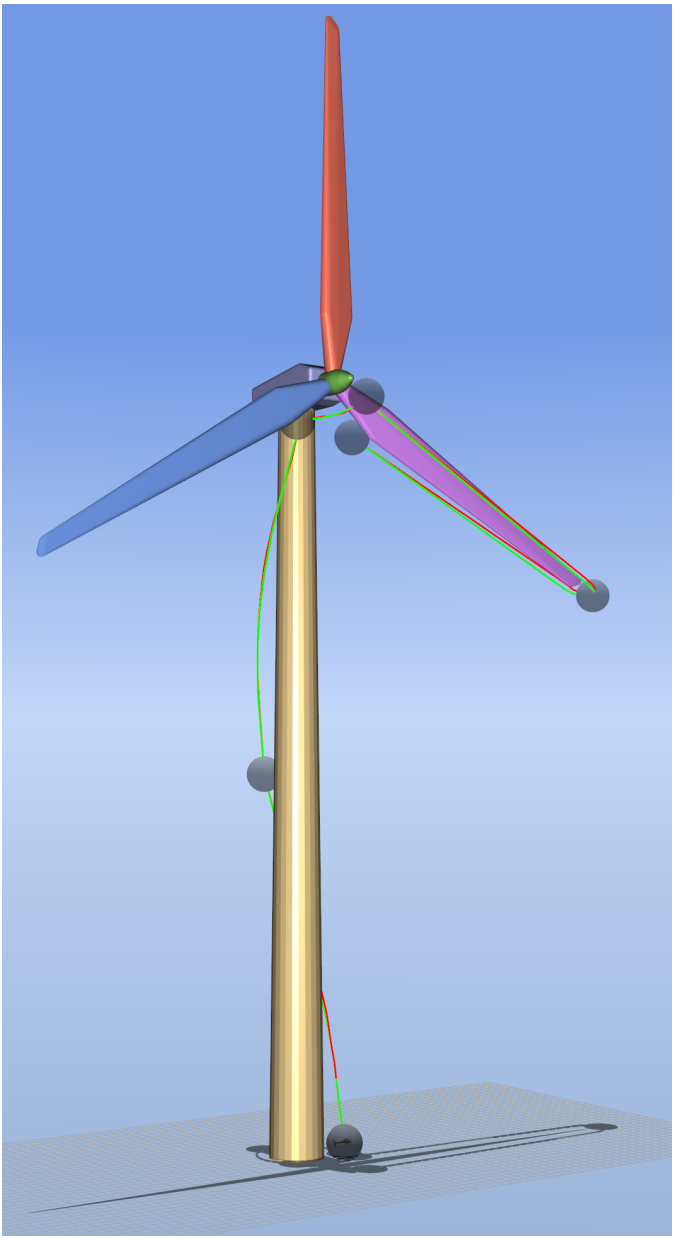


Figure 3. Front view of wind turbine. The grid at the base is 1 x 1 m. The approximated path is drawn in red. The geodesic path is drawn in green. Via points are indicated with a gray sphere.

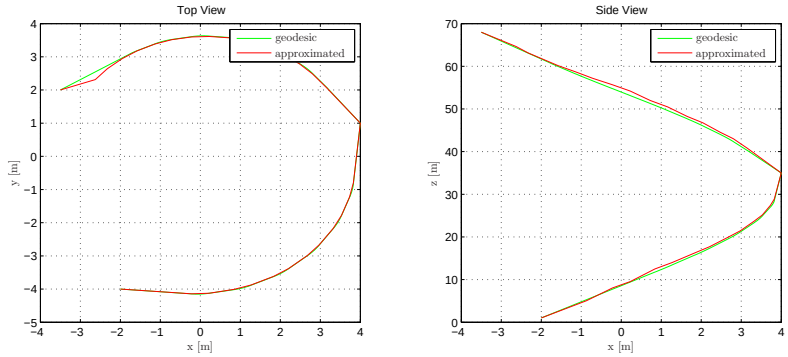


Figure 4. Side view (vertical plane) and top view (horisontal plane) of path between vp 1 and vp 3.

visibility graph that violates another configuration space are removed when combining the visibility graphs. Ideally new nodes and links should be added where these obstacles intersect, since they form ridges where local minima are likely. This should result in a better approximation of the shortest path, but is left as an issue to be addressed in further work.

Vp 4-6 The path between vp 4 and 6 starts off in front of the rightmost rotor blade, proceeds to its tip at vp 5, and continues to vp 6 behind the blade. The approximated and geodesic path can be seen both in Figure 9 and Figure 8. Although the approximated path is somewhat different from the geodesic, they have similar length. The approximated path is just 0.06 % longer than the geodesic path. Since there might be several local minima for the path planning problem, the approximated path might be entirely different and still be nearly as short at the geodesic path.

IV. Conclusion

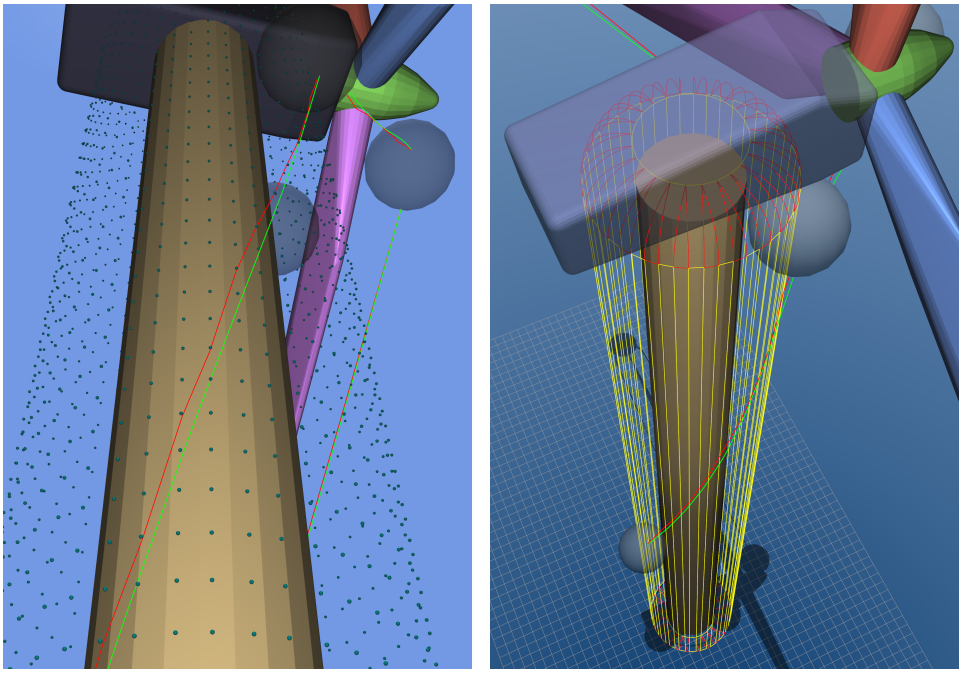
The shortest path around polyhedral configuration space obstacles are a combination of paths on the surface of the obstacles and supporting lines. Finding such a shortest path in 3D is NP-complete, since it does not in general traverse only vertices of the polyhedron, as in the 2D case, but also points on its edges. If one suffices with an approximately shortest path, a finite set of points can be selected near extremities of the configuration space. From these points a visibility graph can be build that can be searched to give an approximated shortest path.

A method for generating a 3D visibility graph that can be searched to provide a path in an environment with multiple obstacles was presented. The generated VG is composed of obstacle and supporting lines visibility graphs. The obstacle graphs are based on first constructing continuously differentiable configuration space obstacles from geometric workspace obstacle, then generating a local visibility graph for each configuration space obstacle that approximates their surface without intersecting it. The supporting lines graphs add links between different obstacle VGs and via points. It was shown that a usable path can be found by searching the visibility graph using a graph search algorithm and that the found solution is close to the optimal for the given test scenario.

For a given resolution the presented method gives a path which length is close to optimal, and converges towards optimal by increasing resolution. This means that the proposed method might find a path that takes different route than the optimal path, but the approximated path will be almost as short as the optimal.

References

[1]Shim, D. H., *Autonomous exploration in unknown urban environments for unmanned aerial vehicles*, University of California, Berkeley, 2005.



(a) Section of path between vp 2 and vp 3. The visibility graph nodes for the tower obstacle are drawn as blue diamonds.

(b) Tower and its patches. Rectangles with alternating colors show cylinder patches, red triangles show sphere patches, and yellow rectangles show the facet patches.

Figure 5. Paths near tower. The approximated path is drawn in red. The geodesic path is drawn in green. Via points are indicated with a gray sphere.

[2]Office of the Secretary of Defense, *Unmanned aircraft systems roadmap 2005-2030*, U.S. Army, 2005.

[3]Pachter, M. and Chandler, P. R., *Challenges of Autonomous Control*, IEEE Control Systems Magazine, vol. 18, no. 4, 1998.

[4]Chum, O., *Two-View Geometry Estimation by Random Sample and Consensus*, PhD Thesis, 2005.

[5]Marton, Z., Pangercic, D., and Blodow, N., *General 3D Modelling of Novel Objects from a Single View*, IEEE International Conference on Intelligent Robots and Systems, 2010.

[6]Dijkstra, E. W., *A note on two problems in connection with graphs*, Numerische Mathematik 1: 269-271, 1959.

[7]Hart, P. E., Nilsson, N. J., and Raphael, B., *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics SSC4: 100-107, 1968.

[8]Lozano-Perez, T. and Wesley, M. A., *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles*, Communications of the ACM, 22, 1979.

[9]Canny, J. and Reif, J. H., *New lower bound techniques for robot motion planning problems*, In Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci., 1987.

[10]Lozano-Perez, T., *Spatial Planning: A Configuration Space Approach*, IEEE Transactions on Computers, 32, 1983.

[11]O'Rourke, J., *Computational Geometry in C*, Cambridge University Press, 1998.

[12]Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. T., *The quickhull algorithm for convex hulls*, ACM Trans. Mathematical Software 22, 1996.

[13]Buss, S. R. and Fillmore, J., *Spherical Averages and Applications to Spherical Splines and Interpolation*, ACM Transactions on Graphics 20, 2001.

[14]Haines, E., *Point in polygon strategies*, Graphics gems iv, 1994.

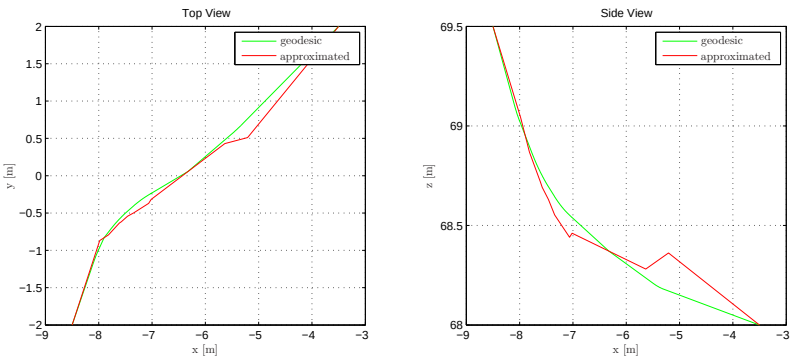


Figure 6. Side and top plot of path between vp 3 and vp 4.

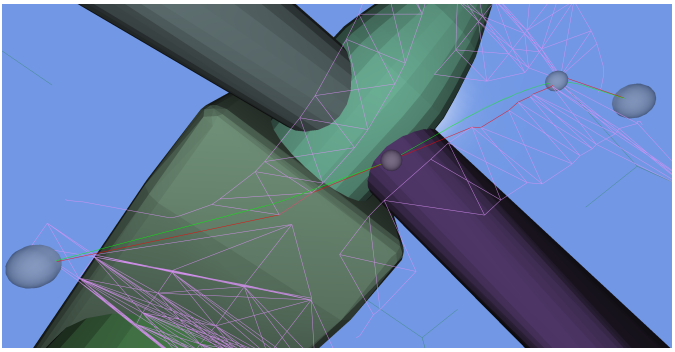


Figure 7. Path between vp 3 and vp 4. The approximated path is drawn in red and the geodesic path is drawn in green. A section of the link set is drawn for each obstacle.

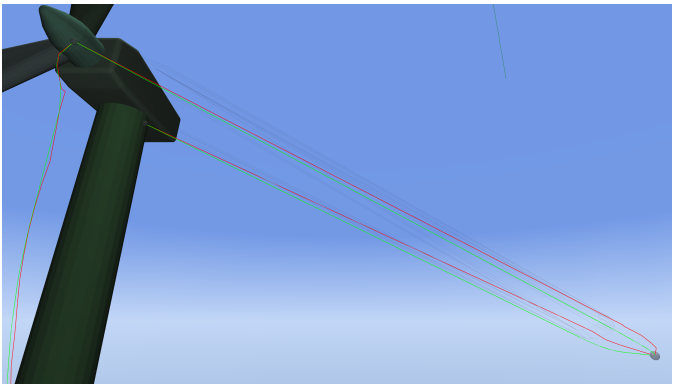


Figure 8. Path between vp 4 and vp 6. The approximated path is drawn in red and the geodesic path is drawn in green. The rotor is drawn as wire frame to allow easier view of the path between vp 5 and 6.

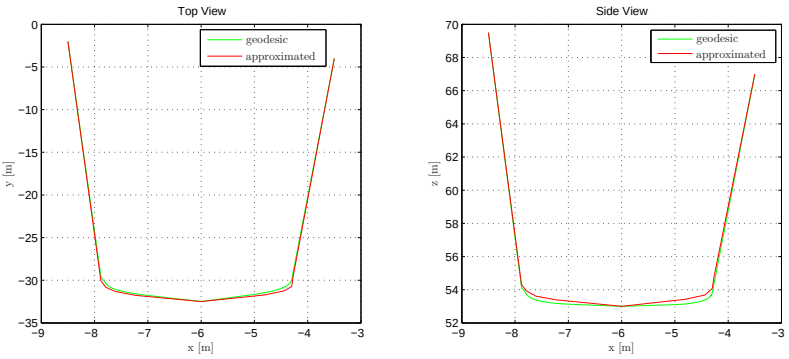


Figure 9. Side and top plot of path between vp 4 and vp 6.

Paper B

Generating Approximative Minimum Length Paths in 3D for UAVs

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

This paper was accepted by:
CATEC Research, Development and Education on Unmanned Aerial Systems, 2011.
Resubmitted to:
IEEE Intelligent Vehicles Symposium, 2012

Generating Approximative Minimum Length Paths in 3D for UAVs

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

***Abstract*— We consider the challenge of planning a minimum length path from an initial position to a final position for a rotorcraft. The path is found in a 3-dimensional Euclidean space containing a geometric obstacle. We base our approach on visibility graphs which have been used extensively for roadmap based path planning in 2-dimensional Euclidean space. Generalizing to 3-dimensional space is not straightforward, unless a visibility graph is generated that, when searched, will only provide an approximate minimum length path. Our approach generates such a visibility graph that is composed by an obstacle graph and two supporting graphs. The obstacle graph is generated by approximating a mesh around the configuration space obstacle, which is build from the convex hull of its work space counterpart. The supporting graphs are generated by finding the supporting lines between the initial or final position and the mesh. An approximation to the optimal path can subsequently be found using an existing graph search algorithm. The presented approach is suitable for fully known environments with a single truly 3-dimensional (not merely "raised" 2-dimensional) obstacle. An example for generating a nearly minimum length path for a small-scale helicopter operating near a building is shown.**

Path planning and trajectory generation are fundamental areas for UAS development and both provide the ability to figure out a way to efficiently and safely travel through an environment under a set of constraints. Specifically, for tasks such as surveillance, inspection, aerial mapping, etc., small-scale autonomous helicopters are increasingly being used. In many such tasks it is advantageous to operate in close proximity to the obstacles or to follow their surface.

A. Background

The practical use of the presented work is to obtain methods for path planning that can be used for operating a small-scale helicopter in an environment constrained by obstacles. The presented method is near-optimal, that is, able to produce a path close to the Euclidean shortest path inside a space. This space is constrained by a single obstacle such that no parts of the vehicle may intersect the obstacle at any time. In this space the vehicle is represented by its bounding sphere, which is centered in center-of-mass. The presented approach for path planning use visibility graphs (VG). Theses VGs are based on describing the obstacles in a configuration space (CS), which in turn are based on the work space (WS). The generation of those CS obstacles and VGs are the main contribution of this paper.

F. Schøler is with Institute of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, Denmark fls@es.aau.dk

A. la Cour-Harbo is with Institute of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, Denmark alc@es.aau.dk

M. Bisgaard is with Institute of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, Denmark bisgaard@es.aau.dk

The WS obstacle can be build from any set of vertices such as a point-cloud from sampled data. The WS obstacle is made convex since non-convex parts are not relevant when searching the VG for a solution. In fact, when computing the shortest path between two points in the CS, the shortest path is a series of connected edges that are either on the convex hull of the obstacle, or tangents from the points to the obstacle.

B. Previous work

Previous work on how to practically, automatically, and efficiently generate a configuration space from a 3D point-cloud in WS seems limited to building primitive solids or geometric representations, and in many path planning publications an existing configuration space is simply assumed.

Visibility graphs for 2D path planning have been used extensively. However finding an optimal solution to the general path planning problem in 3D is NP-complete, see [1], [2]. The difficulty of the problem is that the shortest path around a polyhedral obstacle does not in general traverse only vertices of the polyhedron, but also points somewhere on the edges of the polyhedron. The concept of adding additional vertices along these edges in configuration space so that no edge vertices are spaced more than a specified maximum length is introduced by Lozano-Perez and Wesley [1]. This approach generally results in a good approximation to the optimal path.

Lozano-Perez [3] presents an algorithm for the 2D case that use the Minkowski sum (see [4] for details on Minkowski sum) to generate a CS obstacle from a polygonal obstacle and a polygonal vehicle. While CS obstacles could be generated by approximating the vehicle with a geodesic dome, the obstacles are described by a sequence of algebraic equations, which makes the optimization problem difficult to solve without resorting to numerical methods. Also Bajaj and Kim [5] describe algebraic algorithms for representing the boundary of CS obstacles as patches of quadratic surfaces.

C. Present work

We propose a method for generating a VG from 1) a point-cloud that represent the WS obstacle, 2) a vehicle bounding sphere with radius r_{bs}, and 3) an initial and final position for the vehicle. The generalized visibility graph $\mathrm{VG}(N, L)$ contains a node set N and a link set L of links between node pairs. Each node in the graph is assigned a possible configuration (or position), and each link represents a visible (linear) connection between them. The VG is composed of two parts; an obstacle visibility graph VG_o that links node-pairs near the obstacle surface, and a supporting visibility

graph VG_{vp} that connects the initial and final position (denoted "via points") to the CS obstacle by supporting lines:

$$\text{VG}_f = \text{VG}_o \cup \text{VG}_{vp}$$

An example of such a graph is shown in Figure 1. The advantage of separating the obstacle and via point (VP) graphs is that while both graphs must be generated when creating the initial full VG, only VG_{vp} is created when either the obstacle is moved or rotated or either via point is moved. The full VG is generated such that it contains only links that are relevant when computing the Euclidean shortest path between via points, and subsequent pruning is not necessarily. The full VG can be searched to obtain the shortest path using a graph search algorithm, e.g. Dijkstra's algorithm, see [6] or the A* algorithm, see [7]. This part is not the focus of this research and will not be discussed further.

I. METHOD

The full VG is used to find an approximative minimum length path between two positions. It is composed of an obstacle graph and supporting graphs. The subject of generating the obstacle graph VG_o is first treated in Section I-A. The supporting graph VG_{vp} is then treated in Section I-B.

A. Obstacle Visibility Graph

An obstacle visibility graph VG_o is based on the configuration space of a spherical vehicle moving amongst convex, polyhedral obstacles, i.e. obstacles described by facets. Thus, CS obstacles are grown as the Minkowski sum of a sphere and convex WS polyhedra generated from the convex hull (see e.g. [8] for a robust $O(n^2)$ algorithm for finding the convex hull) of the point-cloud. An example is shown in Figure 2 where this is done in two steps. The first step, shown in Figure 2(a), is to generating the CS obstacle by first translating each facet of the WS obstacle along its outward pointing normal by the vehicle bounding sphere radius r_{bs}. The subsequent step, shown in Figure 2(b), is to fill the occurring gaps between edges of translated facets by cylindrical patches, and gaps at facet vertices by spherical patches. Any such generated CS obstacle will have a surface with G1 continuity. As shown in Figure 2(c) the VG of a CS obstacle can be seen as a mesh wrapped around (and fully enclosing) that obstacle. The mesh points act as nodes in the graph, and the links are edges of a convex hull of the obstacle. The mask size of the mesh is given by the desired accuracy of the near-optimal path. To fully enclose the obstacle, these nodes must be located slightly above the obstacle surface. This distance is denoted r_e and increases as the desired accuracy decreases.

The nodes (mesh points) are a combination of the points from the edges of the patches and points internally to the patches. Only cylindrical and spherical patches are used when generating the nodes. Points in facets can be skipped since interior points will not be part of an optimal solution, and facet edges are shared with cylindrical patches. Once the node set is generated for a patch, the link set can be determined using the convex property of the CS obstacle. Convexity means that links exist only between nodes of the same patch. In the following, we describe how to find the nodes for the two patch types.

1) Sphere patch vertices: The vertices on a sphere patch has four separate origins as seen by the black, white, green, and blue vertices in Figure 2(c). The black endpoint vertices form the sphere patch polygon, the white vertices sit along the polygon edges, and green or blue vertices are inside the polygon. The black vertices are given by the neighboring translated facets, while the other types are interpolated to achieve a sufficiently high resolution of the VG. The blue central vertex is used when interpolating the green vertices.

To obtain the interpolated nodes, let N_{sn} be the number of neighboring cylindrical patches. The spherical polygon is then composed of N_{sn} edges between N_{sn} vertices. The set of these vertices is denoted $\{v_i\}$ for $i = 0, \ldots, N_{\text{sn}}-1$. Using a central vertex inside the polygon, the spherical patch polygon can now be decomposed into N_{sn} spherical triangles. Each triangle uses a different edge but all share the central vertex.

A central vertex that minimize spherical distance to the vertex set is given as

$$\underset{v_c}{\text{argmin}} \sum_{i=0}^{N-1} w_i \text{dist}_{\text{geo}}(v_c, v_i)^2 \ ,$$

where $\text{dist}_{\text{geo}}(v_c, v_i)$ is the geodesic distance from v_c to v_i, and w_i are non-negative weights that sum to 1. This problem is well-defined for vertices on a hemisphere and a fast iterative algorithm with quadratic convergence rate exist, see [9].

All remaining nodes in the patch are found by two consecutive interpolations. The first interpolation is along the edges of the sphere patch polygon to find a set of vertices v_a. The second interpolation is along the edges formed by each of these vertices and the central vertex.

The number of vertices generated by interpolation is controlled by the parameter $l_{\max}$ that determines the maximum allowed surface distance between two neighboring vertices. Since the VG must fully enclose the obstacle, a lower interpolation resolution requires a larger patch radius to ensure no links intersect the obstacle. Both factors affect the near-optimality of solution, since a smaller obstacle and more nodes are more likely to result in a solution closer to optimal.

The number of nodes $N_{\text{se}}(i)$ along an exterior edge i of a spherical polygon is

$$N_{\text{se}}(i) = \lceil (r_{bs} + r_e) \arccos(v_i \cdot v_{i+1}) / l_{\max} \rceil \ ,$$

where $\{v_i\}$ is the set of consecutive endpoint vertices in the spherical polygon. Using the following function for spherical linear interpolation

$$p(v, w, k) = \left(1 - (v \cdot w)^2\right)^{-\frac{1}{2}} \left(\sin\left(k \arccos(v \cdot w)\right) w + \sin\left((1-k) \arccos(v \cdot w)\right) v\right) \ ,$$

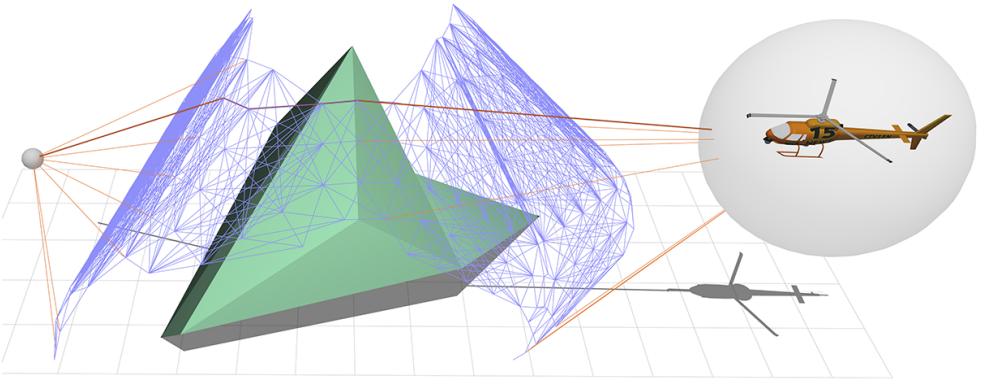
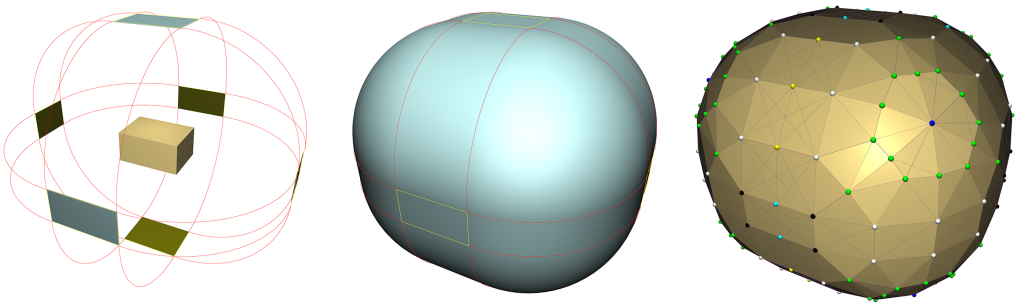


Fig. 1. The VG_f for a helicopter flying between two via points around an obstacles. The blue segments are parts of the obstacle visibility graph VG_o. The orange segments are parts of the supporting graph VG_{vp}. The red shortest found path connects the two via points at the gray spheres. The larger sphere represents the vehicle bounding sphere. The CS obstacle is not shown.



(a) Cuboid in workspace and translated facets (b) Cuboid in configuration space (c) Cuboid visibility graph

Fig. 2. The configuration space and visibility graph generated from a cuboid. VG nodes are colored according to origin.

the vertices on the arc between v_i and v_{i+1} are given by

$$v_a(i,s) = p\big(v_i, v_{i+1}, s/N_{\mathrm{se}}(i)\big)$$

and then (with a slight abuse of notation)

$$V_a = \bigcup_{i=0}^{N_{\mathrm{sn}}-1} \bigcup_{s=0}^{N_{\mathrm{se}}(i)} v_a(i,s) \ .$$

In Figure 2(c) the white and black nodes are the set V_a.

Interior nodes are then added by spherical linear interpolation between v_c and the vertices in V_a. The required number of nodes for each edge is given by

$$N_{\mathrm{si}}(i,s) = \lceil (r_{bs} + r_e) \arccos\left(v_a(i,s) \cdot v_c\right)/l_{\max} \rceil \ .$$

The vertex for each interpolation is then

$$n_{sp}(i,s,t) = p\big(v_a(i,s), v_c, s/N_{\mathrm{si}}(i,s)\big) \ ,$$

and the total set of vertices for the spherical patch is

$$V_s = v_c \cup \bigcup_{i=0}^{N_{\mathrm{sn}}-1} \bigcup_{s=0}^{N_{\mathrm{se}}(i)} \bigcup_{t=0}^{N_{\mathrm{si}}(i,s)} n_{sp}(i,s,t) \ . \qquad (1)$$

The interior nodes in this set are shown in Figure 2(c) as green nodes. The interpolation functions above have no singularities since the angular spacing between consecutive vertices is always less than π for any obstacle that extends three dimensions.

2) Cylinder patch vertices: From the sphere patch a set of vertices was found at the arc edges of the cylinder patch. Both arc edges have the same number of vertices, since the edges have the same length and are both interpolated using $l_{\max}$. Assume now that the vertices v at one end is paired one-to-one with the vertices w at the other end, in an ordered manner starting at the same facet edge. Since the height of the cylinder is constant, the number of vertices along this height must be

$$N_{\mathrm{c}} = \lceil ||w_0 - v_0||/l_{max} \rceil \ .$$

Each interpolation point for pair i is given by

$$n_{\mathrm{cp}}(i,j) = v_i + (w_i - v_i)j/N_{\mathrm{c}}$$

VP no.	1	2	3	4	5
X (m)	-1	0	10	-10	10
Y (m)	0	2	4	-1	1
Z (m)	-11	10	-10	-1	4

TABLE I
POSITION OF THE 5 VPS

for $j = 1, \ldots, N_c - 1$ and for $i = 0, \ldots, N_{se}(u)$, where u is the index of the corresponding sphere patch polygon edge. Now the set of all cylinder patch vertices is given as

$$V_c = \bigcup_{i=0}^{N_{se}(u)} \bigcup_{j=1}^{N_c-1} n_{cp}(i,j) \quad (2)$$

3) Full obstacle visibility graph: The VG for the obstacle is now given as the union of the sets V_s in (1) and V_c if (2). The links of each patch is found by connecting all its vertices. Links between vertices on the same facet edge can be removed unless both are endpoint nodes, since such links will not be part of the shortest path.

B. Via Points Visibility Graph

When generating a VG between a via point (initial or final vehicle position) and the obstacle nodes, only links on supporting lines are part of the shortest path and hence included.

In order for a link L_n from a node n_e to a node n_s on a polyhedron surface S to be on a supporting line, at least one of the N_f facets of n_s must be 'visible' and at least one facet must be non-'visible'. A facet k of n_s is visible from node n_e if

$$f_v(n_e, n_s, k) = (n_e - n_s) \cdot g(k)$$

is ≥ 0. The function $g(k)$ gives the outward-pointing normal vector for a facet k of n_s. This means that L_n is a on a supporting line iff there exists $j = 0, \ldots, N_f - 1$ such that $f_v(\cdot, \cdot, j) \geq 0$ and there exists $k = 0, \ldots, N_f - 1$ such that $f_v(\cdot, \cdot, k) < 0$.

We now define the full graph as

$$\mathrm{VG_{vp}} = \mathrm{tg}(\mathrm{vp}_0, \mathrm{VG}_o) \cup \mathrm{tg}(\mathrm{vp}_1, \mathrm{VG}_o) \cup \mathrm{int}(\mathrm{vp}_0, \mathrm{vp}_1) \ ,$$

where $\mathrm{tg}(p, G)$ is a function that gives the supporting VG from the supporting link set between p and G, and $\mathrm{int}(p_0, p_1)$ is a function that returns the VG between p_0 and p_1, if its link does not intersect the CS obstacle. Such a function can be based on segment/triangle intersection, see e.g. [10].

II. RESULTS

A test case is created where a helicopter must fly between the five via points in Table I, which are placed in a star shaped pattern. The paths must be planned such that the helicopter avoids a building, centered between the via points. The setup and resulting paths can be seen in Figure II. A small-scale helicopter with radius r_{bs} of 1.70 m is used. The interpolation parameter $l_{\max}$ is set to 0.75 m and r_e to 0.076 m. The full path through all VPs is found by solving the five path planning problems for connected VPs. This approximated shortest path p_a is then compared to the optimal path p_g. The optimal path is composed of geodesics and tangents on the CS obstacle and is calculated using a minimization algorithm. The building consists of 33 thousand vertices and 59 thousand triangles. The resulting WS obstacle has 128 vertices and 182 facets. In CS this obstacle consists of 182 facets, 128 sphere patches, and 308 cylinders patches. The first VG was generated in 553 milliseconds[1] with 4 thousand nodes and 53 thousand links. On subsequent planning, only the supporting graph needs to be updated. Updating the supporting graph took less than 3 milliseconds in all subsequent cases. From each VG a path was found in less than 500 milliseconds by searching the VG with Dijkstra's graph search algorithm.

Figure II shows overall good consistency between optimal and approximated paths, except for the initial case. In this case, it can be seen how a (local) minimum on one side of the chimney has been found, whereas the optimal path is located on the other side. Although different paths have been chosen, both have nearly same length. The optimal path is 27.4 meters while the approximated path is 27.9 meters, giving an increase in distance of less than 2%. A similar increase can be seen when comparing the full paths. The full optimal path has a length of 126.9 meters, while the full approximated path is 128.6 meters. To improve the solution, the interpolation parameter $l_{\max}$ can be reduced. To generate a smaller VG that is faster to search, $l_{\max}$ can be increased. However, theres is a limit on how small the VG will become for a given obstacle model. An example is the dome in the left part of the building in Figure II. Since all points on the sphere are part of its convex hull, a dome will add many smaller patches to the CS, which results in a higher density of nodes. To overcome this, a polygon reduction algorithm can be applied to the model before generating the workspace.

III. CONCLUSIONS

Finding an optimal solution to the path planning problem in 3D is NP-complete. The shortest path around a polyhedral configuration space obstacle does not in general traverse only vertices of the polyhedron, as in the 2D case, but also points on its edges. For a given resolution the presented method gives a path for which the length is close to optimal, and converges towards optimal by increasing resolution. This means that the proposed method might find a path that takes different route than the optimal path, but the approximated path should be almost as short as the optimal.

We proposed a method that generates a VG that is improved for finding a shortest path in an environment with a single obstacle. The applied method is based on first constructing the CS obstacle from the WS obstacle described by a point-cloud, then generating an obstacle VG for the CS obstacle that approximates its surface. This VG is combined with VGs that links the obstacle VG to the initial and final

[1] All tests were done on a single core of a 2.2GHz Intel Core 2 Duo laptop

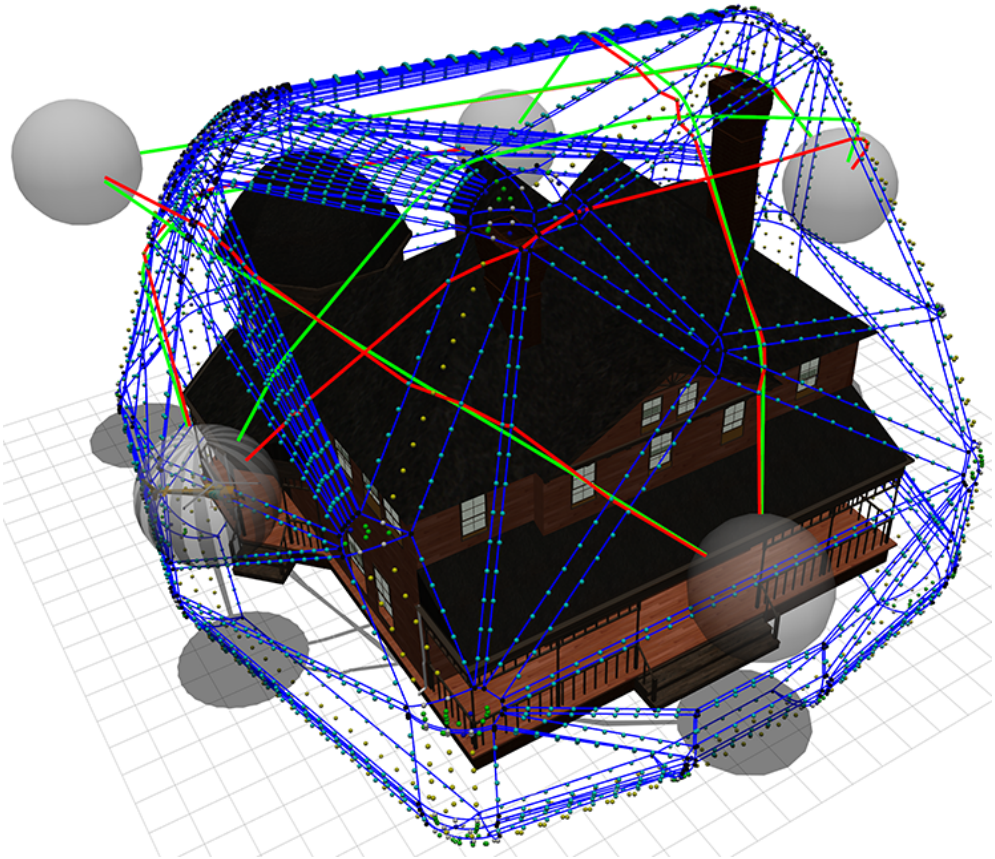


Fig. 3. Minimum length paths for a small-scale helicopter flying around a building through five VPs. These are indicated by the helicopter bounding spheres. Both the optimal (green) and approximated (red) path are shown. Smaller spheres represent the nodes in the obstacle VG. The obstacle CS is outlined by blue patches and facets.

configuration. If a path exists in the VG that connects the initial to the final configuration, it can be found using a graph search algorithm.

Results show that the method has good performance in terms of accuracy versus computational time. Visibility graphs for finding 5 paths amidst an obstacle with 600 patches were generated in less than 600 milliseconds. The resulting paths had a length within 2% of optimal.

Extending the method to work with multiple obstacles is possible, though this will require intersection testing of link and node candidates in the VG.

REFERENCES

[1] T. Lozano-Perez and M. A. Wesley. *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles*. Communications of the ACM, 22, 1979.

[2] J. Canny and J. H. Reif. *New lower bound techniques for robot motion planning problems*. In Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci., 1987.

[3] T. Lozano-Perez. *Spatial Planning: A Configuration Space Approach*. IEEE Transactions on Computers, 32, 1983.

[4] E. Oks and M. Sharir. *Minkowski Sums of Monotone and General Simple Polygons*. Discrete and Computational Geometry, 2006.

[5] C. Bajaj and M. Kim. *Generation of configuration space obstacles 1: the case of a moving sphere*. IEEE Journal of Robotics and Automation, 1988.

[6] E. W. Dijkstra. *A note on two problems in connection with graphs*. Numerische Mathematik 1: 269-271, 1959.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics SSC4: 100-107, 1968.

[8] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[9] S. R. Buss and J. Fillmore. *Spherical Averages and Applications to Spherical Splines and Interpolation*. ACM Transactions on Graphics 20, 2001.

[10] T. Moller and B. Trumbore. *Fast, Minimum Storage Ray-Triangle Intersection*. J. Graphics Tools 2(1), 1997.

Paper C

Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

This paper was submitted to:
Special issue on Micro-UAV Perception and Control in Journal of Autonomous Robots

Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning

Flemming Schøler · Anders la Cour-Harbo · Morten Bisgaard

Abstract We consider the challenge of planning a minimum length path for a set of destinations for an autonomous micro-UAV, that is, a sequence of positions starting from some current position to a sequence of goal positions. The path is found in a bounded 3D Euclidean space containing geometric obstacles represented by point-clouds. We present an algorithm that calculates the configuration space obstacles as patches and subsequently a visibility graph for pairs of subsequent via points. We focus on the ability to do fast replanning under limited changes to the environment. Visibility graphs have been used extensively for path planning in 2D Euclidean space. Since generalizing to 3D space is very difficult, we suffice with generating a visibility graph that, when searched, can only provide an approximate minimum length path. The generated visibility graph is composed of obstacle graphs and supporting graphs. The obstacle graphs can be thought of as a mesh that approximates the configuration space obstacle, which is build from the convex hull of its work space counterpart. The supporting graphs are generated by finding the supporting lines between 1) different obstacles and 2) the via points and the obstacles. An approximation to the optimal path can be found using an existing graph search algorithm. The performance and accuracy of the algorithm is evaluated based on a worst case scenario. We find that the generated path is near-optimal in terms of length and intrinsic straightness. Paths for a micro-UAV flying between several obstacles and destinations are also found in acceptable time in environments where existing obstacles transform and new obstacles appear.

Keywords UAS · micro-UAV · path planning · configuration space · visibility graph

Flemming Schøler
Aalborg University, Fredrik Bajers Vej 7C, Aalborg University, 9220 Aalborg, Denmark
Tel.: +45-9940-8702
Fax: +45-9815-1739
E-mail: fls@es.aau.dk

Anders la Cour-Harbo
Aalborg University, Fredrik Bajers Vej 7C, Aalborg University, 9220 Aalborg, Denmark
Tel.: +45-9940-8702
Fax: +45-9815-1739
E-mail: alc@es.aau.dk

Morten Bisgaard
Aalborg University, Fredrik Bajers Vej 7C, Aalborg University, 9220 Aalborg, Denmark
Tel.: +45-9940-8702
Fax: +45-9815-1739
E-mail: bisgaard@es.aau.dk

1 Introduction

Unmanned aerial systems (UAS) have become a preferred, indispensable, and increasingly used platform for many applications where manned operation is considered unnecessary, repetitive, or too dangerous[1, 2]. Both path planning and trajectory generation are fundamental areas for UAS development[3] and both provide the ability to figure out a way to efficiently and safely travel through an environment under a set of constraints. Specifically, for tasks such as surveillance, inspection, search and rescue, aerial mapping, cinematography, etc., small-scale autonomous helicopters are increasingly being used. In many such tasks it is advantageous to operate in close proximity to the obstacles or to cover their surface. Methods for describing these obstacles as geometric figures from real-world measurements have also been improved recently with 3D environment mapping data getting increasingly detailed and readily available including entire city maps. See i.e. [4, 5] for algorithms that reconstruct 3D obstacles from pictures.

Such recent advances allow better use of 3D maps for high-level path planning.

1.1 Background

We focus on obtaining a method for path planning that can be used for operating a small-scale UAV in an environment with multiple obstacles. The presented method is near-optimal for convex obstacles, that is, able to produce a path close to the Euclidean shortest path inside some space. The method is also suitable when multi-query searches must be done, for instance when a small part of the environment changes and replanning is necessary, or when multiple destinations exist.

We distinguish the term "trajectory" from "path" in that a path is without time and trajectory includes time. In this paper we are mainly concerned with the geometric problem of path planning. We consider the general trajectory planning problem for a UAS as having two levels; 1) a path that avoids larger static obstacles, such as buildings, mountains, etc., and 2) a trajectory along the path that also avoids smaller, possibly mobile object and at the same time takes the dynamics of the vehicle into account by generating a feasible state trajectory and possibly a related control trajectory. The present work is a method for generating a near-optimal solution to the first level. And because that level is static the method presented also prioritize the ability to easily do multi-query searches and replanning.

The presented approach for path planning is based on visibility graphs (VG). These VGs are build from the obstacles in configuration space (CS) that in turn are build on the obstacles in work space (WS). The aim of this is to transform the more difficult geometric problem of finding a path for a vehicle around the WS obstacles, into the easier dual problem of finding a path for a point around obstacles in CS. In CS the obstacles are generated by 'inflating' the WS obstacles using the geometric shape of the vehicle. The WS obstacle is build from any set of vertices such as a point-cloud from sampled data. The WS obstacle is made convex since non-convex parts are not relevant when searching the VG for a solution. In fact, when computing the Euclidean shortest path between two points in the CS, the shortest path is a series of connected edges that are either on the convex hull of the obstacle, or supporting lines from the points to the obstacle or between obstacles. An edge is called a supporting line if the line passing through its endpoints meets the respective obstacles only at its endpoints.

To obtain the actual path, the VG is searched using a graph search algorithm, for instance Dijkstra's algorithm [6] or A* [7]. This part is not the focus of this research, and will not be discussed in detail. Contrary to other popular methods such as the Rapidly-exploring random tree method, searching the graph will also reveal if no solution exists.

The configuration space for a micro-UAV, or indeed any other airborne vehicle, usually has six degrees-of-freedom, with three dimensions to describe position and three to describe orientation. However, by instead planning a path for the bounding sphere of the vehicle, the attitude will be irrelevant for determining a collision free path and the configuration space can be reduced to three degrees-of-freedom. The configuration space obstacles are defined as the Minkowski sum of the convex work space obstacles and the bounding sphere. They have a continuously differentiable surface and any shortest path (geodesic) on their surface will be continuously differentiable, which in turn will be easier to track for a vehicle with bounded dynamics. This is because a path that is continuously differentiable does not require instantaneous changes in velocity (although it does for acceleration, unless it is twice differentiable). We avoid the problem of risking collisions due to tracking/position error by increasing the vehicle bounding sphere accordingly.

1.2 Previous work

Previous work on how to practically, automatically, and efficiently generate a CS from a 3D point-cloud and a spherically shaped vehicle seems very limited, and in many path planning publications an existing configuration space is simply assumed.

The standard visibility graph is defined in a 2D polygonal CS [8]. The nodes of the graph correspond to the start location, the goal location, and the vertices of the configuration space obstacles on the convex hull. The graph edges are straight-line segments that connect pairs of nodes, if they are in line-of-sight of each other. The reduced visibility graph [9] reduces the size of such a graph by only retaining edges that correspond to supporting and separating line segments.

The classic three-step approach to path planning with VGs was introduced by Lozano–Perez and Wesley [10, 11], and consists of first calculating the CS, then building a VG, and finally searching the VG for a shortest path. The main difficulty lies in expressing the CS obstacles, which unlike the WS obstacles are not clearly defined. Lozano–Perez [11] generates a CS obstacle representation from the contact conditions of a polygonal obstacle and a polygonal vehicle in the 2D case. This method basically builds each CS obstacle from the Minkowski sum [12] of the work space obstacle and vehicle polygons. The approach can be ex-

panded to 3D by projecting the problem to 2D, but is cumbersome when an obstacle or the vehicle is not a polytope, or when obstacles are not axis-aligned. The concept of adding additional vertices along these edges in CS so that no edge is longer than a specified maximum length was mentioned by Lozano–Perez [10]. Their method generally results in a good approximation to the optimal path. Aronov and Sharir [13] presented a randomized algorithm that constructs the 3D configuration space for a convex polyhedral amidst convex polyhedral obstacles.

Finding an optimal solution to the general path planning problem in 3D is NP-complete[14]. The difficulty of the problem is that the shortest path around a polyhedral obstacle does not in general traverse only vertices of the polyhedron, but also points on the edges of the polyhedron. An optimal shortest path can be found using numerical methods. Schwartz and Sharir [15] used cylindrical algebraic decomposition for the first complete general planning algorithm. Canny [16] introduced a roadmap algorithm which improved this approach, while Kim and Bajaj [17] used algebraic patches of quadratic surfaces to representing the CS obstacles. These methods employ techniques from computational real algebraic geometry [18] that due to numerical considerations are very difficult to implement correctly. The running times of these algorithms also grow quickly with the number of primitives in the obstacle and vehicle representations. Many planning problems require thousands of primitives, which is well beyond what can be handled by algorithms that work directly with algebraic constraints on the obstacle region [19].

In more resent work, the CS obstacle representations are calculated as the convolution product between a function representing the vehicle and functions representing the obstacles [20]. Kavraki [21] computes this with Fast Fourier Transform and Shu [22] with polynomial transform. These approaches produce a bitmap that can be searched for an approximate shortest path.

Because of the difficulty in representing the CS obstacles for more than 2 dimensions, the focus has shifted the last 20 years to methods that that completely avoid such explicit representation. This focus has led to methods based on sampling such as the randomized path planner, the probabilistic roadmap planners, and the rapidly-exploring random trees that have successively solved challenging planning problems with in spaces with many dimensions.

1.3 Present work

This paper presents an algorithm that continues the earlier ideas of Lozano–Perez and Wesley [10]. Contrary to their approach, it builds explicit geometric CS obstacles that are suitable for easy generation of a 3D VG from a spherically shaped vehicle and a group of WS obstacles. This VG is a roadmap that links vertices that are mutually visible. This is done by test the link and node candidates for intersection. While this is a computational expensive operation, we show that our algorithm also efficiently builds and maintain the CS and VG under some changes in the WS. Each workspace obstacle is made convex and is build from vertices feed to the algorithm. These vertices do not need to be convex and could be from, for instance, the mesh of geometric obstacle or a point-cloud from sampled data.

Our method generates a VG inside the rectangular cuboid P with convex holes. The graph contains all links relevant for calculating the Euclidean shortest path between any two points in P. The graph is generated from 1) groups of point-clouds that each represents a WS obstacle, 2) a vehicle bounding sphere with radius r_{bs}, and 3) via points, that is, a set of the initial and consecutive goal positions for the vehicle. Figure 1 shows an example of such a VG.

The generalized visibility graph $\mathrm{VG}(N, L)$ contains a node set N and a link set L of links between node pairs. Each node in the graph represents a possible configuration, and each link represents a visible (linear) connection between them. We split the task of generating such graph VG_f into the union of three graphs; an obstacle visibility graph VG_{os} that links nodes near each CS obstacle surface, a supporting line visibility graph VG_{ot} that links nodes of different CS obstacles, and a supporting line visibility graph VG_{vp} that links via points to each other and to the nodes of the CS obstacles:

$$\mathrm{VG}_f = \mathrm{VG}_{os} \cup \mathrm{VG}_{ot} \cup \mathrm{VG}_{vp}$$

The advantage of separating the graphs is lower cost of maintaining VG_f under various changes. The overall stepwise approach is shown in Figure 2. While all graphs must be generated when creating the initial VG, only a partial update of the VG is required when obstacles are added or removed, even less when obstacles are just moved or rotated, and even further less when just via points change. The full VG is generated such that it contains only links that are relevant when computing the Euclidean shortest path between via points, and subsequent pruning is not necessarily. When generating VG_{os}, we first generate a surface graph VG_o for all N_n obstacles that contain only nodes and links required to plan a path near its surface. These graphs are subsequently combined and intersecting parts of their VGs are pruned. An obstacle visibility graph VG_o is based

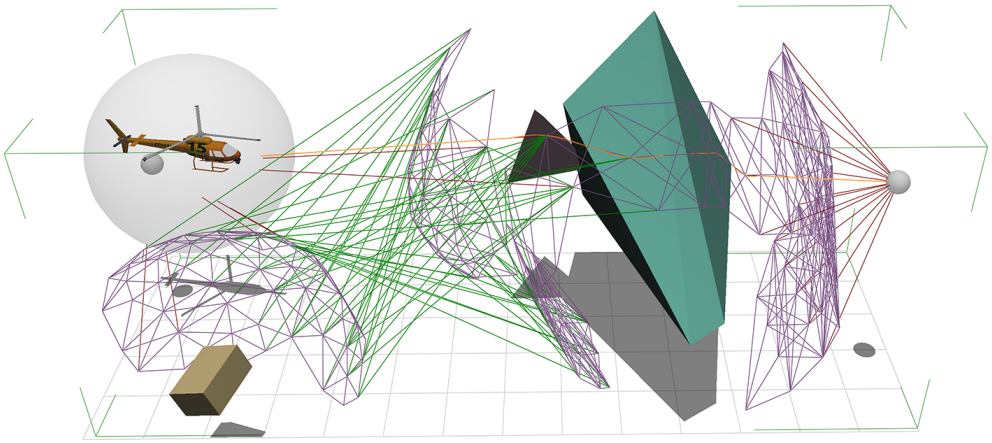


Fig. 1 The VG_f for a vehicle flying between two via points. The vehicle flies inside a cuboid with three obstacles. Obstacles are the convex hull of a set of WS vertices. All graphs are a subset of the CS. The surface graph VG_{os} is drawn in purple. The supporting graph VG_{ot} is drawn in green. The supporting graph VG_{vp} is drawn in red. The cuboid that is bounding the solution space is outlined by green segments. The shortest path is orange and connects the two via points drawn as gray spheres. The larger sphere represents the vehicle bounding sphere.

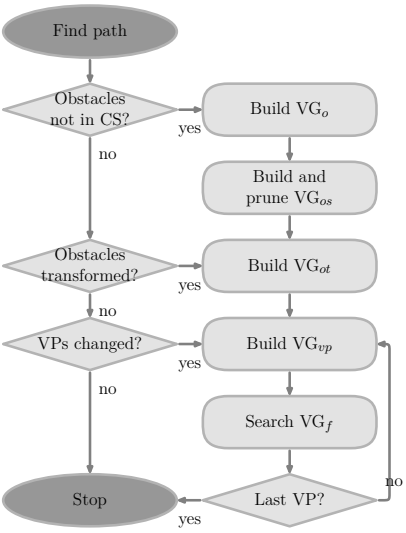


Fig. 2 Representation of the update process for generating the VG and finding a path.

on a spherical vehicle moving amongst convex, polyhedral obstacles, i.e. obstacles described by patches of planar surfaces. Thus, CS obstacles are grown as the Minkowski sum of a sphere and convex polyhedron. This is equivalent to generating the CS by translating each surface patch along its outward pointing normal by an amount equal to the radius of the sphere, while connecting pairs of translated patches by cylinders, and their endpoints by spheres. The method works for any WS obstacle shaped as polyhedron since these can be grown using only three types of geometrical shapes; spheres, cylinders, and planes. The obstacle VG forms a closed mesh that contains the CS, such that only neighboring nodes, i.e. nodes part of the same mesh polygon, are linked. In cases where CS obstacles intersect each other or the bounds of the CS, intersecting parts of their respective graph must be removed. As shown in Figure 1 the mesh of an obstacle graph will be open in such cases. Another example of a VG can be seen in Figure 11 and Figure 13.

2 METHOD

Section 2.1 explains how an obstacle visibility graph VG_o is generated from the surface of a configuration space obstacle. This CS obstacle is build from a WS polyhedron and is decomposed into patches of primitives, which are used for constructing a local VG for each obstacle. This obstacle VG is invariant under rotation and translation which makes subsequent replanning faster. In Section 2.2 all single graphs are combined and intersecting obstacles handled to form VG_{os}. A procedure for generating the supporting line graph VG_{ot} that links nodes of different obstacle VGs is described in Section 2.3. Section 2.4 explains how VG_{vp} that links via points with obstacles and other via points is generated. Finally, Section 2.5 briefly describes how

an approximated optimal path is found using a graph search algorithm.

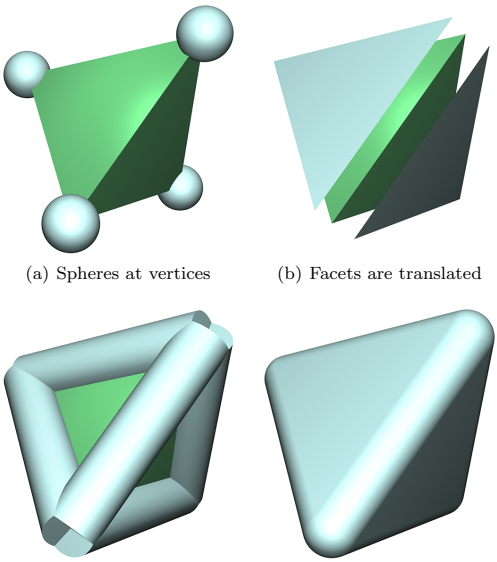
2.1 Obstacle Visibility Graph

In this section the methods for the two main steps for generating an obstacle VG are presented; generating the configuration space obstacles, and then generating the actual VG in configuration space. As previously described it is assumed that the vehicle is given as a sphere, and that the configuration space therefore has 3 dimensions, and that the configuration space obstacles are given as the Minkowski sum of the vehicle bounding sphere and the convex hull of work space obstacles. The configuration space obstacles are represented as a set of primitives (planes, cylinders, and spheres) generated by the algorithm from the set of work space obstacles. The second main step is generating a VG.

2.1.1 Configuration Space

The space of all allowed positions for the vehicle, denoted the configuration space, is constrained by the obstacles. For each obstacle in work space, a CS obstacle is created. The CS obstacle is the Minkowski sum of the vehicle bounding sphere and the convex hull of the WS obstacle polyhedron. Conceptually, this Minkowski sum can be obtained by replacing vertices of the WS convex hull with spheres, replacing edges with cylinders, and replacing facets with translated facets (translated along their normal). Both the amount of translation and the radii of the spheres and cylinders are fixed, and are equal to the vehicle bounding sphere radius. The resulting surface has G^1 continuity, since the tangent of any two intersecting curves on the surface are in the same tangent plane at the intersection point. An example of a configuration space obstacle grown from a workspace obstacle is shown in Figure 3.

In further details, the surfaces of the configuration space obstacles are composed of patches of primitive surfaces. These primitive surfaces are spheres, cylinders, and planes, and the patches are the "visible" parts of these primitives. A patch P is a simple, convex, closed polygon on the surface of a primitive. The edges of the polygon are formed by geodesic segments between N cyclic ordered vertices v_i, $i = 0$ to $N-1$. The last vertex v_N is assumed to be the same as the first, i.e. the polygon is closed. Any two neighboring patches of the configuration space obstacle intersect at one common edge. Since a patch is convex the geodesic between any two vertices in P lies entirely in P. For curved patches any geodesic is less than π radians in length. Each vertex of a patch is shared by two cylinder patches, one sphere patch and one planar patch. Each edge is either straight or on a great circle, in which case its curve is less than π. It follows that spherical patches only has cylindrical patches as neighbors. Cylindrical patches have two opposing sphere patches and two opposing facet patches as neighbors. Planar patches (also called facets) have only neighboring cylindrical patches.



(a) Spheres at vertices (b) Facets are translated

(c) Cylinders at edges (d) Full configuration space

Fig. 3 Configuration space is formed by the Minkowski sum of a polyhedron workspace obstacle and a vehicle bounding sphere.

The construction of the surface graph $\mathrm{VG}_{\mathrm{os}}$ starts with the set of endpoint vertices of all patches. Each vertex in a sphere patch is obtained by translating the same workspace vertex along the neighboring facet normals, so the sphere patch has the same number of vertices as number of neighboring workspace facets. The four vertices in the cylinder patch are obtained by translating the endpoints of a workspace edge along the two neighboring facet normals. The vertices in the facet are obtained by translating the workspace facet along its normal.

For the purpose of generating a surface VG for an obstacle a number of nodes are added to this given set of vertices. This addition is based on interpolation over patches. This set of vertices form a mesh around this obstacle that approximates its surface in CS. This mesh is used in generating the visibility graph for a single obstacle.

2.1.2 Obstacle Graph Nodes

As shown in the example in Figure 4 the VG for each CS obstacle can be seen as a mesh wrapped around that obstacle. This mesh completely contains the obstacle. The mesh points act as nodes in the graph, and the links are edges of a convex hull of the obstacle. The mask size of the mesh is given by the desired accuracy of the near-optimal path. Numerous algorithms exist for finding the facets that construct the convex hull, see e.g. O'Rourke[23] for a robust $O(n^2)$ three-dimensional implementation. The quickhull algorithm[24] is also efficient.

The mesh points originate from the surface of the patches, and the VG nodes are added from points obtained by interpolating both the edges and interior of the patches. Interior points in facets can be skipped, though, since they will not be part of an optimal solution. Once the node set is formed for a patch, the link set can be determined using the convex property of both the configuration space and assembly of patches. This means that links between nodes belonging to an obstacle are restricted to nodes of the same patch.

The points of the visibility graph nodes should not actually be located on the surface of the configuration space obstacle, since any such points would not have line-of-sight due to the curvature of the obstacle. Instead they are "lifted" above the surface, so that the configuration space obstacle is completely contained inside the hull spanned by these points. The needed amount of lift (we call this distance r_n) depends on the density of points at cylindrical and spherical patches, i.e. the more nodes the less lift is required. A desired resolution is given by this spacing of nodes $l_{\max}$, which determines the maximum allowed surface distance between two neighboring nodes on a surface grown from the work space obstacle by r_n. An upper bound for the necessary lift is calculated as the worst case scenario. This case is a regular quadrilateral spanned by neighboring nodes, spaced $l_{\max}$ on an arc with radius r_n. Here the center of the regular quadrilateral must have a distance to the workspace obstacle of r_{bs}. The maximum needed lift distance can be expressed as

$$r_n = \frac{r_{bs}}{\sqrt{\frac{1}{2}\cos(a_{\max})^2 + \frac{1}{2}}} ,$$

where

$$a_{\max}(r_n) = \min(l_{\max}, r_{bs}\pi\sqrt{2})/r_n$$

is the angle between two neighboring nodes, $l_{\max}$ is the maximum allowed arc distance between neighboring nodes, and r_{bs} is the vehicle bounding sphere radius. Thus r_n can be found as the solution to the following minimization problem

$$\underset{r_n \in (r_{bs}, \sqrt{2}r_{bs})}{\operatorname{argmin}} \left| r_n - \frac{r_{bs}}{\sqrt{\frac{1}{2}\cos\big(a_{\max}(r_n)\big)^2 + \frac{1}{2}}} \right| . \tag{1}$$

The objective is now to determine where to add nodes and links for each patch, such that as few nodes as possible are used in the visibility graph, while also limiting the volume of the configuration space obstacle.

2.1.3 Sphere patch vertices

The vertices on a sphere patch have four separate origins as seen by the black, white, green, and blue vertices in Figure 4(c). The black endpoint vertices form the sphere patch polygon, the white vertices lie along the polygon edges, and green or blue vertices are inside the polygon. The black vertices are given by the neighboring translated facets, while the other types are interpolated to achieve a sufficiently high resolution of the VG. The blue central vertex is used when interpolating the green vertices.

To obtain the interpolated nodes, we let N_{sn} be the number of neighboring cylindrical patches. The spherical polygon is then composed by N_{sn} edges between N_{sn} vertices. The set of these vertices is denoted $\{v_i\}$ for $i = 0, \ldots, N_{\text{sn}} - 1$. Using a central vertex inside the polygon, the spherical patch polygon can now be decomposed into N_{sn} spherical triangles. Each triangle uses a different edge but all share the central vertex.

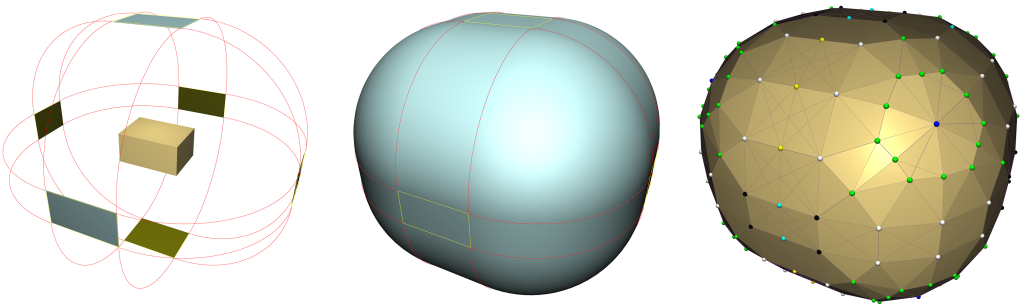
A central vertex that minimize spherical distance to the vertex set is given as

$$\underset{\mathbf{v_c}}{\operatorname{argmin}} \sum_{i=0}^{N-1} w_i \operatorname{dist}_{\text{geo}}(\mathbf{v_c}, \mathbf{v}_i)^2 ,$$

where $\operatorname{dist}_{\text{geo}}(\mathbf{v_c}, \mathbf{v}_i)$ is the geodesic distance from $\mathbf{v_c}$ to $\mathbf{v}_i$, and w_i are non-negative weights that sum to 1. This problem is well-defined for vertices on a hemisphere and a fast iterative algorithm with quadratic convergence rate exists, see [25].

All remaining nodes in the patch are found by two consecutive interpolations. The first is along the edges of the sphere patch polygon to find a set of vertices $\mathbf{v_a}$. The second interpolation is along the edges formed by each of these vertices and the central vertex.

The number of vertices generated by interpolation is controlled by the parameter $l_{\max}$ that determines the maximum allowed surface distance between two neighboring vertices. Since the VG must fully enclose the obstacle, a lower interpolation resolution requires a larger patch radius to ensure no links intersect the obstacle.



(a) Cuboid in workspace (b) Cuboid in configuration space (c) Cuboid visibility graph

Fig. 4 The steps that form the obstacle visibility graph for a cuboid. (a) shows a cuboid in workspace, its translated facets, and outline for the cylindrical and spherical patches. (b) shows the surface of the resulting configuration space obstacle. (c) shows nodes and links for a one choice of VG. Black nodes are endpoints of the translated workspace facets. White nodes are interpolated on the sphere patch edges. Blue are the central vertices. Green nodes are interpolated spherical triangle nodes. The yellow and cyan nodes are interpolated on the cylinder interior and straight edges, respectively.

Both factors affect the near-optimality of solution, since a smaller obstacle and more nodes are more likely to result in a solution closer to optimal.

The number of nodes $N_{\text{se}}(i)$ along an exterior edge i of a spherical polygon is

$$N_{\text{se}}(i) = \lceil r_n \arccos(\mathbf{v}_i \cdot \mathbf{v}_{i+1})/l_{\max} \rceil \ ,$$

where $\{\mathbf{v}_i\}$ is the set of consecutive endpoint vertices in the spherical polygon. Using the following function for spherical linear interpolation between two vertices $\mathbf{v}$ and $\mathbf{w}$, and with k being the interpolation parameter

$$\mathbf{p}(\mathbf{v}, \mathbf{w}, k) = \Big(1 - (\mathbf{v} \cdot \mathbf{w})^2\Big)^{-\frac{1}{2}} \big(\sin\left(k \arccos(\mathbf{v} \cdot \mathbf{w})\right) \mathbf{w} + \sin\left((1-k)\arccos(\mathbf{v} \cdot \mathbf{w})\right) \mathbf{v}\big) \ ,$$

the vertices on the arc between $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ are given by

$$\mathbf{v_a}(i, s) = \mathbf{p}\big(\mathbf{v}_i, \mathbf{v}_{i+1}, s/N_{\text{se}}(i)\big) \tag{2}$$

and then (with a slight abuse of notation)

$$\mathbf{V_a} = \bigcup_{i=0}^{N_{\text{sn}}-1} \bigcup_{s=0}^{N_{\text{se}}(i)} \mathbf{v_a}(i, s) \ . \tag{3}$$

In Figure 4(c) the white and black nodes are the set $\mathbf{V_a}$.

Interior nodes are then added by spherical linear interpolation between $\mathbf{v_c}$ and the vertices in $\mathbf{V_a}$. The required number of nodes for each edge is given by

$$N_{\text{si}}(i, s) = \lceil r_n \arccos\left(\mathbf{v_a}(i, s) \cdot \mathbf{v_c}\right)/l_{\max} \rceil \ .$$

The vertex for each interpolation is then

$$\mathbf{n_{sp}}(i, s, t) = \mathbf{p}\big(\mathbf{v_a}(i, s), \mathbf{v_c}, N_{\text{si}}(i, s)\big) \ , \tag{4}$$

and the total set of vertices for the spherical patch is

$$\mathbf{V_s} = \mathbf{v_c} \cup \bigcup_{i=0}^{N_{\text{sn}}-1} \bigcup_{s=0}^{N_{\text{se}}(i)} \bigcup_{t=0}^{N_{\text{si}}(i,s)} \mathbf{n_{sp}}(i, s, t) \ . \tag{5}$$

The interior nodes in this set are shown in Figure 4(c) as green nodes.

The interpolation functions above have no singularities since the angular spacing between consecutive vertices is always less than π for any obstacle that extends three dimensions.

2.1.4 Cylinder patch vertices

From the sphere patch a set of vertices was found at the arc edges of the cylinder patch. Both arc edges have the same number of vertices, since the edges have the same length and are both interpolated using $l_{\max}$. Assume now that the vertices $\mathbf{v}_i$ at one end is paired one-to-one with the vertices $\mathbf{w}_i$ at the other end, in an ordered manner starting at the same facet edge. Since the height of the cylinder is constant, the number of vertices along this height must be

$$N_{\text{c}} = \lceil ||\mathbf{w}_0 - \mathbf{v}_0||/l_{\max} \rceil \ .$$

Each interpolation point for pair i is given by

$$\mathbf{n}_{\text{cp}}(i, j) = \mathbf{v}_i + (\mathbf{w}_i - \mathbf{v}_i)j/N_{\text{c}}$$

for $j = 1, \ldots, N_{\text{c}} - 1$ and for $i = 0, \ldots, N_{\text{se}}(u)$, where u is the index of the corresponding sphere patch polygon edge. Now the set of all cylinder patch vertices is given as

$$\mathbf{v_c} = \bigcup_{i=0}^{N_{\text{se}}(u)} \bigcup_{j=1}^{N_{\text{c}}-1} \mathbf{n}_{\text{cp}}(i, j) \tag{6}$$

2.1.5 Full obstacle visibility graph

The VG for the obstacle is now given as the union of the sets $\mathbf{V_s}$ in (5) and $\mathbf{V_c}$ in (6). The links of each patch is found by connecting all its vertices. Links between vertices on the same facet edge can be removed unless both are endpoint nodes, since such links will not be part of the shortest path. We let $\mathbf{V_o}(p, f, i)$ refer to vertex i of facet f in patch p. The surface visibility graph VG_o for the obstacle is then given as the union of the complete graphs for each facet in the convex hull of the set of all vertices of all patches for this obstacle

$$\mathrm{VG}_o = \bigcup_{p=0}^{N_p-1} \bigcup_{f=0}^{N_f(p)-1} \bigcup_{\substack{i=0 \\ j=i+1}}^{N_v(p,f)-1} \mathrm{sg}(\mathbf{V_o}(p,f,i), \mathbf{V_o}(p,f,j))$$

The function $\mathrm{sg}(\mathbf{v_a}, \mathbf{v_b})$ builds a surface graph between vertices $\mathbf{v_a}$ and $\mathbf{v_b}$. The function returns the empty set if two vertices are on the same facet edge unless both are the endpoints.

2.2 Full obstacle surface visibility graph

We now build the VG_{os} that combines all N_o obstacle VGs. This can be calculated as the union of all obstacle graphs. However, intersecting nodes and links in graphs due to intersecting CS obstacles and nodes not in P must be removed. Facets of different workspace obstacles spaced less than $2r_{bs}$ intersect in configuration space. Furthermore, an obstacle VG might intersect the CS of a different obstacle at an earlier point, i.e. when their WS facets are spaced less than $r_n + r_{bs}$.

When removing intersecting parts of the VG using $\mathrm{ob}(G)$ and $\mathrm{int_G}(G, O)$, this pruned obstacle VG becomes

$$\mathrm{VG}_{\bar{o}}(i) = \mathrm{VG}_{o_i} - \bigcup_{j=0}^{N_V-1} \mathrm{ob}(\mathbf{V_{o}}_j, \mathrm{VG}_{o_i}, P) - \bigcup_{\substack{j=0 \\ j\neq i}}^{N_o-1} \mathrm{int_G}(\mathrm{VG}_{o_i}, \mathrm{O}_j) \; .$$

The function $\mathrm{ob}(V, G, P)$ calculates the subset of $G \notin P$. For a node in the G, it determines if the associated vertex is outside the bounding box of P. This function is trivial in our case since P is an axis-aligned cuboid, but any convex shape could be used without changing (7). The other function $\mathrm{int_G}(G, O)$ must give a subset of G that intersects an obstacle O.

The union of all these graphs finally gives

$$\mathrm{VG}_{os} = \bigcup_{i=0}^{N_o-1} \mathrm{VG}_{\bar{o}}(i) \; . \tag{7}$$

2.2.1 Intersecting links

The collision graphs between a graph and an obstacle $\mathrm{int_G}(G, O)$ is the compliment to the visibility graph. It determines the links of G that intersects an obstacle O. That is, the links of G that intersect the Minkowski sum of the vehicle bounding sphere and the WS obstacle O. This sum is only equal to the CS obstacle for convex WS obstacles, but it is usually reasonable to base such a function on the triangulated convex hull of $\mathbf{V_o}$.

The collision graph between a graph and an obstacle $\mathrm{int_G}(G, O)$, and two vertices $\mathrm{int_L}(\mathbf{v_a}, \mathbf{v_b})$ (used in the following sections) are done as a segment/triangle test by triangulating the hull of V_o. Since the obstacle VG is a (slightly larger) superset of the CS obstacle a link might intersect the VG hull without intersecting the configuration space obstacle. Consequently, a link might be subject to pruning even though it does not intersect the obstacle. However, having a sufficient high resolution on the obstacle visibility graphs will limit these cases. When it occurs the effect of pruning the intersecting link will usually only be a slightly longer path. A more accurate approach would be to perform intersection tests against sphere and cylinder solids located at vertices and edges of the work space obstacles, and the facets of the work space obstacles themselves. This approach should also work for non-convex workspace obstacles.

The basic idea of the intersection test done here is to test all links (segments) against all triangulated facets in all obstacles. Although efficient implementations exists for such tests, see i.e. [26], this step can be very expensive to perform even after reducing the number of segments through pruning. In an effort to reduce this cost octrees can be used. Octrees are also used for node intersection test that determines when a node from one obstacle is inside another obstacle. An octree is a spatial search tree for the triangles on which a form of bucket search can be applied. The tree is constructed by recursively subdividing the bounding box of the obstacle visibility graph into eight octants until a desired level is reached. Each octant at the final level is then linked with those triangles it intersects. Empty octants are also made aware if they are inside or outside the hull.

2.3 Full obstacle supporting line visibility graph

When multiple obstacles exist a visibility graph must be generated that links all obstacle pairs to allow travel between obstacles. Links in this graph can be pruned unless both vertices are on each others horizon ridge, i.e. they form a supporting line link. Since a supporting line

link between two different obstacles might still intersect a third obstacle, all supporting line links must be also tested for intersection.

We let the full supporting line graph be calculated as

$$\mathrm{VG}_{ot} = \bigcup_{\substack{\alpha,\gamma=0 \\ \beta=\alpha+1 \\ \gamma\neq\alpha,\beta}}^{N_o-1} \bigcup_{i=0}^{N_{\bar{v}}(\alpha)-1} \bigcup_{j=0}^{N_{\bar{v}}(\beta)-1} \big(\mathrm{tg}(\mathbf{V}_{\bar{o}(i,\alpha)}, \mathbf{V}_{\bar{o}(j,\beta)}) \setminus \mathrm{int_L}(\mathbf{V}_{\bar{o}(i,\alpha)}, \mathbf{V}_{\bar{o}(j,\beta)}, O_\gamma)\big) \quad (8)$$

that contains the links between pairs of non-pruned vertices (indexed by i and j) on their respective different obstacles (indexed by α and β). This is achieved using the function $\mathrm{tg}(\mathbf{v}_{\boldsymbol{\alpha}}, \mathbf{v}_{\boldsymbol{\beta}})$ that builds a supporting lines graph between vertices $\mathbf{v}_{\boldsymbol{\alpha}}$ and $\mathbf{v}_{\boldsymbol{\beta}}$ of obstacle α and β, and $\mathrm{int_L}(\mathbf{v}_{\boldsymbol{\alpha}}, \mathbf{v}_{\boldsymbol{\beta}}, O_\gamma)$ that builds a collision graph. This collision graph is the compliment of a visibility graph and contains a link between vertices $\mathbf{v}_{\boldsymbol{\alpha}}$ and $\mathbf{v}_{\boldsymbol{\beta}}$ if it intersects some other obstacle γ. A link is only added if its two vertices are supporting lines to each other, and the link does not intersect other obstacles. Only supporting line links are included since only these are can possible be parts of the shortest path. Whereas the latter intersection function is calculated using a relatively expensive segment/triangle test as described in Section 2.2.1, the supporting line links can be determined more easily.

2.3.1 Supporting line links

When determining the shortest path, it is sufficient to examine links connecting horizon ridges of obstacles. In other words, when moving from a point in front of an obstacle to a point on the obstacle, it is sufficient to look at the obstacle outline in order to find the optimal path around it. This concept can be used to remove links in the visibility graph that might be accessible, but are not part of any optimum solution.

In order for an link L_n from a vertex $\mathbf{v_e}$ to a vertex $\mathbf{v_s}$ on a polyhedron surface S to be a supporting line, at least one of the N_f facets of $\mathbf{v_s}$ must be 'visible' and at least one facet must be non-'visible'. For a vertex $\mathbf{v_s}$ on a polyhedron surface S to be located at the horizon ridge seen from another vertex $\mathbf{v_e}$, at least one of the N_f facets of v_s must be visible and at least one facet must not be visible.

A facet k of $\mathbf{v_s}$ is visible for vertex $\mathbf{v_e}$ if

$$f_v(\mathbf{v_e}, \mathbf{v_s}, k) = (\mathbf{v_e} - \mathbf{v_s}) \cdot \mathbf{g}(k)$$

is ≥ 0. The function $\mathbf{g}(k)$ gives the outward-pointing normal vector for a facet k of $\mathbf{v_s}$. The case of equality occurs when $\mathbf{v_e}$ is coplanar to the facet. In such cases the facet is visible. This means that a vertex $\mathbf{v_s}$ seen from $\mathbf{v_e}$ is on the horizon ridge if there exists $j = 0, \ldots, N_f - 1$ such that $f_v(\cdot, \cdot, j) \geq 0$ and there exists $k = 0, \ldots, N_f - 1$ such that $f_v(\cdot, \cdot, k) < 0$.

Links in VG_p are between a via point and an obstacle node. Links are pruned if the obstacle node is not on the horizon ridge seen from the via point.

2.4 Via points visibility graph

The via points $\mathbf{vp}_i$, $i = 0$ to $N_{\mathrm{vp}} - 1$ are a consecutive series of positions that the vehicle must visit. Ihe initial position is given by $\mathbf{vp}_0$. Finding the full path through all via points may be split into subtasks of finding $N_{\mathbf{vp}} - 2$ paths between segments $\mathbf{vp}_i\mathbf{vp}_{i+1}$. The supporting line VG that links a via point i to the pruned obstacle vertices in $\mathbf{V}_{\bar{o}}$ can be calculated as

$$\mathrm{VG}_{\mathrm{ovp}}(i) = \bigcup_{\substack{\alpha,\gamma=0 \\ \gamma\neq\alpha}}^{N_o-1} \bigcup_{j=0}^{N_{\bar{v}}(\alpha)-1} \big(\mathrm{tg}(\mathbf{vp}_i, \mathbf{V}_{\bar{o}(j,\alpha)}) \setminus \mathrm{int_L}(\mathbf{vp}_i, \mathbf{V}_{\bar{o}(j,\alpha)}, O_\gamma)\big) \, . \quad (9)$$

The visibility graph between the two via points of a path segment i is the compliment collision graph between the via points of this segment

$$\mathrm{VG}_{\mathrm{vpvp}}(i) = \left(\bigcup_{\gamma=0}^{N_o-1} \mathrm{int}_L(\mathbf{vp}_i, \mathbf{vp}_{i+1}, O_\gamma) \right)^c \, . \quad (10)$$

We now define the full graph for a segment s between two consecutive via points as

$$\mathrm{VG}_{\mathbf{vp}}(s) = \mathrm{VG}_{\mathrm{ovp}}(s) \cup \mathrm{VG}_{\mathrm{ovp}}(s+1) \cup \mathrm{VG}_{\mathrm{vpvp}}(s) \, ,$$

that is, the union of the supporting line graphs at both via points and the compliment collision graph between the via points of this segment. Subsequent segments beyond the first can be calculated faster if the environment remains static. This is because $\mathrm{VG}_{\mathrm{ovp}}(s)$ at the initial via point for a segment $s > 0$ was already calculated at the destination via point by the previous segment $s - 1$.

2.5 Searching Visibility Graph

Now given a visibility graph we aim at finding the shortest path. This path is found by searching through the graph with a cost associated to each link in the graph. The Euclidean distance is used for this cost, which makes this approach independent of vehicle dynamics. Since the visibility graph is constructed in 3D it is based on approximation through a finite number

of points located on surfaces of the configuration space obstacles. This in unlike the 2D case where (at least for obstacles with non-differentiable edges) a finite set of nodes is sufficient to guarantee an optimal path. Thus, the line-of-sight path is only locally shortest among the vertices in the visibility graph. It is not globally shortest as it would be in the 2D case. However, when Euclidean distance is used as metric then for the sampling density $l_{\max}$ tending to zero the shortest path tends to the globally optimum. Consequently, for any given value ε there exists a sampling density such that the difference between the global shortest path and the shortest path within the VG is smaller than ε.

Generally, a path with higher order of differentiability is preferred, and we consider feasibility of the generated path from this qualitative perspective only. While our method is based on approximations and does not consider the structural and dynamical capabilities of the vehicle, it does provide a closer-to-feasible path as the resolution improves. When the resolution improves the generated path becomes closer to continuously differentiable.

Table 1 Node spacing.

$l_{\max}$	0.250	0.375	0.500	0.675	1.000
r_n	1.015	1.033	1.056	1.078	1.180

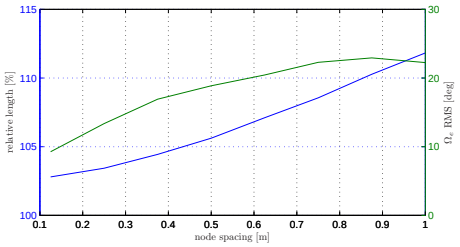


Fig. 7 Node spacing $l_{\max}$ impacts the quality of the approximated path. By quality we consider how much greater the length of the approximated path is compared to the optimal, and how large the average angle is between an incoming path and outgoing path at the nodes. The optimal path is composed of geodesics and tangents on the CS obstacle. It is found using a minimization algorithm that has limited usage for cases with multiple obstacles. A low node spacing $l_{\max}$ gives more nodes in the visibility graph, which on average also gives a shorter path and smaller changes in angle at the nodes.

3 Results

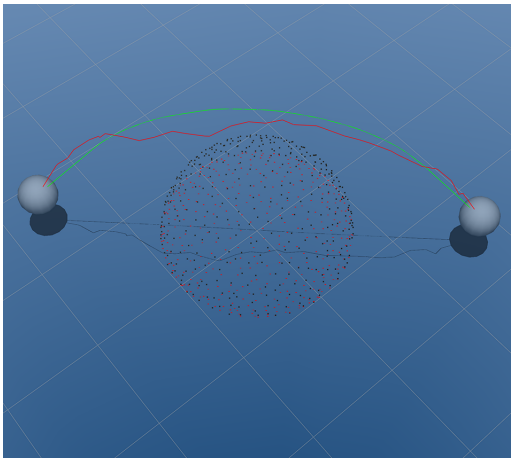
The test in Section 3.1 shows a limitation of the method. It shows which effect the node spacing parameter $l_{\max}$ has on the size of the visibility graph for a worst case obstacle. This is used for evaluating the performance of the method and accuracy of the resulting path. Results are also provided from two scenarios to demonstrate the behavior of the planning approach. The example in Section 3.2 demonstrates planning in a cluttered environment. It also shows how the VG is updated when a new obstacle is found. The example in Section 3.3 is a more practical example of planning near a large obstacle. This example demonstrates how the VG is updated when obstacles transform. All tests were done on a single core of a 2.2 GHz Intel Core 2 Duo laptop. We use a helicopter inside a sphere to represent the bounds of the micro-uav with maximum allowed tracking error included.

3.1 Performance and Accuracy Test

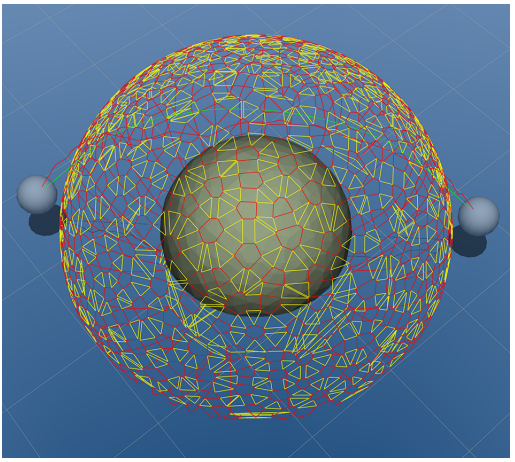
The test determines which effect the node spacing parameter $l_{\max}$ has on the size of the visibility graph. It also shows how the size of the visibility graph, measured in number of nodes and links for a single obstacle, relates to the number of vertices in the convex hull of the workspace. The test should also verify that lowering the node spacing results in a path closer to optimal when searching the graph and a path where the angle between incoming and outgoing path at the via points are lower. This is preferred as it is easier to track. For each vertex in the work space, several nodes are added to the visibility graph. This number is generally larger when a high number of facets that shares the vertex and when the distance to other vertices is high. This also means that if more vertices are added to the workspace without increasing volume much, fewer nodes per vertices should result. The test is done by generating a workspace obstacle for a point cloud. An example for one point cloud is shown in Figure 5. Each point cloud consists of between 4 to 300 randomly distributed vertices on a unit hemisphere. A vehicle radius r_{bs} equal to the grid size shown in Figure 5 is used when generating the configuration space. The length of the optimal path between two points located on opposite sides of the obstacle at $(1.75, 0.5, 1.5)$ and $(-1.75, 0.5, -1.5)$ is calculated. A visibility graph is generated and searched for each different pair of $l_{\max}$ and r_n listed in Table 1. This is repeated 6 thousand times and averaged to give a better statistical confidence.

3.1.1 Graph size

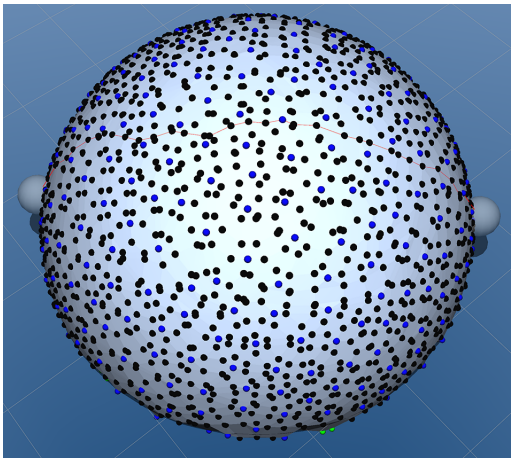
The size of the visibility graph is important when the VG is used to search for a shortest path, since the worst-case running time for Dijkstra's search algorithm on a



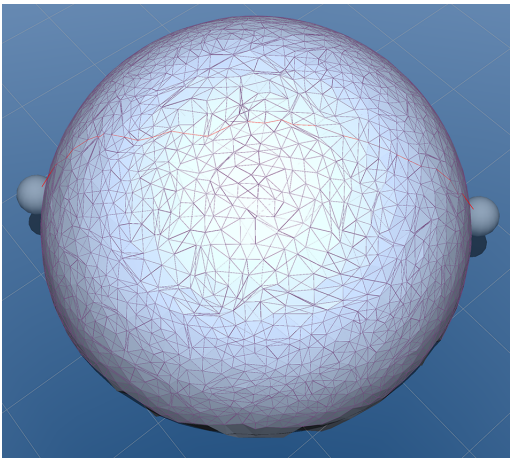
(a) A dense point cloud randomly distributed on a unit hemisphere. The geodesic between the two positions is drawn in green. The approximated path is drawn in red.



(b) The solid polyhedron in the center is the workspace obstacle formed by the convex hull of the point cloud. Outlines of the CS patches are also drawn. Facet patches are drawn in yellow, sphere patches in red, and cylinder patches in alternating red and yellow. Because the point cloud is located on the surface of a sphere, all vertices are used by the convex hull, and a high number of patches result.



(c) The configuration space obstacle and visibility graph nodes. The nodes origin mostly from the endpoints of the patches (black) and the central vertex of the sphere patches (blue). Because of the small are of each patch, only a few interpolated nodes are visible at the bottom where the surface is more flat.



(d) The configuration space obstacle and visibility graph links between neighboring nodes.

Fig. 5 Building an obstacle visibility graph from a point cloud located on the surface of a unit hemisphere.

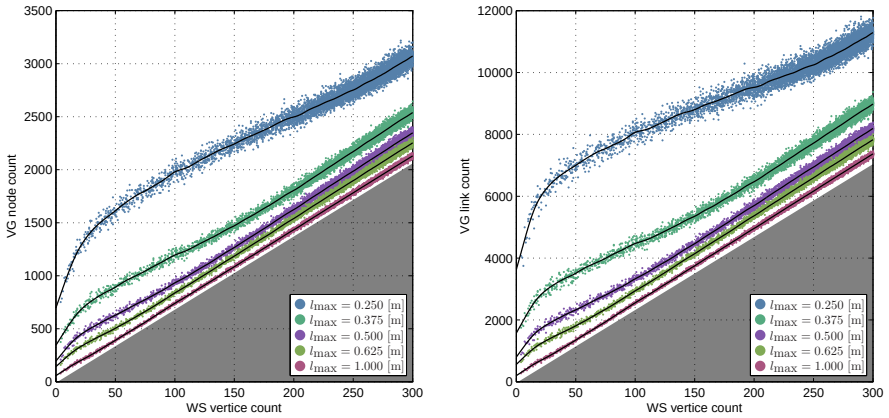


Fig. 6 These plots show the size of the generated visibility graph for point clouds on a unit hemisphere, for a node spacing $l_{\max}$ ranging from 0.25 to 1. The black curves show an average for each case of $l_{\max}$. The left plot shows the total number of nodes, and the right plot shows the total number of links between these nodes. The gray area shows the lower bound.

graph with n nodes is $O(n^2)$. The size of the obstacle visibility graph is affected by the node spacing $l_{\max}$, and the shape, size and number of the WS obstacles. In this test a single unit hemispherical obstacle is used and different workspaces are generated from a varying number of vertices, sampled on its surface. This means that its volume is nearly constant and that the size of the visibility graph should depend primarily on the node spacing.

Figure 6 shows the size of the generated visibility graph for different values of the node spacing $l_{\max}$. The size initially increases rapidly when the node spacing is low and many nodes are interpolated inside the patches. As the vertex count increases fewer node are interpolated and affine growth becomes dominating. The offset to the lower limit is because of the capped area at the bottom, where no vertices are sampled and the nodes are always interpolated. The two plots show similar behavior and there is a nearly proportional relation between number of CS nodes and links, as also suggested by the minimum bounds for number of nodes and links. A quite high number of patches are generated even when for workspace obstacles with few vertices. These patches are build from the convex hull of each workspace obstacle. The number of patches N_p generated from a hull is the sum of number of its facets F, edges E, and vertices V and can be stated using Euler's formula as $N_p = 2E + 2$.

3.1.2 Path quality

A path found in the visibility graph will be an approximation to the optimal path, since the number of nodes sampled for the visibility graph is finite. However, a higher number of nodes and links in the visibility graph generally results in a path closer to optimal. The geodesic path is the part of the non-approximated optimal shortest path located on the surface of a closed CS obstacle. Besides being the shortest path, this optimal path is also intrinsically straight (that is, straight from the perspective of an individual moving on the surface) everywhere for a CS obstacle, which means that it is continuously differentiable. This also means that when the path crosses an edge between two patches, the incoming velocity vector match the outgoing velocity vector at the edge. The same is true for paths between obstacles, since they are tangents to both obstacles by definition. Obviously, this property is not preserved for approximated shortest paths generated by our approach.

Since 'intrinsic straightness' is a condition for the optimal path, this can be used in combination with length as a measure of 'quality' of a given path. It is likely that a path that is far from intrinsic straight is far from shortest. Of cause a path that is intrinsic straight is not necessarily the shortest (not even locally).

We measure intrinsic straightness as an angle error Ω_e at the nodes in the approximated path as shown in Figure 8. We define the angle error at a path node to be the angular difference between the supporting line

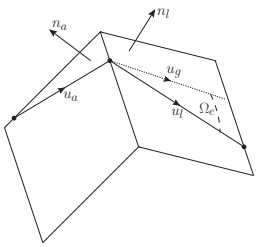


Fig. 8 The angle error Ω_e at a path node is the angular difference between the supporting line $\mathbf{u}_l$ pointing to the next node in the approximated path, and the supporting line of the geodesic path $\mathbf{u}_g$.

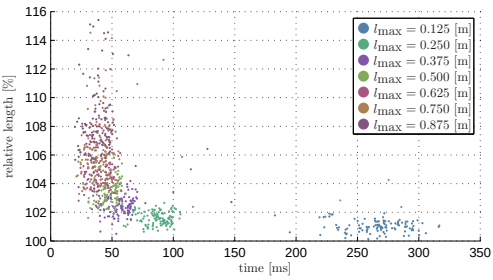


Fig. 9 The time spent on building the VG compared to the relative length of the resulting path. The plot is generated for a WS with up to 50 vertices, since this range has the largest potential for interpolated nodes, that is, highest CS node to WS node ratio in Figure 7.

u_l going to the next node in the approximated path, and the supporting line of the geodesic path u_g. Using the property of intrinsic straightness, supporting line $\mathbf{u}_g$ can be found by rotating the supporting line vector of the approximated path leading to the node $\mathbf{u}_a$ around the crossover edge $\mathbf{w}_e$ until this vector is aligned with the facet of the leaving path. It can be calculated using Rodrigues' rotation formula [27] as

$$\mathbf{w}_e = \frac{\mathbf{n}_a \times \mathbf{n}_l}{|\mathbf{n}_a \times \mathbf{n}_l|}$$
$$\lambda_e = \mathbf{n}_a \cdot \mathbf{n}_l$$
$$\mathbf{u}_g = \mathbf{u}_a\lambda_e + \mathbf{w}_e \cdot (\mathbf{w}_e \cdot \mathbf{u}_a)(1-\lambda_e) + \mathbf{w}_e \times \mathbf{u}_a\sqrt{1-\lambda_e^2}$$
$$\Omega_e = \arccos(\mathbf{u}_g\mathbf{u}_l)\ ,$$

where $\mathbf{n}_a$ is a unit normal from the facet of the arriving path, and $\mathbf{n}_l$ is a unit normal from the facet of the leaving path.

The approximated path is measured relative to the optimal path l_{geo} located on the surface of the patches. The upper bound on the length of the approximated path depends both on the node spacing $l_{\max}$ and its number of nodes N_p. The maximum length of the approximated path is $l_{\text{geo}}+l_{\max}(N_p-1)/2$, which is based on crossing a series of closely spaced edges, with maximum distance between nodes on consecutive edges. This suggests that lowering the node spacing $l_{\max}$ to obtain a larger visibility graph could have the adverse effect of increasing the number of nodes and hence the upper bound on length of the approximated path.

Figure 7 shows how various values of node spacing $l_{\max}$ affects the quality of the path. A lower $l_{\max}$ gives a higher density of nodes in the visibility graph and a path closer to optimal is found when the graph is searched. As $l_{\max}$ tend to zero, the node density tend towards infinity, and the approximated path tends towards the optimal path. The performance in terms of relative length versus computational time for different values of node spacing is shown in Figure 9. Obtaining better approximations comes at the price of computational time. While paths with length in the 100 to 102 range can be found at $l_{\max}$ is 0.125, it is often reasonable to accept lengths in the 104 to 106 range ($l_{\max}$ being 0.50 to 0.75), since such can be found significantly faster.

3.1.3 Replanning

Generating a VG for finding a path through an environment consist of first generating a per-obstacle graph, then combining these graphs while testing links and nodes for intersections, and finally searching the graph for the shortest path. The left plot in Figure 10 shows a summation of the time spent on each part for a different number of nodes. The graph search algorithm is shown for comparison. Once the visibility graph is generated, subsequent updates of it can be done faster depending on which changes occurred to the environment. The plots on the right show the time spent on updating the visibility graph in the case of a static and dynamic environment.

In the case of replanning in a static environment, a change to initial or final via point has been made, and a new path is found. This means that only affected via point nodes and their links are updated. Although intersection test must be performed on all affected links, these links are also candidates for pruning and replanning can be done fast, even though the computational cost per node is higher that in the case of a dynamic environment. Unfortunately, it is generally more expensive to process a link to a via point than a link between two nodes of the same obstacle.

Replanning in a dynamic environment happens if a new obstacle is introduced or an existing obstacle is transformed. It is cheaper to evaluate a candidate link between two nodes of an obstacle, especially when

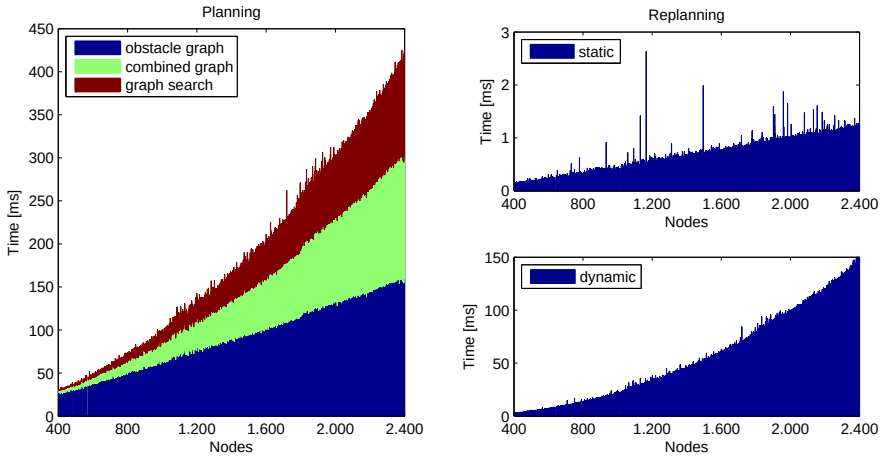


Fig. 10 The time required for planning and replanning as number of nodes increase. The visibility graph is created with a fixed $l_{\max}$ of 0.5. The left plot shows the initial time spend when finding a path. The plots on the right show the time used for subsequent replanning in a 1) static environment and 2) dynamic environment. Replanning in a static environment constitute a change to initial or final via point, while replanning in a dynamic environment occurs when an obstacle is rotated or translated. Time spent on graph search is not included in the plots on the right.

most nodes belong to that obstacle. This is because all candidate links between nodes in the same obstacle is determined when generating the obstacle visibility graph. Updating this step amounts to removing edges and nodes that intersect other obstacles or are outside some specified bounds. Links between nodes on different obstacles must still be processed, although many links are usually pruned.

3.2 Planning in a cluttered environment

The method is demonstrated on a UAV flying amidst eight obstacles. The UAV must visit five VPs and all five paths are initially planned. The initial setup and resulting paths can be seen in Figure 11(a). A close-up of some of the initial path can be seen in Figure 12. At the third VP, a new obstacle is found and replanning is required for the remaining two paths, see Figure 11(d). The full initial and updated path between all five VPs is found by solving six local path planning problems for connected VPs. The parameters are set relative to the size of the grid in the Figure. The UAV has radius r_{bs} of 1.70. The parameter $l_{\max}$ is set to 0.65.

The eight WS obstacles in Figure 11(a) consist of 60 vertices and 60 facets combined. Table 2 show these details for WS obstacle. It also shows the size of the obstacle VGs before and after pruning. After building the CS in Figure 11(b), the initial full visibility graph seen

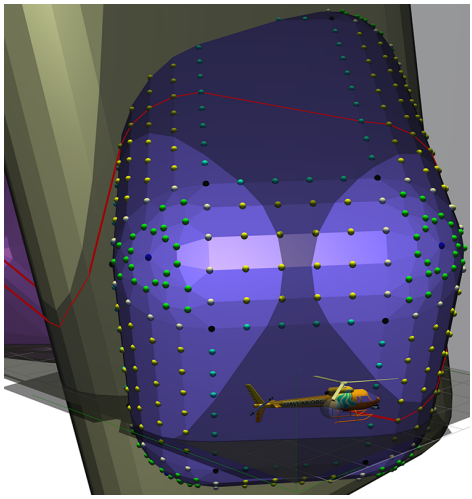
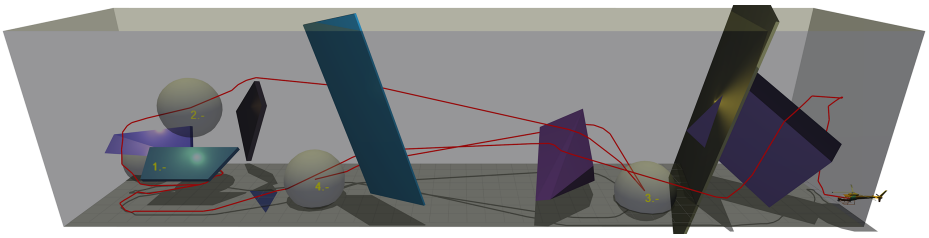
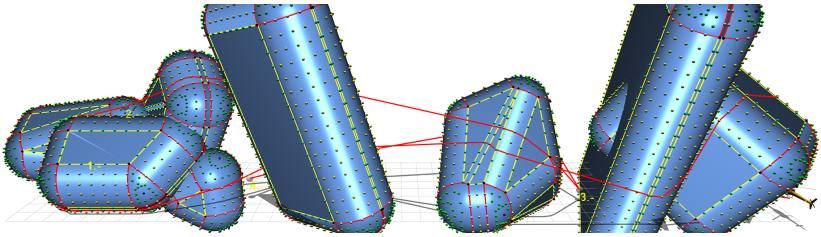


Fig. 12 Restrictions on the CS make the UAV take the longer path on the right. Note that a shorter path that moves up or left is not found because of the location of nodes.

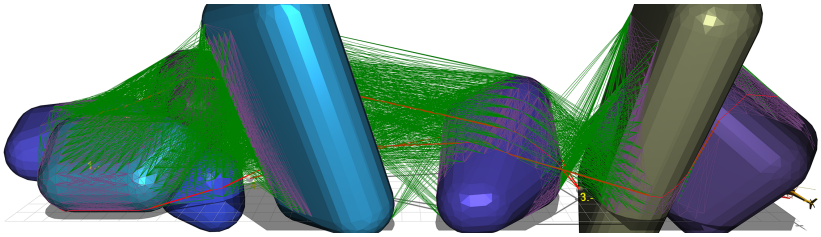
in Figure 11(c) has 1.9 thousand nodes and 16.6 thousand links. The updated graph in Figure 11(e) has 2.2 thousand nodes and 19.0 thousand links. In both cases the number is relative low since the majority of nodes and links stems from vertices and edges that intersect obstacles in CS, which are pruned before building full



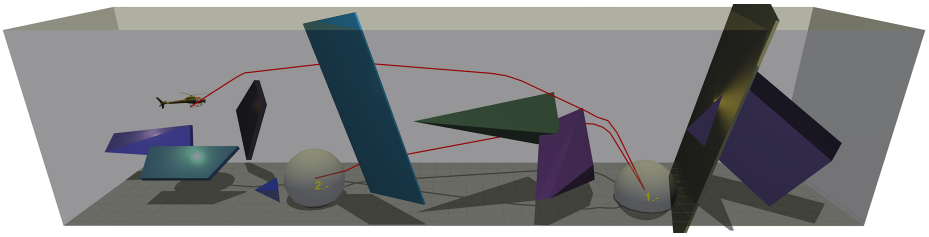
(a) A path is found between five VPs inside the semi-transparent gray box with eight obstacles



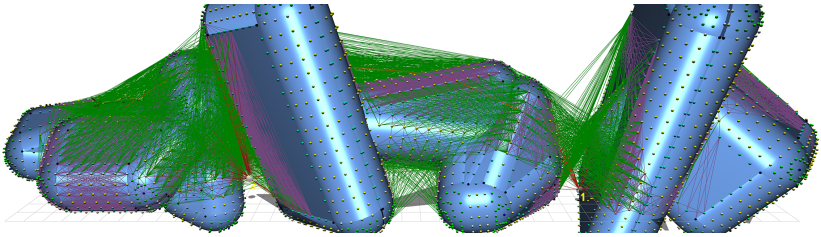
(b) The initial CS with VG node candidates. The eight CS obstacles have 224 patches.



(c) The mesh around each CS obstacle and the initial VG. No supporting graphs for via points are shown.



(d) A new obstacle is found.



(e) The CS and updated VG.

Fig. 11 Path planning in a cluttered environment. A different path is found after a new obstacle is introduced in (c).

Table 2 Size of each obstacle

Obstacle	1	2	3	4	5	6	7	8
WS Vertices	8	8	8	4	8	8	8	8
WS Facets	6	9	11	4	6	12	6	6
WS Edges	12	15	17	6	12	18	12	12
VG_o time [ms]	17	19	19	13	22	21	36	19
VG_o nodes	528	585	566	316	848	714	1136	776
VG_o links	3129	3574	3667	1234	10574	5565	21934	6978
$VG_{\bar{o}}$ nodes pruned [%]	45	52	68	70	65	36	90	71
$VG_{\bar{o}}$ links pruned [%]	53	66	78	79	61	42	96	81

Table 3 Time spent building and maintaining the VG at each path. The initial paths are annotated with 'a'. The updated paths are annotated with 'b'.

Path	a 1	a 2-4	b 3	b 4	All
VG_o time [ms]	517	0	137	0	654
VG_{os} time [ms]	573	0	268	0	841
VG_{ot} time [ms]	528	0	769	0	1297
VG_{vp} time [ms]	4	11	4	4	23
Total time [s]	1622	11	1177	4	2815

graph. The obstacle visibility graph VG_{os} that links nodes near each CS obstacle surface contains 76-80% of the links. The supporting line visibility graph VG_{ot} that links nodes of different CS obstacles contains another 20-23%, while the supporting line visibility graph VG_{vp} (that links via points to each other and to the nodes of the CS obstacles) only accounts for 0.6% of the links.

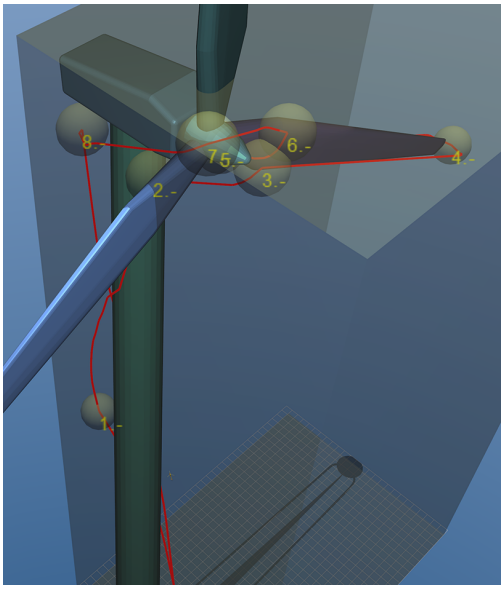
The full path was found in 3.1 seconds. Table 3 lists the time spent on building each graph for each path. While the initial VG for the first path is expensive to calculate, updating a single node is inexpensive. This means that the three subsequent paths are found in 11 milliseconds in total. Rebuilding the VG after a new obstacle is introduced is also expensive, but still less than for the initial path. This is because that VG_{os} was built incrementally per obstacle as detailed in Section 2.1.5. This means that updated VG_{os} can be calculated as the union of the old VG_{os} that is pruned in respect to the new obstacles, and $VG_{\bar{o}}$ for the new obstacles. Building VG_o takes 137 milliseconds, while updating VG_{ot} is more expensive since it is recalculated entirely and there are more nodes. A similar approach could be implemented for this graph, since the supporting lines between static obstacles would be retained unless intersected by a transformed obstacle. This could be done by updating the collision graph of VG_{ot} shown in (8). In addition to the times spent on the VGs, another 270 milliseconds were spent in total on searching the graphs for a path using an implementation of Dijkstra's search algorithm.

A limitation in the implemented method can be seen in Figure 12. In order to reach the points at the lower left of the yellow obstacle, the path starts by going left. This is because the VG lacks nodes at the cylinder patches directly above and to the right. If one less or one additional series of vertices had been interpolated at the cylinder patch above, a shorter path would likely have been found here. In fact, the actual shortest path (geodesic) follows the intersection ridge between the CS obstacle and bounding box. We suggest adding an 'intersection VG' in future works, where vertices for graph nodes are added at the intersection points between patches and bounding box. The intersection graph forms a mesh that closes the holes between intersecting obstacles.

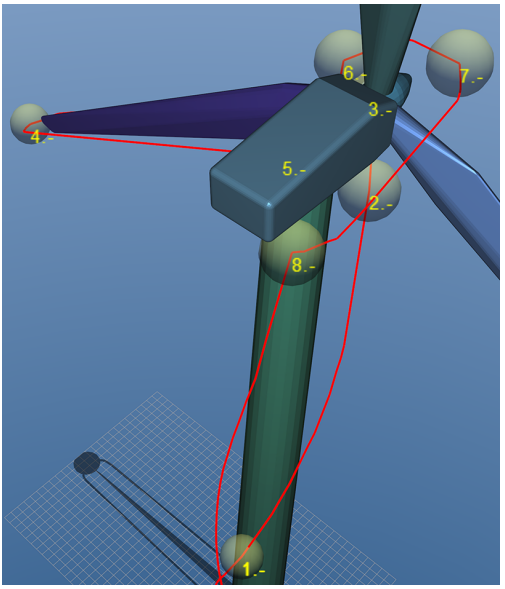
3.3 Planning near a large structure

The implemented method for constructing VGs is demonstrated on WS obstacles with a higher resolution. This test also demonstrates how only a part of the graph is rebuild after obstacles are transformed. In this scenario, the vehicle must visit a number of VPs located near the surface of a wind turbine (Vestas V52). The turbine model seen in Figure 13(a) and Figure 13(b) is composed of six separate obstacles as indicated by the different colors. These can be grouped into a stator assembly that consists of the tower and nacelle, and a rotor assembly that consists of three rotors attached to the nose cone. The vehicle starts near the base of the turbine and must visit the 8 numbered locations and then return to the initial position. The vehicle hovers at VP 5 (the start of path 6) where it is clear of the rotor blades while they are rotated 30 degrees. This rotation is clearly visible when comparing Figure 13(c) with Figure 13(d). The vehicle has radius r_{bs} of 1.70 meters. The $l_{\max}$ parameter is set to 0.65 meters.

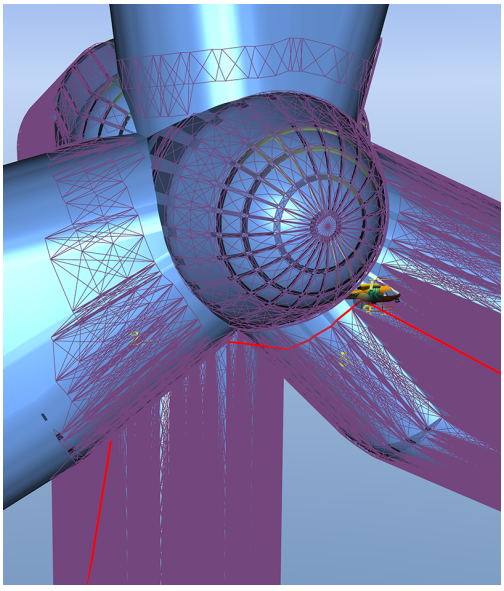
The problem was solved as nine local path planning problems in 193 seconds. This covers both the time spend on building and maintaining the VG, listed in Table 4, and time spent searching the graphs. The initial VG had 12.9 thousand nodes and 144 thousand links until the transformations at VP 5. The updated VG had 12.6 thousand nodes and 172 thousand links.



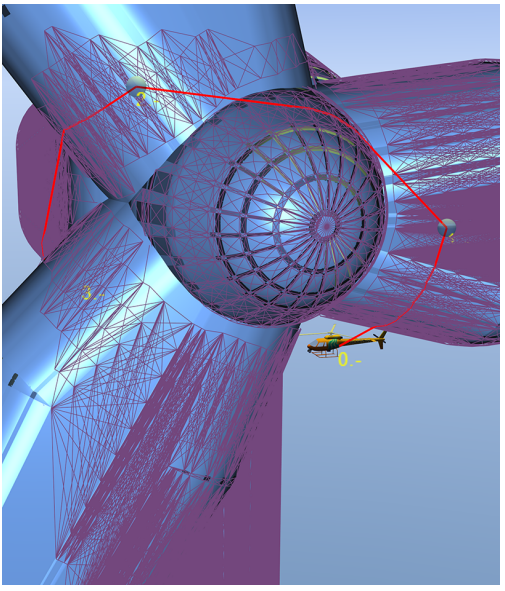
(a) Front view of wind turbine and VPs. Graph is build inside the semitransparent box.



(b) Back view of wind turbine and VPs.



(c) The vehicle at VP3. CS obstacles and VG_{os} graphs are shown. Note that individual obstacle graphs are not connected.



(d) The vehicle at VP5. WS obstacles and obstacle graphs VG_{os} are shown. The turbine blades are rotated 30 degrees.

Fig. 13 The vehicle starts near base of turbine, and must visit all VPs in order while staying inside the semitransparent box. After VP8 the vehicle returns to initial position

Table 4 Time spent building and maintaining the VG at each path

Path	1	2-5	6	7-9	All
VG_o time [s]	1.36	0	0	0	1.36
VG_{os} time [s]	57.5	0	44.2	0	102
VG_{ot} time [s]	23.7	0	24.2	0	47.9
VG_{vp} time [s]	0.09	0.35	0.03	0.10	0.57
Total time [s]	82.7	0.35	68.4	0.10	152

The generated VGs are searched with an implementation of Dijkstra's algorithm to obtain a path. This took approximately 4.6 seconds per path.

From Table 4, it can be seen that much time is spent on building VG_{os} and VG_{ot}. In both cases most is spent on collision tests. When building VG_{os} the vertices all nodes and edges of all links of each obstacle are tested for intersection with other obstacles. When building VG_{ot} link candidates between nodes of different obstacles are tested for intersection. See Section 2.2 and Section 2.3 for details.

It is about 24 seconds faster to rebuild the graph at VP 5 than at initial construction. This is mostly because VG_{os} does not have to be rebuild completely. Since only the four obstacles in the rotor assembly are transformed, the pruned graphs $VG_{\bar{o}}$ for the remaining two static obstacles are updated to reflect this. This is done by performing intersection tests against the transformed obstacles and adding the links that these had had previously intersected.

We believe it is possible to further reduce the time spent on rebuilding the graph when obstacles can be grouped into assemblies. There are two assemblies in the demonstrated case. In other cases all static obstacles could be grouped into an assembly. We suggest first building a graph for each assembly, and subsequently combining these graphs. The difference is to first build VG_{os} and VG_{ot} for each assembly, since such graphs are invariant under transformation of an assembly. In the second step each assembly graph is pruned to remove intersections, and the supporting graph between the assemblies is calculated. The advantage is that only the latter step must be recalculated after a transformation. This is left as an issue for future works.

The current implementation removes nodes of intersecting vertices between two intersecting CS obstacles. It is unfortunately that these are simply removed, since it appears that the actual shortest paths tend to follow these ridges of intersection. An example of this is visible in Figure 13(c). The vehicle at VP 3 arrived from VP 2 behind the leftmost rotor. The actual shortest path is through the intersection point between the two downwards pointing rotors and nose cone. It then follows the intersection ridge between the nose cone and rightmost rotor, before leaving the surface to VP 3. The situation is similar to that in Figure 12, see Section 3.2, and again we suggest building intersection graphs between intersecting CS obstacles for future works.

4 Conclusions

We considered the problem of multi-query path planning for autonomous small- and micro-scale UAVs. We formulate the problem as the shortest path problem for a sphere that must cover both bounding radius and all errors on position of the vehicle. The shortest path for a sphere amidst polyhedral obstacles is a combination of paths on the surface of the obstacles and supporting lines. Finding such a shortest path in 3D is NP-complete, since it does not in general traverse only vertices of the polyhedron, as in the 2D case, but also points on its edges. If one suffices with an approximately shortest path, a finite set of points can be selected near extremities of the configuration space. From these points a visibility graph can be build that can be searched to give an approximated shortest path.

We presented a method for building and maintaining a 3D visibility graph from a known space with obstacles. We focus on the ability to maintain this graph under changes, such as new or rotated obstacles, without the need to recalculate the entire VG. This led to a stepwise approach that composes the VG of obstacle and supporting lines visibility graphs.

We found that the visibility graph can generally be used to find a path that is near-optimal in terms of length and angle (it is nearly intrinsically straight). The near-optimality depends on a $l_{\max}$ parameter. The path tends to optimal as this parameter is reduced but also rapidly increasing demands to computational time, and setting the parameter too low is not recommended. A high number of CS nodes per WS node are generally produced even for a moderate choice of $l_{\max}$. This means that a very high number of obstacles or too detailed obstacles are not recommended, since computational load increases rapidly with the number of convex vertices. The use of a polygon reduction algorithm and a high value for $l_{\max}$ can reduce the problem somewhat.

We give recommendations for improvements in both accuracy and performance for future works. Two examples demonstrate planning in a cluttered environment and near a large structure. We show how a potentially better path can be missed in cases where CS obstacles intersect each other or the bounds. This can happen because shortest paths tend to follow the surface of the CS. Accuracy could possibly be improved in future works by introducing intersection graphs that ensure

surface coverage of intersecting obstacles and where obstacles intersect the bounds. The examples also show how replanning and planning between several VPs is significantly faster because we keeping track of changes in the environment and subsequently update the graph accordingly. We suggest building supporting graphs between pairs of obstacles and grouping obstacles that form rigid obstacles to assemblies to further reduce the computation load when replanning. ´

References

1. D. H. Shim. *Autonomous exploration in unknown urban environments for unmanned aerial vehicles.* University of California, Berkeley, 2005.
2. Office of the Secretary of Defence. *Unmanned aircraft systems roadmap 2005-2030.* Washington, DC, 2005.
3. M. Pachter and P. R. Chandler. *Challenges of Autonomous Control.* IEEE Control Systems Magazine, vol. 18, no. 4, 1998.
4. O. Chum. *Two-View Geometry Estimation by Random Sample and Consensus.* PhD Thesis, 2005.
5. Z. Marton, D. Pangercic, and N. Blodow. *General 3D Modelling of Novel Objects from a Single View.* IEEE International Conference on Intelligent Robots and Systems, 2010.
6. E. W. Dijkstra. *A note on two problems in connection with graphs.* Numerische Mathematik 1: 269-271, 1959.
7. P. E. Hart, N. J. Nilsson, and B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics SSC4: 100-107, 1968.
8. N. J. Nilsson. *A mobile automaton: An application of artificial intelligence techniques.* In 1st International Conference on Artificial Intelligence, pages 509-520, 1969.
9. J.-C. Latombe. *Robot motion planning.* Springer, 1991.
10. T. Lozano-Perez and M. A. Wesley. *An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles.* Communications of the ACM, 22, 1979.
11. T. Lozano-Perez. *Spatial Planning: A Configuration Space Approach.* IEEE Transactions on Computers, 32, 1983.
12. E. Oks and M. Sharir. *Minkowski Sums of Monotone and General Simple Polygons.* Discrete and Computational Geometry, 2006.
13. B. Aronov and M. Sharir. *On translational motion planning in 3-space.* In ACM Symposium on Computational Geometry, pages 21–30, 1994.
14. J. Canny and J. H. Reif. *New lower bound techniques for robot motion planning problems.* In Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci., 1987.
15. J. T. Schwartz and M. Sharir. *On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds.* Advances in Applied Mathematics, 4:298–351, 1983.
16. J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, 1988.
17. C. Bajaj and M. Kim. *Generation of configuration space obstacles 1: the case of a moving sphere.* IEEE Journal of Robotics and Automation, 1988.
18. S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry.* Springer-Verlag, Berlin, 2003.
19. S. R. Lindemann and S. M. LaValle. *Current issues in sampling-based motion planning.* In P. Dario and R. Chatila, editors, Proceedings International Symposium on Robotics Research. Springer-Verlag, Berlin, 2004.
20. R. C. Brost. *Computing metric and topological properties of configuration obstacle.* in Proc. IEEE Conf. Robotics and Automation, pp.170-176, 1989.
21. L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* IEEE Transactions on Robotics & Automation 12 (4) 566–580, 1996.
22. W. Shu and Z. Zheng. *Computing configuration space obstacles using polynomial transform.* Proceedings of International Conference of Robotics and Automation Vol.4, 2004.
23. J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1998.
24. C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. *The quickhull algorithm for convex hulls.* ACM Trans. Mathematical Software 22, 1996.
25. S. R. Buss and J. Fillmore. *Spherical Averages and Applications to Spherical Splines and Interpolation.* ACM Transactions on Graphics 20, 2001.
26. T. Moller and B. Trumbore. *Fast, Minimum Storage Ray-Triangle Intersection.* J. Graphics Tools 2(1), 1997.
27. D. Koks. *Explorations in Mathematical Physics.* Springer Science+Business Media, Ch.4. A Roundabout Route to Geometric Algebra, 2006.

Paper D

3D Path Planning with Geodesics on Geometric Work Spaces

Flemming Schøler and Anders la Cour-Harbo

This paper was submitted to:
Journal on Robotics and Autonomous Systems, 2012

Available online at www.sciencedirect.com
ScienceDirect
Robots and Autonomous Systems 00 (2012) 1–19
Autonomous Systems

3D Path planning with geodesics on geometric work spaces

Flemming Schøler, Anders la Cour-Harbo

Aalborg University, 9220 Aalborg East, Denmark

Abstract

Path planning problems are difficult in 3D. We address the problem of identifying a shortest path for a sphere moving around a single polyhedron in a 3D work space (WS). Our method calculates the configuration space (CS) obstacle as a collection of patches. Using properties of geodesics to compose a path on these patches, the 3D path planning problem can be reduced to a one dimensional strictly quasiconvex optimization problem. The solution space can then be sampled to identify the convex region containing the optimal solution. The problem converges to a solution very quickly when numerical optimization is applied. Results show that the approach is very fast for a single obstacle. It is expected that the method scales very well to environments with additional obstacles, and that it is applicable to polyhedral CS obstacles.

Keywords: path planning, helicopter, geodesics

1. Introduction

We present a method for finding the shortest path for a spherical vehicle around a convex polyhedron in 3D. The method provides a position path in x, y, z coordinates, and does not take time or vehicle dynamic into account. However, under certain conditions the method guarantees that the path found is indeed the shortest possible in Euclidean distance. Usually, the model of a rigid vehicle operating in 3D has six degrees-of-freedom, with three dimensions to describe position and three to describe orientation. In this research we disregard orientation by planning a path for the bounding sphere of the vehicle. That is, we plan a path in 3D that will not intersect an obstacle when traveled by a sphere. This is done by describing the obstacles in a configuration space (CS) that in turn is based on the original obstacles in work space (WS). The aim of this is to transform the more difficult problem of finding a path for a vehicle around the WS obstacle into the easier dual problem of finding a path for a point around the CS obstacle. In CS the obstacle is generated by 'inflating' the WS obstacle according to sphere radius. The WS obstacle is build from the convex hull of any set of vertices such as a point-cloud from sampled data. The presented method is limited to finding solutions on convex obstacles.

1.1. Previous work

The notion of Euclidean shortest path for vehicle translating in a 2D or 3D space with multiple convex polytopes is straightforward. Exact algorithms exist in the 2D case with methods such as such as Visibility Graphs for multi-query searches, see [1, 2] and i.e. [3, 4] for improved algorithms, and continuous Dijkstra algorithms, see [5, 6] for single query searches. Such 2D methods are described more generally in [3, 7, 8]. While it is clear that the shortest path is on the surface of the polytope in both 2D and 3D, and that this means that the problem is fairly straight forward in 2D,

Email addresses: fls@es.aau.dk (Flemming Schøler), alc@es.aau.dk (Anders la Cour-Harbo)

these methods unfortunately do not extend nicely to 3D. The main problem in 3D is that the shortest path does not in general traverse only vertices of the convex polytope, as in the 2D case, but also points on the edges of the polytope. It is not generally clear where on the edges the path should cross in order to achieve minimal length. In fact, since it was shown by [9] that the 3D shortest-path problem in a multi-polyhedral environment is NP-hard, the only practical approach is to use an approximative method, such as [10, 11].

A special variant is the problem of computing a shortest path between two points along the surface of a single convex polytope. Approximation algorithms exist for this problem, see [12, 13, 14], and an exact method was presented by Sharir and Schorr [15] that exploits the property that a shortest path on a polyhedron unfolds into a straight line. Another approach was described by Chen and Yijie [16] with an implementation made public available by Kaneva and O'Rourke [17]. This non-Dijkstra based algorithm finds the shortest path from one source point to all vertices on a polyhedral surface. Another approach is the fast marching method, which is an optimal time numerical method for solving the Eikonal equation on a Cartesian grid [18, 19]. The central idea is to advance a wavefront starting at an initial grid point though neighboring grid points continuously unit it first reaches the final grid point. It is related to Dijkstra's algorithm [20] used in finding shortest paths between nodes linked in a network. However, the fast marching method is effective for surfaces because it is not limited to path planning along links. Kimmel and Sethian [21] extended the method from points in Cartesian grids to points on triangulated manifolds.

1.2. Present work

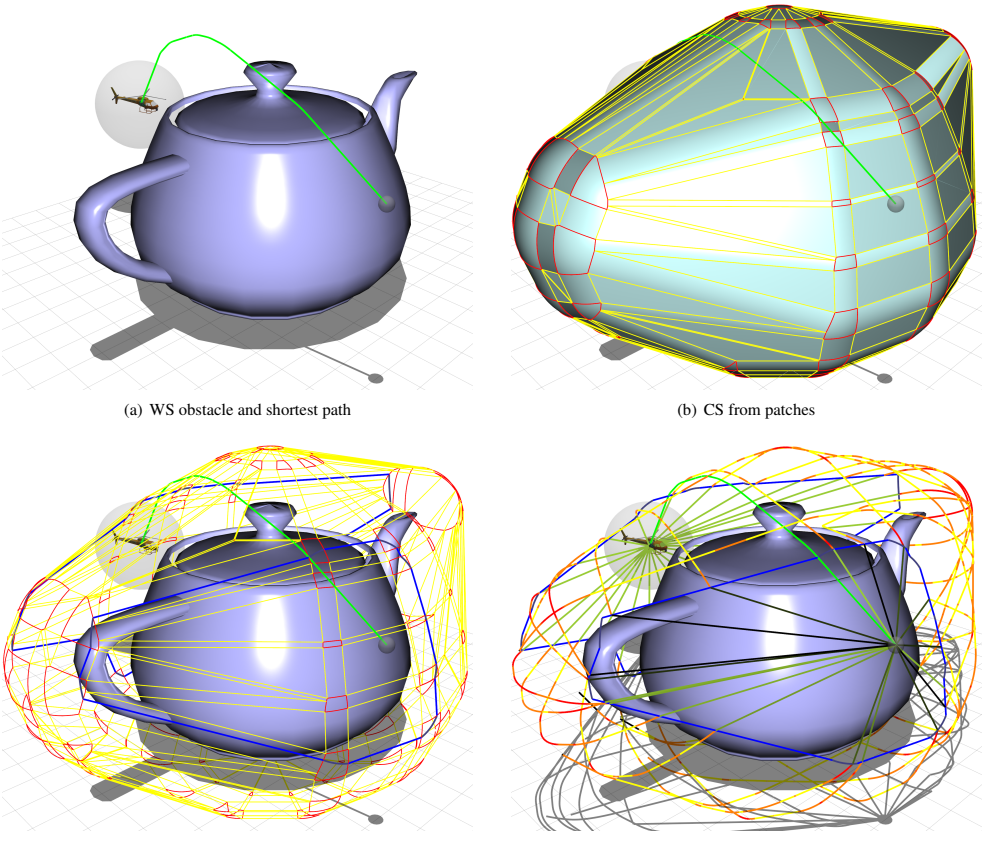
The methods described above operate on polytope obstacles. In our case the CS obstacle has a continuously differentiable surface since it is generated as the Minkowski sum of a convex WS obstacle and a sphere. We also seek a method that is not limited to path planning between points on the obstacle surface. Hence we suggest a different approach based on the wavefront idea. To do so, an extension is needed that allows finding paths between two points A and B outside the obstacles. For each A and B a set of points are identified on the obstacle, which will be intersected by the shortest path between them. The problem is now to find a path between these two sets. That is, instead of having a single source and target point, the concept of wavefront planning is applied on point sets. Naturally, the shortest path between these point sets does not necessarily represent a subset of the globally shortest. However certain end point conditions can be introduced on the path subset that ensures that the full path is locally shortest. The globally shortest path is then given from the set of these locally shortest paths.

1.2.1. Conditions

We note that when computing the shortest path between two points A and B in a Euclidean space containing convex obstacles, the shortest path is a series of connected edges. Each edge is either on the hull of the CS obstacle or it is a line segment that extends between a point and an obstacle or between obstacles. Such a line segment is called a supporting line, and it is tangent to the obstacle(s). The set of points where the supporting lines of a point A meets the obstacle surface is denoted the 'horizon' of A. This means that the shortest path from point A to point B around some obstacle will follow a supporting line to the horizon of A, the continue by following the obstacle surface to the horizon of B, and then follow a supporting line to B. The part of the path on the obstacle is a geodesic. A geodesic is a locally length-minimizing curve on a surface, and geodesics preserve their direction on this surface [22]. A locally shortest path also preserves direction at the horizon. This has the effect that any candidates for a shortest path are uniquely determined by their initial direction, and it is a locally shortest path if it intersects the target position. The direction preserving property is not limited to the geodesic but apply to the full path. The shortest path will be located on the surface formed by the CS obstacle and all supporting lines through A to B, that is, the convex hull of A, B and the CS obstacle. In this dual problem the shortest path is a geodesic since it is on a surface, and hence preserves direction. While we can consider the full path to be a geodesic, we use this term only for the path segment on the CS obstacle. We use the terms supporting line or tangent to describe the path segment connecting the obstacle to A and B.

2. Method

Figure 1 shows the basic steps for finding the shortest path around an obstacle. We start by constructing the CS obstacle in Figure 1(b) from the convex hull of the work space (WS) obstacle in Figure 1(a). The actual construction of this CS obstacle is discussed in Section 2.1. Due to the way it is constructed, the CS obstacle will be continuously



(a) WS obstacle and shortest path

(b) CS from patches

(c) Horizons from VPs

(d) Traces

Figure 1. Determining the shortest path.

differentiable and consist of patches of spheres, cylinders, and facets. Those patches are outlined in yellow and red in Figure 1(b).

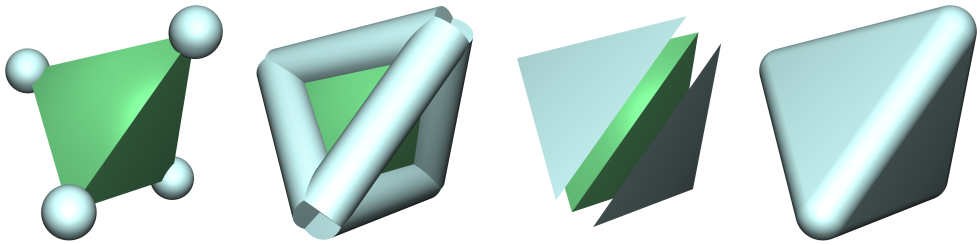
The shortest path between two points in CS consists of three parts; 1) a supporting line from the initial point to the horizon of the initial point on obstacle surface, 2) a number of geodesics on the surface of the CS obstacle, and 3) another supporting line from the horizon of the target point to the target point. In Section 2.2 it is shown how to find the horizon of a point. A horizon is a closed path composed of small-circle segments on sphere patches, and line segments on the cylinder patches. Figure 1(c) shows the two horizons in blue, and the green path is composed of a supporting line from the helicopter (initial point) to the blue horizon on the far side of the CS obstacle, then a geodesic on the surface of the CS obstacle, and finally a supporting line from the horizon on the near side of the CS obstacle to the target point (the grey sphere). Note that two horizons intersect if and only if the shortest path is trivial, that is the initial and target points are in line-of-sight.

Because the surface is continuously differentiable the supporting lines are tangents to the surface, and therefore the supporting line fully describes the initial state (point and direction) for the part of path that is on the surface of the CS obstacle. Thus, we know the initial state for the first of the geodesics of the path. In Section 2.3 we define the

parametric equations used to describe the geodesic for each type of patch, and the initial state is enough to uniquely define the segment of the geodesic that is on the patch. A parametric equation for a particular type of patch is only valid inside the patch, and therefore we must be able to determine when the geodesic intersects the edges of a patch to proceed to a neighboring patch. The final state of one patch is used as initial state for the geodesic on the next patch. To initially find a good approximation to the shortest path, we trace in a number of different directions from the initial point, and compare the length to the target point for each direction. Figure 1(d) shows such traces. If a trace intersects the target horizon before becoming too long, it is near a shortest path. In such case convex optimization (such as steepest descend) can be used to find the locally shortest path. This is discussed in Section 2.4.

2.1. Configuration Space

The space of all allowed states of the vehicle is denoted the configuration space or CS. In this work vehicle states are spatial position, and thus the CS is three dimensional and is constrained by a CS obstacle that originates in a WS obstacle. The CS obstacle is the Minkowski sum of the vehicle bounding sphere and the convex hull of the WS obstacle polyhedron. Conceptually, this Minkowski sum can be obtained by replacing vertices of the WS convex hull with spheres, replacing edges with cylinders, and replacing facets with translated facets (translated along their normal). The radius of the spheres and cylinders, and amount of translation is equal to the vehicle bounding sphere radius. The resulting surface has G1 continuity and an example of a CS obstacle constructed in this way from a WS obstacle is shown in Figure 2.



(a) Spheres at vertices (b) Cylinders at edges (c) Facets are translated (d) Complete CS obstacle

Figure 2. A CS obstacle is effectively constructed from patches of spheres, cylinders, and facets.

More accurately, the surface of a CS obstacle is composed of patches of the primitive surfaces spheres, cylinders, and planes. The patches are the 'visible' parts of these primitives. Each of these patches is a simple, convex, closed polygon on a primitive surface. The edges of the polygon are formed by geodesic segments between N counter-clockwise ordered vertices v_0 to v_N. The last vertex v_N is assumed to be the same as the first, i.e. the polygon is closed. Any two neighboring patches of the CS obstacle intersect at one common polygon edge. Since each patch P is convex the geodesic between any two vertices in P lies entirely in P. For curved patches, the angular span of any geodesic in P is less than π. Each vertex of a patch is shared by two cylinder patches, one sphere patch and one planar patch. It follows that spherical patches only have cylindrical patches as neighbors. Cylindrical patches have two opposing sphere patches and two opposing facet patches as neighbors. Planar patches (also called facets) have only neighboring cylindrical patches. All sphere patches combined comprise exactly one full sphere. Each vertex in a sphere patch polygon is obtained by translating the same WS vertex along the neighboring facet normals, so the sphere patch has the same number of vertices as number of neighboring WS facets. The four vertices in the cylinder patch are obtained by translating the endpoints of a WS edge along the two neighboring facet normals. The vertices in the facet are obtained by translating the WS facet along its normal.

2.2. Horizon

The shortest path from a point in CS (which we will call a view point) around a CS obstacle necessarily 'arrives' at the obstacle in a point where the (supporting) line from the view point to the obstacle is tangent to the obstacle.

The entire set of points on the CS obstacle where this is the case is called the horizon, and it is dependent on the view point location and shape of the obstacle.

Definition 1 (View point)**.** *A view point is a point*

$$\mathbf{w} = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} .$$

in configuration space such that $\mathbf{w}$ *is not inside any of the CS obstacles. It may be on the surface of a CS obstacle.*

Definition 2 (Horizon)**.** *For a differentiable manifold* S *and a view point* $\mathbf{w}$ *we define the horizon on the manifold as the set*

$$\{\mathbf{x} \in S \mid \mathbf{w} \in T_x S\} ,$$

where T_x *is the tangent space to* $\mathbf{x}$ *for* S*. That is, the set of points* $\mathbf{x}$ *on* S *whose tangent space intersect* $\mathbf{w}$*.*

Definition 3 (Supporting line)**.** *A line segment extending from a view point* $\mathbf{w}$ *to a point on a horizon (on a CS obstacle) associated with* $\mathbf{w}$ *is called a supporting line.*

The surface of an obstacle is composed of patches of primitives, but only primitives with a curved surface can have a horizon. These are the sphere and cylinder primitives. There are no horizons on a facet primitive. Although, it is possible to find an intersection tangent space if the view point is in the same plane as the facet, such cases will also intersect a cylinder patch. Two horizons belonging to two different view points intersect if and only if the view points have mutual line-of-sight.

Definition 4 (Horizon map)**.** *The closed horizon path* $\mathbf{h}(s) : [0; 1) \mapsto \mathbb{R}^3$ *is defined as the map from the normalized horizon parameter* s *to points on the horizon, and it is composed of* N *local horizon segments called* $\mathbf{h}_i(s)$*, where* $i = 0, \ldots, N-1$*.*

These segments are alternating sphere patch segments $\mathbf{h_s}(s)$ and cylinder patch segments $\mathbf{h_c}(s)$. That is, the parametric equation for the closed horizon path is

$$\mathbf{h}_i(s) = \begin{cases} \mathbf{M}_i \mathbf{h}_{\mathbf{s}i}(sS_N - S_{i-1}) + \mathbf{O}_i & (i \text{ even}) \\ \mathbf{M}_i \mathbf{h}_{\mathbf{c}i}(sS_N - S_{i-1}) + \mathbf{O}_i & (i \text{ odd}) , \end{cases} \tag{1}$$

where S_i denotes the combined length of segments 0 through i. Then $\mathbf{h}(s)$ is equal to $\mathbf{h}_i(s)$ whenever index i fulfills $S_{i-1} \leq sS_N < S_i$. Since $\mathbf{h}$ is defined in coordinates local to each primitive surface (in the following subsections) an orthonormal rotation matrix $\mathbf{M}_i$ and a translation vector $\mathbf{O}_i$ transforms the local horizon segment to the world frame. The formula for S_i is

$$S_i = \sum_{j=0}^{i} \int_{s_{j,\text{start}}}^{s_{j,\text{end}}} \|\mathbf{h}'_j(s)\| ds$$

where the prime on $\mathbf{h}$ means derivative with respect to s. Note that the integrals can be computed quite easily since each $\mathbf{h_s}$ is simply a part of a circle and each $\mathbf{h_c}$ is straight line.

The parametric equations horizon segments for cylinder patches $\mathbf{h_c}(s)$ and sphere patches $\mathbf{h_s}(s)$ are treated in the following two sections.

2.2.1. Horizon on cylinder

On a cylinder primitive the horizon is composed of two straight line-segments on the surface of the cylinder primitive along its height. These line-segments are calculated relative to a right hand reference frame. This frame is centered at the base of the cylinder with the z axis coinciding with the center axis of the cylinder. Its y axis is

perpendicular to the plane containing the z axis and the view point. The parametric equation for the two horizon segments $\mathbf{h_c}(s)$ is given as

$$\Theta = \arctan \frac{\sqrt{d_w - r^2}}{r} \tag{2}$$

$$\mathbf{h_c}(s_c) = \begin{pmatrix} r\cos(\pm\Theta) \\ r\sin(\pm\Theta) \\ s_c \end{pmatrix}, \tag{3}$$

for $s_c \in [0, d_h)$, where r is the vehicle radius, d_h the height of the cylinder along its axis, and d_w is the shortest distance between z axis and the view point. None, either, or both horizon segments may be inside the patch. If either value of $\pm\Theta$ is inside the limitations of the patch, it represents a horizon that continues on both neighboring sphere patches.

2.2.2. Horizon on sphere

On a sphere primitive, the horizon is a small circle. Its radius is determined by distance to view point. This circle is calculated relative to a reference frame centered at the center of the sphere $\mathbf{p}_s$ with the z axis pointing towards the view point $\mathbf{w}$. The parametric equation for a horizon circle $\mathbf{h_s}(s)$ on the primitive is given as

$$d_r = \frac{r}{|\mathbf{p}_s - \mathbf{w}|} \tag{4}$$

$$\mathbf{h_s}(s_s) = r \begin{pmatrix} \cos s_s \sqrt{1 - d_r^2} \\ \sin s_s \sqrt{1 - d_r^2} \\ d_r \end{pmatrix}, \tag{5}$$

for $s_s \in [0; 2\pi)$. Only those parts of the horizon circle that are inside the sphere patch are part of the CS obstacle horizon. This means that a sphere patch will have at most N_c horizon arcs, with N_c being the number of neighboring cylinder patches. Each arc is then defined by non-overlapping pairs of $\{s_{\text{start}}, s_{\text{end}}\} \in [0; 2\pi)$. These are identified by transforming cylinder horizon endpoints $\{\mathbf{h_c}(s_{\text{start}}), \mathbf{h_c}(s_{\text{end}})\}$ to the sphere horizon frame.

2.3. Geodesics

A geodesic is a locally length-minimizing curve that preserves a direction on a surface. It can be uniquely defined by a position and direction at some point on it. In the following work, we will need to distinguish between a geodesic local to a patch (we call this a *geodesic*) and a geodesic spanning multiple patches (we call this a *geodesic path*). Our geodesic paths are defined by the view point and horizon point. The vector from a view point to a point on its horizon, given by the horizon parameter s, gives the starting point and direction of the geodesic path. Since the obstacle surface is composed of patches, this geodesic path becomes a piecewise function where the equation and domain for each piece (geodesic) must be known.

Definition 5 (Geodesic path)**.** *A geodesic path is a parametric function* $\mathbf{g}(t) : \mathbb{R} \mapsto \mathbb{R}^3$ *that is a piecewise function composed of N successive geodesics* $\mathbf{g}_i(t)$ *indexed by* $i = 0, \ldots, N$.

A geodesic path is uniquely given by a point and a direction on surface of a CS obstacle, or, equivalently, by a vector tangent to the CS obstacle. Such a vector is called a geodesic path tangent. Using this concept we can now also define all possible geodesics associated with a view point.

Definition 6 (Geodesic set)**.** *For a CS obstacle and a view point* $\mathbf{w}$ *let* $\mathbf{h_w}(s)$ *be the corresponding horizon. Then the geodesic set for* $\mathbf{w}$ *is the set of geodesic paths given by the set of supporting of lines for* $\mathbf{h_w}(s)$*. If* $\mathbf{w}$ *is on the surface the geodesic set is the set of all geodesic paths starting at* $\mathbf{w}$.

With the horizon and geodesic paths defined we can now also give a formal definition of the trace.

Definition 7 (Trace)**.** *For a given CS obstacle and initial and target view points,* $\mathbf{w}_{\text{init}}$ *and* $\mathbf{w}_{\text{tar}}$*, a trace is the piecewise, connected curve consisting of*

1. *the supporting line from* $\mathbf{w}_{\text{init}}$ *to a point* $\mathbf{h}_{\text{init}}(s)$ *on the initial horizon,*
2. *the geodesic path* $\mathbf{g}(t)$ *defined by* $\mathbf{h}_{\text{init}}(s)$ *and the direction given by the supporting line,*
3. *the supporting line from the intersection between* $\mathbf{g}(t)$ *and the target horizon* $\mathbf{h}_{\text{tar}}$ *to* $\mathbf{w}_{\text{tar}}$.

A trace is continuously differentiable except possibly at the intersection point on the target horizon. Any trace that represent a locally shortest path will be continuously differentiable everywhere. A geodesic path continues through a number of patches, and might or might not reach the target horizon. Its length is given by a geodesic parameter, and since we are specifically interested in length in this work, the geodesic parameter is an important parameter (it will be used to compute the length of the geodesic path and subsequently in practice for terminating unsuccessful traces).

Definition 8 (Geodesic parameter)**.** *The parameter* $t_i \in (0, t_{i,end}]$ *is defined for each* $\mathbf{g}_i(t)$ *such that distance traveled* T_i *by* $\mathbf{g}(t)$ *after i segments is*

$$T_i = \sum_{j=0}^{i} \int_0^{t_{j,end}} \|\mathbf{g}_j'(t)\| dt \ . \tag{6}$$

Consequently, the point reached after traveling a distance t is

$$\mathbf{g}(t) = \mathbf{M}_i \mathbf{g}_i(t - T_{i-1}) + \mathbf{O}_i \ , \tag{7}$$

where i is the index for which $T_{i-1} \leq t < T_i$. The parametric equations in $\mathbf{g}_i(t)$ each describe the path on a sphere, facet, or cylinder patch. They are calculated for a unit bounding vehicle radius, and can be scaled accordingly. The rotation matrix $\mathbf{M}_i$ and point $\mathbf{O}_i$ of each patch, transforms the geodesic path to the world frame. Below we describe how these equations are identified in their respective patch frames and propose methods for finding the length t_{end} of each geodesic.

Since a geodesic preserves direction on the surface and our surfaces are differentiable, both $\mathbf{g}_{i-1}(t_{i-1,\text{end}}) = \mathbf{g}_i(0)$ and $\mathbf{g}'_{i-1}(t_{i-1,\text{end}}) = \mathbf{g}'_i(0)$ are fulfilled at the transition point between patches. This condition is used for determining the parameters of the individual parametric equations in $\mathbf{g}(t)$, and it is done by finding the entry point $\mathbf{e_p}$ and entry direction $\mathbf{e_d}$

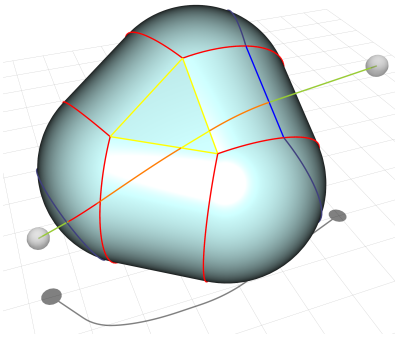
$$\mathbf{e_p} = \begin{bmatrix} e_{p,x} \\ e_{p,y} \\ e_{p,z} \end{bmatrix} , \qquad \mathbf{e_d} = \begin{bmatrix} e_{d,x} \\ e_{d,y} \\ e_{d,z} \end{bmatrix}$$

local to a given patch based on the geodesic of the previous patch. Figure 3 shows an example for a WS tetrahedron. Both the two horizons (shown in blue) and the shortest path (shown in green/red/orange) are shown in all 6 sub-figures. This geodesic path consists of four geodesic segments on the CS obstacle and two supporting lines. Figure 3(a) shows the CS obstacle drawn with all the patches. The primitive belonging to each of the four segment of the geodesic is shown in Figure 3(b) through Figure 3(e). The facets of the primitives are outlined by edges between numbered vertices. The figures also show the local frame of each path segment, and the WS obstacle where possible. Figure 3(f) shows paths for six different traces.

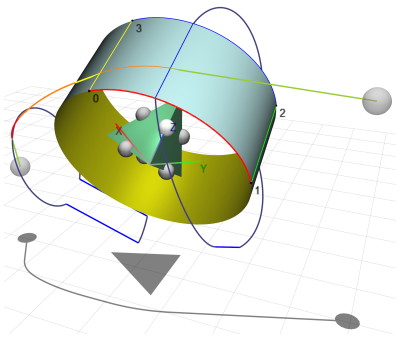
Because all patches are convex, the geodesic path between any two points inside a patch stays inside the patch. Any geodesic of a sphere patch is a great arc. For a cylinder patch it is a helix, and for a facet it is a line segment. This makes it relatively easy to find the shortest path between two points on the same patch. For points on neighboring patches we must cross the edge that separates these patches. Edges are line or arc segments located between two different types of patches. Line segments are located between facet and cylinder patches, and arc segments are located between cylinder and sphere patches. The endpoints of these segments represent the meeting point of two cylinder patches, a facet patch, and a sphere patch.

2.3.1. Geodesic on Cylinder

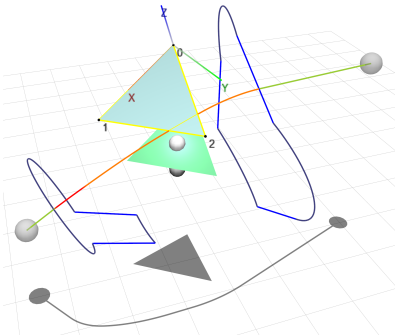
When 'unrolling' a cylinder to a planar surface the geodesic between two points becomes a straight line. When rolling the surface back up this geodesic becomes a helix. Figure 3(b) and 3(d) shows examples of these helices. The parametric equation for a helix starting at point $\mathbf{e_p}$ and with unit direction vector $\mathbf{e_d}$ also depends on the cylinder axis. The cylinder patch has four vertices enumerated in counter clockwise order when viewed from outside the cylinder.



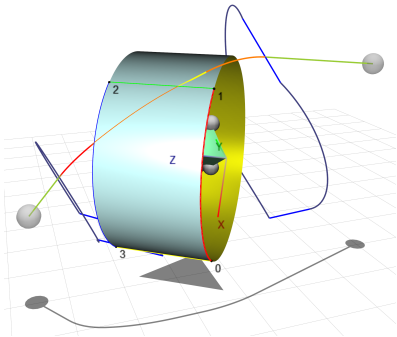
(a) The shortest trace starts at the small grey sphere to the right (the initial view point) and reaches the surface at the horizon shown in blue. The four patches for the geodesic path are shown in Fig (b), (c), (d), and (e).



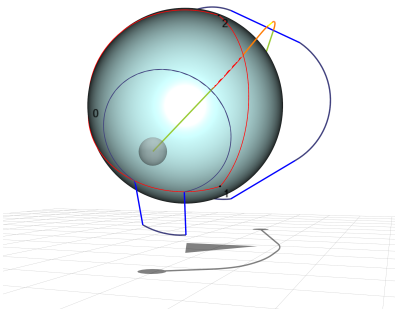
(b) The cylinder primitive that contains the first part of the geodesic (orange). The patch is outlined by the four numbered black vertices. The local reference frame of the patch is also shown.



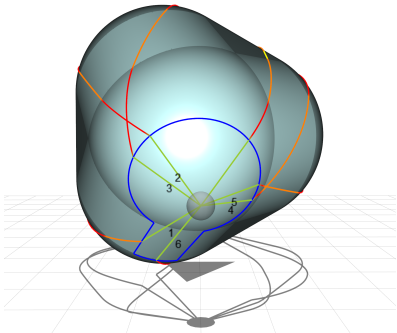
(c) The facet patch containing the second part of the geodesic path (yellow). The patch is outlined by the three numbered black vertices.



(d) The cylinder patch containing the third part of the geodesic path (orange).



(e) The sphere patch containing the fourth part of the geodesic path (red). Note that this figure has a somewhat different view angle than the four preceding figures.



(f) Six different traces are shown. The locally shortest traces will subsequently be found using numerical minimization. The shortest of those is the globally shortest.

Figure 3. Example of shortest path consisting of 4 geodesics and 2 supporting lines.

We let $\mathbf{v}_0$ and $\mathbf{v}_1$ be at the base of the cylinder patch (i.e. $\mathbf{v}_0$ and $\mathbf{v}_1$ are in the xy plane) . We now let $\mathbf{n_c}$ (in the world frame) be parallel to the local z axis and use this to defined a rotation matrix $\mathbf{M}$ for the cylinder patch that rotates vectors from the cylinder frame to a frame with axis aligned with the world frame. The right hand cylinder frame with z axis aligned with the cylinder axis is

$$\begin{aligned}\mathbf{n_c} &= \mathbf{v_0} \times \mathbf{v_1} \\ \mathbf{M_c} &= \left(\mathbf{v_0}, \quad \hat{\mathbf{n}}_\mathbf{c} \times \mathbf{v_0}, \quad \hat{\mathbf{n}}_\mathbf{c}\right) .\end{aligned}$$

The parametric equation for the position $\mathbf{g_c}(t)$ in the cylinder local frame is

$$\mathbf{g_c}(t) = \begin{pmatrix} \cos(\delta t + \theta) \\ \sin(\delta t + \theta) \\ t\alpha + z \end{pmatrix} , \tag{8}$$

for a helix that rises $2\pi\alpha$ per turn. The matrix $\mathbf{M_c}$ is then used to rotate this position to the world frame orientation as shown in (7). The angular offset θ and z are relative to $\mathbf{e_p}$, and $\delta \in \{-1, 0, 1\}$ determines the direction of rotation. The case of $\delta = 0$ happens when moving parallel to cylinder axis. These constants are all obtainable by solving the parametric equations at $t = 0$, where $\mathbf{g_c}(0) = \mathbf{e_p}$ and $\mathbf{g_c}'(0) = \mathbf{e_d}$. It can be shown that these constants are

$$\alpha = \frac{e_{d,z}}{\sqrt{{e_{d,x}}^2 + e_{d,y}^2}} \tag{9}$$

$$\theta = \text{atan2}(e_{p,y}, e_{p,x}) \tag{10}$$

$$z = e_{p,z} \tag{11}$$

$$\delta = \begin{cases} e_{d,y}/e_{p,x} & e_{p,x} \neq 0 \\ -e_{d,x}/e_{p,y} & e_{p,y} \neq 0, e_{p,x} = 0 \\ 0 & e_{p,x} = 0, e_{p,x} = 0 \, . \end{cases} \tag{12}$$

Edge and Horizon Crossover. We may leave the cylinder patch through one of its four edges or through one of two horizon segments. To identify values of t in (8) where an edge or horizon is crossed, we distinguish between three cases:

(A) Crossing to the neighboring sphere patches at the top or bottom edge. This then happens at $t_s = (\{0, |\mathbf{v}_2 - \mathbf{v}_1|\} - z)/\alpha$.

(B) Crossing to the neighboring facet patches at the left or right edge. This then happens at $t_c = \delta\ (\{0, \arccos(\mathbf{v}_1 \cdot \mathbf{v}_2)\} - \theta)$.

(C) Crossing a horizon. This then happens at (where Θ is the horizon angle relative to the view point defined in (2)) $t_h = \delta\ (\text{atan2}(w_y, w_x) - \theta \pm \Theta)$.

The edge resulting in the lowest positive t value is the one first intersected

$$t_{\text{end}} = \min \{t_e = \{t_s, t_c, t_h\} | t_e > 0\} \ .$$

The exit position and direction is then given by $\mathbf{g_c}(t_{\text{end}})$ and $\mathbf{g_c}'(t_{\text{end}})$.

2.3.2. Geodesic on Facet

The geodesic between two points on the facet is a straight line segment. Figure 3(c) shows such a line-segment on a facet patch. The parametric equation for this geodesic can be found from the entry point e_p and entry unit direction vector e_d

$$\mathbf{g_f}(t) = \mathbf{e_p} + t\ \mathbf{e_d} \ .$$

Obviously, the velocity vector at any point along the geodesic, including the exit point, equals the velocity vector at entry.

Edge Crossover. When leaving the facet an edge will be crossed that leads to a neighboring cylinder patch. Identifying which edge is being crossed, and where, is essentially a problem of identifying the intersection between two lines in $\mathbb{R}^2$. We use the 'perp dot' product of Hill [23] to find values of t where the geodesic path intersects each the edges of the polygon. When disregarding singularities where the ray and an edge of the polygon are parallel (the denominator becomes zero), the value of the interpolation parameter at the intersection point can be expressed as

$$t_{\text{end}} = \min \left\{ t_i = \frac{(\mathbf{v}_{i+1} - \mathbf{v}_i)^{\perp} \cdot (\mathbf{v}_i - \mathbf{e_p})}{(\mathbf{v}_{i+1} - \mathbf{v}_i)^{\perp} \cdot \mathbf{e_d}} \;\middle|\; i = 0, \ldots, N-1 \;\wedge\; t_i > 0 \right\} ,$$

where all points are transformed to the same plane and calculated in $\mathbb{R}^2$.

2.3.3. Geodesic on Sphere

A geodesic on a sphere is a great arc. Figure 3(e) shows this geodesic between an edge of the sphere patch and horizon. A geodesic $\mathbf{g_s}(t)$ is parameterized by the arc length t from a starting point $\mathbf{e_p}$ on a unit sphere where the path has initial direction $\mathbf{e_d}$ at unit velocity

$$\mathbf{g_s}(t) = \cos(t)\mathbf{e_p} + \sin(t)\mathbf{e_d} , \qquad t \in [0, \pi) . \tag{13}$$

Edge Crossover. We must also determine where the geodesic $g_s(t)$ leaves the sphere patch, that is, which edge segment of the patch is intersected and its intersection point. The geodesic path and all arc edges of the spherical polygon lay on great circles of the same sphere and all arcs edges have an angular span of less than π. There are N arc edges of a spherical polygon with N vertices v_i, indexed by i. If we extend these edges and consider intersection between their great circles, the geodesic path will intersect multiple great circles. However, because of convexity of the spherical polygon, the forward geodesic path can only intersect one edge (a segment of a great circle) inside the angular span of its arc.

Therefore, to determine where a particular edge intersects the forward geodesic path we first find the intersection between two great half-circles of en edge and the forward geodesic path. We do this by finding the intersection of two half-planes that each contains two great half-circles. These half-planes must intersect each other at a unique ray starting at the center of the sphere. Such a unique intersection always exists for a pair of different half-planes, and the cross product between the normals of the half-planes can be used to find a vector that represents this ray

$$\mathbf{r}_i = (\mathbf{e_p} \times \mathbf{e_d}) \times (\mathbf{v}_i \times \mathbf{v}_{i+1}) , \tag{14}$$

for a particular edge i. When disregarding cases where the half planes are equal ($\mathbf{r}_i$ is the zero-vector), the arc distance at the intersection point can be expressed as

$$t_{\text{end}} = \min \left\{ t_i = \arccos\left(\frac{\mathbf{r}_i}{|\mathbf{r}_i|} \cdot \mathbf{e_p} \right) \;\middle|\; i = 0, \ldots, N-1 \;\wedge\; t_i > 0 \right\} . \tag{15}$$

Horizon Crossover. For some $t \in (0, t_{\text{end}})$, the geodesic path might intersect the target horizon, at which point the geodesic path is complete. A (locally) shortest path has been found if and only if $\mathbf{g_s}'(t)$ intersect the view point of the target horizon. In this case, the geodesic path intersects the horizon at a right angle. That is, the two planes containing the geodesic path and horizon must have perpendicular normals.

We seek the angle t in (13) where intersection between geodesic path and horizon occurs. We do this by combining (5) with the plane equation that contains the geodesic path to get the intersection point relative to the frame containing the horizon. This point is then rotated to the frame of the geodesic with a rotation matrix $\mathbf{M_{hs}}$ and the arc distance t_{end} is obtained as

$$t_{\text{end}} = \arccos\left(\mathbf{e_p} \cdot \mathbf{M_{hs}} \mathbf{x} \right) \tag{16}$$

where

$$\mathbf{n_p} = \begin{bmatrix} n_{p,x} \\ n_{p,y} \\ n_{p,z} \end{bmatrix} = \mathbf{e_p} \times \mathbf{e_d} \qquad \text{and} \qquad \mathbf{x} = \frac{1}{1 - n_{p,z}^2} \begin{bmatrix} n_{p,y} \sqrt{1 - n_{p,z}^2 - d_r^2} - d_r n_{p,z} n_{p,x} \\ -n_{p,x} \sqrt{1 - n_{p,z}^2 - d_r^2} - d_r n_{p,z} n_{p,y} \\ d_r (1 - n_{p,z}^2) \end{bmatrix}$$

and d_r is given by (4).

2.4. Finding the shortest path

Now the parts that go into finding the shortest path have been defined. These are the initial and target points, the two supporting lines, two horizons, and a path on the surface of the CS obstacle. This path in turn consists of a number of geodesics, which is why it is called geodesic path. All candidates for a shortest path will consist of these components and is defined solely by the horizon parameter s that parameterize the initial horizon defined by (1). For any point $s \in [0; 1)$ on this close curve there is a unique trace (set of an initial supporting line, a geodesic path, and a target support line). Thus the distance from the initial point to the target point around the CS obstacle is a function $L(s)$ of the horizon parameter. This function may be infinite in the cases where the geodesic path on the surface of the obstacle continues indefinitely without ever intersecting the target horizon. Otherwise, this function is the distance from the initial point to the initial horizon plus the geodesic distance (in the direction given by the supporting line) along the surface to the target horizon plus the distance from the intersection point between geodesic path and target horizon to the target point. Mathematically, this is given as

$$L(s) = \left|\mathbf{w}_{\text{init}} - \mathbf{h}_{\text{init}}(s)\right| + T_N(s) + \left|\mathbf{w}_{\text{tar}} - \mathbf{g}(t_{\text{end}})\right| , \tag{17}$$

where T_N is the length of the geodesic path, $\mathbf{w}_{\text{init}}$ and $\mathbf{w}_{\text{tar}}$ are the initial and target view points, $\mathbf{h}_{\text{init}}$ is the initial horizon, and $\mathbf{g}$ is the geodesic path (making $\mathbf{g}(t_{\text{end}})$ the point where the geodesic path intersects the target horizon).

One method of finding the globally shortest path is to go through the horizon parameter range computing the distance and picking the smallest value. However, for this approach the initial horizon must be discretized and this introduces the question of the how dense the discrete points must be in order to ensure that this approach will provide the (close to) globally shortest path. Also, this approach can be computationally quite expensive even though for the individual paths do not require much computation to find the distance. We will therefore present a somewhat more subtle method for finding the globally shortest path.

2.4.1. Number of locally shortest paths

An important question is how many locally shortest paths exist around a given obstacle. To address this question we initially consider the geodesic paths between two points p and q on a polyhedron. Here we can 'unfold' the polyhedron to a planar surface by aligning neighboring faces in the same plane and then draw a line between p and q. If this line is inside the surface at all times it will be a representation of the geodesic path, which will in fact be the only candidate for the locally shortest for that particular sequence of polyhedron faces. We may obtain a number of different geodesic paths, depending on how the surface is unfolded. This number is bounded by $\Gamma(N)$, where N is the number of faces on the polyhedron. Often the actual number is much lower, since faces must be neighboring to allow unfolding.

If we initially disregard sphere patches, a similar unfolding can be done with the cylinder and facet patches. As before, we obtain different geodesic paths between p and q by traversing different sequences of patches (again assuming that the straight line from p to q stays within the unfolded surface). And as before, any two geodesic paths from an initial point p that traverse the same sequence of cylinder and facet patches diverge from the common intersection point p. There can be at most one path that intersects both p and q. But when including sphere patches, numerous geodesic paths may exist that traverse the same sequence of patches (now of any type) and converge towards common intersection points p, since any two geodesics on a sphere patch start to converge towards an intersection point after they have traveled a total angular span of at least π. This is also true for geodesics that do not share the same sequence of patches. Consequently, we cannot use the previous upper bound on the number of locally shortest paths. To develop an alternative we will need the concept of a full geodesic path.

Definition 9 (Full geodesic path)**.** *Let $\mathbf{g}(t)$ be a geodesic path starting ($t = 0$) at some point $\mathbf{p}$. Then $\mathbf{g}(t)$ is said to be full if it is defined for $0 \le t \le t_0$, where t_0 is given such that*

$$\int_0^{t_0} \frac{|\mathbf{g}'(t) \times \mathbf{g}''(t)|}{|\mathbf{g}'(t)|} dt = \pi \, .$$

That is, the total curvature of the path equals π.

Definition 10 (Full geodesic set)**.** *A geodesic set consisting of only full geodesic paths is called a full geodesic set.*

We know that any locally shortest path is part of a full geodesic path, and due to the smoothness of the CS obstacle we can also expect that there is a well-behaved correspondence between the length of the trace and the horizon parameter around any locally shortest path. That is, $L(s)$ is a function that needs only to be sampled in relatively few places to determine its local minima. This is because small changes in the initial condition of the geodesic path will have limited effect for the entire geodesic path, and as a consequence it seems reasonable to assume that finding the locally shortest traces can be achieved through a rather sparse sampling of the horizon parameter. We will attempt in the following to show the plausibility of this statement.

2.4.2. Envelope of geodesic paths

Observe first an example of a simple CS obstacle consisting of a few spheres, cylinders, and facets, shown in Figure 4. It is the same example as the CS obstacle in Figure 3. But rather than showing a few traces Figure 4 shows numerous 'consecutive' traces starting at the initial view point (hidden from view behind the obstacle) and reaching the blue target horizon at different points from where they all continue along supporting lines (with various green colors) to the target view point. It is obvious that the large sphere path that most of the traces follow (the part of the traces that is on the sphere patch is shown in red) deflects the traces such that they start to converge, and a number of the traces actually do intersect each other prior to reaching the target horizon. Such intersecting curves will in some circumstances generate an envelope, and part of such an envelope is indeed 'hidden' in the figure where the orange/red curves are intersecting. In fact, for any CS obstacle and any view point there is exactly one such envelope generated by the full geodesic set.

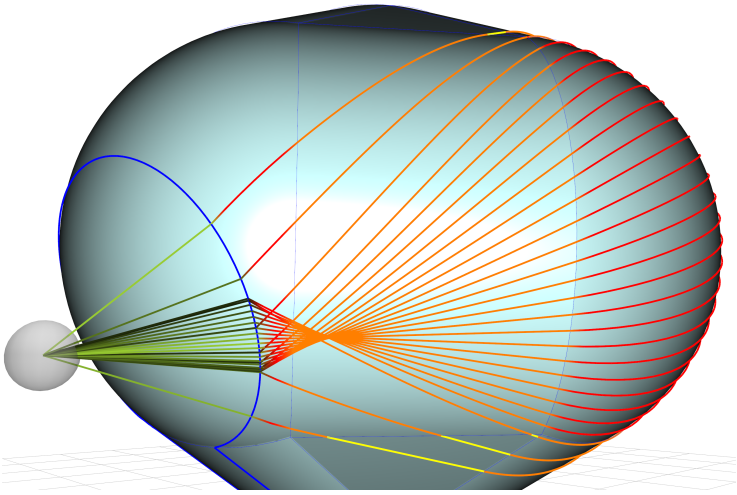


Figure 4. The traces start from the top right and continue around a large sphere patch towards the blue target horizon. At the horizon a greenish segment connects the trace to the view point. Its shade of green is lighter for traces that are closer to optimal determined by the angle error (see Section 2.4.3.

Theorem 1 (Geodesic envelope I)**.** *Let* $\mathbf{p}$ *be a radiant point on the CS obstacle. Let* $G = \{\mathbf{g}_\mathbf{p}\}$ *be the full geodesic set for* $\mathbf{p}$*. Then the envelope of* G *is either the empty set (in which case all* $\mathbf{g}_\mathbf{p}$ *end in a common intersection point) or it is a (diacaustic) continuous, closed curve. This curve is called the geodesic envelope for* $\mathbf{p}$*.*

Proof. Let $\mathbf{d}_1$ and $\mathbf{d}_2$ be two vectors tangent to the CS obstacle and both starting in $\mathbf{p}$, let θ be the angle between $\mathbf{d}_1$ and $\mathbf{d}_2$, and let $\mathbf{g}_{\mathbf{p},1}$, $\mathbf{g}_{\mathbf{p},2} \in G$ be the two full geodesic paths associated with the two directions. Then there exist δ independent of $\mathbf{d}_1$ and $\mathbf{d}_2$ such that for all $\theta < \delta$ there is at least one intersection (besides in the radiant point

$\mathbf{g_p}(0)$) between $\mathbf{g_{p,1}}$ and $\mathbf{g_{p,2}}$. This follows directly from the differentiability of the surface of the CS obstacle and the convergence towards each other of any two members of G.

Since any two geodesic paths will start to converge only after having traveled a combined angular span of more than π, it follows that since the total curvature of $\mathbf{g_p}$ is at most π any two members of G will intersect at most once (beside $\mathbf{p}$). It follows that the envelope of G is closed and continuous. □

The geodesic envelope is now defined for a point on the surface of the CS obstacle, but we are actually interested in a radiant point which is some distance from the obstacle (i.e. a view point). However, moving the radiant point away from the surface does not change the fundamental properties of the envelope.

Theorem 2 (Geodesic envelope II)**.** *Let* $\mathbf{w}$ *be a radiant view point, let* O *be a CS obstacle, and let* $\mathbf{h_w}(s)$ *be the corresponding horizon. Let* $G = \{\mathbf{g_w}\}$ *be the full geodesic set for* $\mathbf{w}$*. Then the envelope of* G *is either the empty set (in which case all* $\mathbf{g_p}$ *ends in a common intersection point) or it is a (diacaustic) continuous, closed curve.*

Proof. For $s_1,\ s_2 \in [0; 1)$ let $\mathbf{d}_1$ and $\mathbf{d}_2$ be two vectors tangent to the CS obstacle and starting in $\mathbf{h_w}(s_1)$ and $\mathbf{h_w}(s_2)$, respectively. Let θ be the angle between $\mathbf{d}_1$ and $\mathbf{d}_2$. There exist ε independent of s_1 and s_2 such that when $|s_1 - s_2| < \varepsilon$ then there is at least one intersection between $\mathbf{g_{w,1}}$ and $\mathbf{d}_2$. To see this imagine the CS obstacle augmented with the surface consisting of all supporting lines for $\mathbf{w}$. This new object is continuously differentiable except in $\mathbf{w}$. The result now follows from Theorem 1. □

The following theorem, if true, provides a rather low upper limit on the number of locally shortest traces. We have not been able to prove it, but in the following section we will discuss the background for believing that it is true.

Theorem 3 (Limit on number of locally shortest traces)**.** *Let* $\mathbf{w}_{init}$ *and* $\mathbf{w}_{tar}$ *be initial and target view points, respectively, and let* O *be a CS obstacle. Let* E *be the geodesic envelope for* $\mathbf{w}_{init}$ *on* O *and let* $\mathbf{h}_{tar}(u)$ *be the target horizon. Further, let* $u_0 < u_1 < \ldots < u_N$ *be the parameters of* $\mathbf{h}_{tar}$ *for which this horizon intersects* E*. Then the number of locally shortest traces is bounded by*

$$H(u_N, u_0) + \sum_{n=0}^{N-1} H(u_n, u_{n+1}) \ ,$$

where $H(u_n, u_{n+1})$ *is the number of horizon segments* $\mathbf{h}_i$ *(see Definition 4) completely or partly in the span* $[u_n, u_{n+1})$*.*

To justify this theorem we will in the next section introduce a simple method for determining whether a trace is locally shortest.

2.4.3. Shortest path

Any locally shortest trace must have the property that the entire trace is continuously differentiable as a consequence of geodesics being direction preserving. This is by definition ensured for all points except for the intersection point between the trace and the target horizon. At this point there is a potentially non-zero angle between the trace before and after the point. We define this angle as the angle error of the trace.

Definition 11 (Angle error)**.** *We define the angle error of a trace as an angle between geodesic velocity direction* $\mathbf{d}(t_{end})$ *at target horizon and vector towards target view point* $\mathbf{p_t}$ *at intersection point on target horizon* $\mathbf{h}(s)$*. For any locally shortest path the angle error*

$$\epsilon = \arccos\left(\mathbf{d}(t_{end}) \cdot \frac{\mathbf{p_t} - \mathbf{h}(s)}{|\mathbf{p_t} - \mathbf{h}(s)|}\right)$$

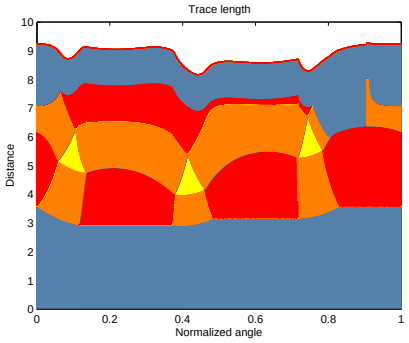
is zero and the derivative of the length $L'(s)$ *is also zero.*

An example of angle errors can be seen in Figure 4. Here all the shown traces intersect the target horizon, and the value of the angle error is shown by the darkness of the green color of the supporting lines. Lighter means smaller, and darker means larger angle error.

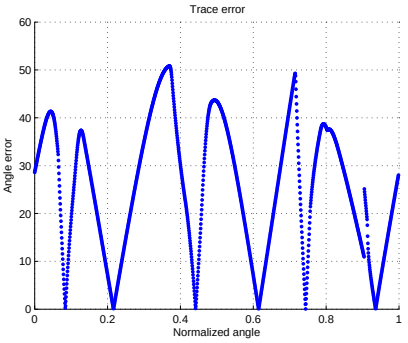
A more complete view of the traces are shown in Figure 5. Figure 5(a) shows the length of all traces as a function of the initial horizon parameter, and Figure 5(b) shows the corresponding angle error. A 3D view of the traces can be seen in Figure 6. The apparent discontinuity seen in both plots when the horizon parameter is 0.9 happens at the

transition between patch sequences. This discontinuity can happen when two subsequent trace intersects different horizon segments because the horizon is non-convex. Figure 6 shows this in details at the blue region. In a practical implementation the horizon sweep will also be discontinuous for traces that are ended because they do not reach the target horizon within some expected distance (they are too long even before they reach the target horizon). The difficulty of reaching the target horizon depends on the size of target horizon and the obstacle. For large obstacles and a small target horizon, it can be necessary to use a higher sweeping resolution to find the optimum.

Since a limited change in horizon parameter within a neighborhood of a local optimum results a limited change in both angle error and path length the problem of determining the horizon parameter at which a local optima occurs is a strictly quasiconvex problem, which can be solved with convex optimization after identifying each convex region.



(a) Length of trace for entire initial horizon. Blue parts are between view points and horizons. Facets are yellow, cylinders are orange, and spheres are red.



(b) Angle error for entire initial horizon. Notice how the local minima in the trace length plot to he left corresponds to the zero angle errors.

Figure 5. Traces for CS obstacle in Figure 4.

3. Results

We show a few examples of shortest paths created for a small-scale helicopter flying between different positions, denoted via points or VPs, and obstacles. The helicopter bounding sphere has a radius r of 1.70 meter. To give some idea of how much time is required to find a path, each test case is repeated several times and the maximum required time recorded [1].

3.1. Teapot

Figure 7 shows trace results for the example in Figure 1 that has a high number of patches. Most traces newer reach the target horizon. To reduce computations, a trace is terminated when its length is longer than a previous trace or $r\pi$, that is, half the circumference of a great arc of the obstacle bounding sphere. The trace shows that there are three geodesics candidates for optimum. The geodesic of the shortest path is found close to the starting point. The shortest path between the two VPs took less than 160 milliseconds in repeated tests at a trace resolution of 1440 prior to minimization. They were found in less than 7 milliseconds when estimating trace resolution.

[1] All tests were done on a single core of a 2.2GHz Intel Core 2 Duo laptop

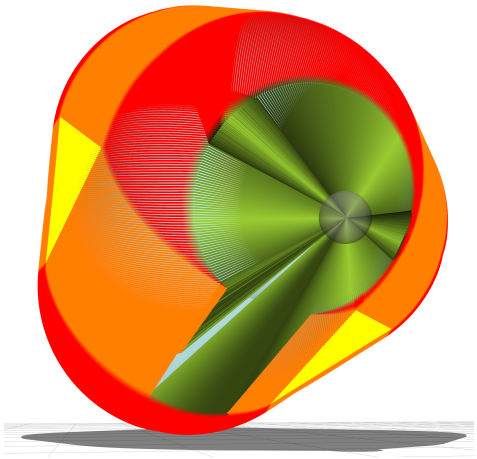
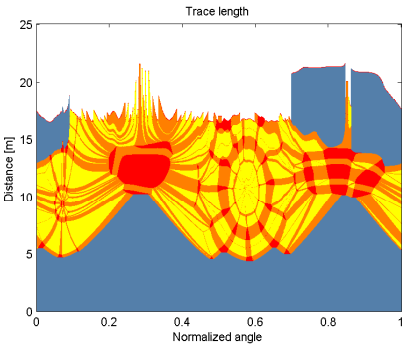
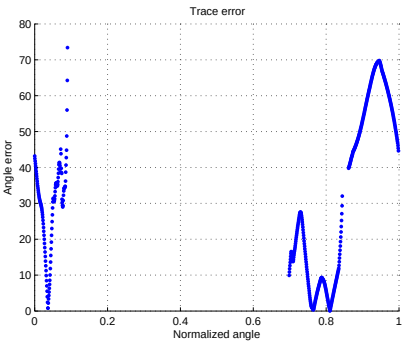


Figure 6. Dense tracing for the obstacle in Figure 3(f). The six locally shortest paths can be clearly seen from the lighter green traces. At the blue region the horizon parameter is 0.9.



(a) Length of trace path for different sweep angles. Blue parts are between view point and horizon. Facets are yellow, cylinders are orange, and spheres are red. The top of the teapot can be recognized around 0.1 on the x-axis and the bottom at 0.6. The large red patch at 0.3 is the spout. The handle can be seen at 0.8.



(b) Angle error for different sweep angles. All geodesics must have zero angle error

Figure 7. Trace results for example in Figure 1.

3.2. Multiple obstacles

The presented method is only described here for single obstacles. A piecewise optimal path can found between several obstacles, if an estimated shortest path exists. The problem is solved by adding intermediate VPs between obstacles, and solving the path planning problem for consecutive VPs, see Figure 8. A good estimation on where to place these can be obtained by initially finding an approximated shortest path, using the method from [11]. This path consists of connected line segments, and intermediate VPs are placed midway on segments with endpoints on different obstacles. This approach will only gives estimation on the globally optimal path, since optimality can only be achieved between subsequent VPs. The shortest path between the four successive VPs took in repeated tests less than 60 milliseconds at a trace resolution of 1440 prior to minimization. They were found in less than 5 milliseconds when estimating trace resolution.

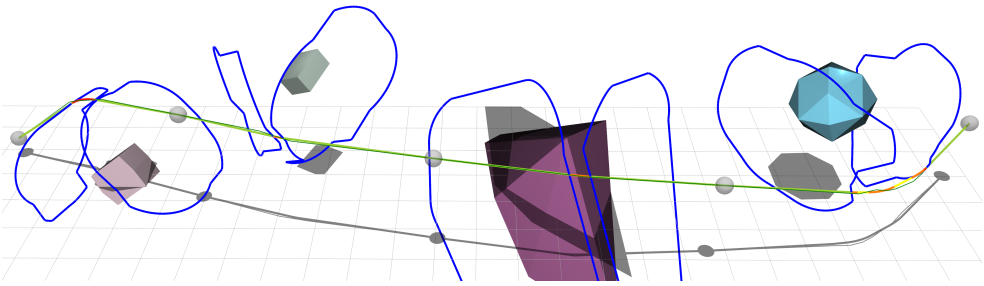


Figure 8. Obtaining the shortest path between the two end points using the described method is made difficult by the fact that there are four obstacles, although none intersect in CS.

3.3. Flying around building

A test case is created where a helicopter must fly between the five via points that are placed in a star shaped pattern. The setup is taken from [24]. The paths must be planned such that the helicopter avoids a building, centered between the via points. The setup and resulting paths can be seen in Figure 9. The full path between all VPs is found by solving the five path planning problems for connected VPs. Five shortest paths between five successive VPs around a building were in repeated tests found in less than 500 milliseconds at a trace resolution of 1440 prior to minimization. They were found in less than 50 milliseconds when estimating trace resolution. The approach in [24] spends 600 milliseconds obtaining a visibility graph that must subsequently be searched to obtain an approximated path.

3.4. Wind turbine

We use the test case from [25] with three additional VPs, where a helicopter must fly near the wind turbine between VPs as seen in Figure 10. Since the method as presented can only handle one obstacle at a time, it can be problematic to apply it to a compound model such as the V52. One approach is to build a single WS obstacle from the convex hull of the compound model. However, this would cause some VPs to not be connected in CS. Instead, we use the approximated path from [25] to add intermediate via points on the approximated path between, such that each part of the geodesic path is only located on a single obstacle. This approach means that the optimal path will only be optimal to, and between, the intermediate via points. Nine shortest paths between nine successive VPs near a Vestas V52 wind turbine were in repeated tests found in less than 600 milliseconds. A trace resolution of 1440 was used prior to minimization.

4. Discussion

In this paper we solve the shortest path planning problem for a sphere moving amongst a single polyhedron in work space. Finding such a shortest path is difficult, since a vehicle traveling along this path does not traverse only vertices of the WS polyhedron, as in the 2D case, but also points on its edges.

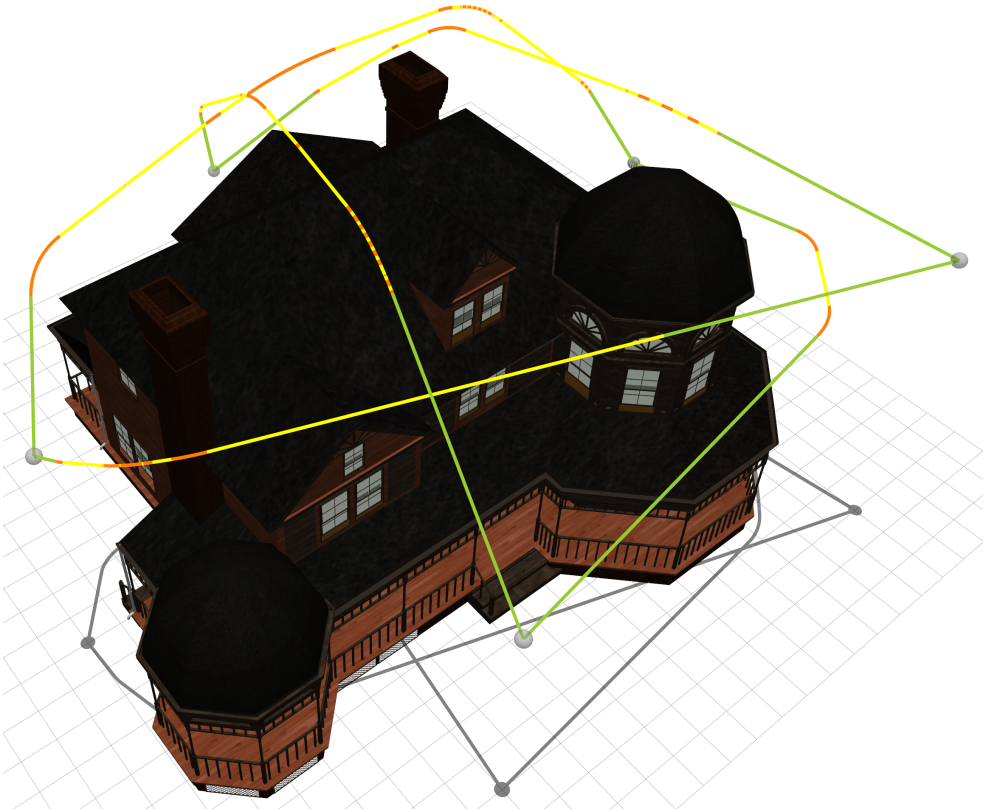


Figure 9. Shortest paths for a small-scale helicopter flying around a building between five VPs.

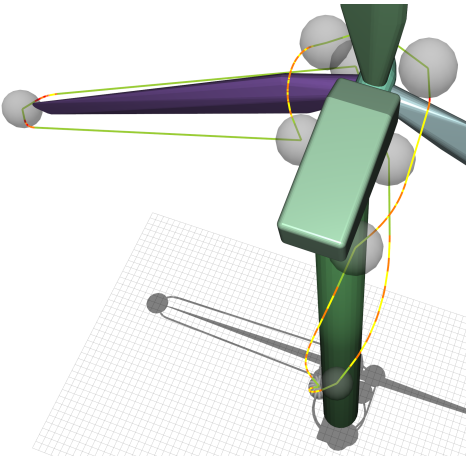


Figure 10. Shortest paths for a small-scale helicopter flying near a Vestas V52 wind turbine between nine VPs.

The CS obstacle from the Minkowski sum of a WS obstacle and a spherical vehicle will consist of patches of spheres, cylinders and facets, and have a continuously differentiable surface. As results, the shortest path must also be continuously differentiable, and consist of combination of tangents and geodesics. The tangents will connect the initial and target point to the obstacle at the horizons of these points. The geodesic must connect the horizons to complete the path, in such way that it is continuously differentiable. The problem now becomes that of finding a point on one horizon, such that the full path will be continuously differentiable and shortest.

We present a problem formulation that makes this problem strictly quasiconvex inside some subsets. To initially find a good approximation to the shortest path, we sample (or trace) geodesics in different directions and compare length. For a sufficiently high sampling resolution the subset containing the optimal path will be found. We subsequently use convex optimization to find the shortest path inside this subset. The presented approach is capable of finding paths for several examples. This includes examples with several obstacles, although a suboptimal path is found in such cases, since optimally can currently only be assured between single obstacles.

Our method works well in combination with approximation based methods that gives a continuous, but not continuously differentiable path. Often, such a path can be split into several pieces, where each covers no more than one obstacle, and our method applied between these split points to give a piecewise optimal continuously differentiable path.

5. Future Works

The method presented has not been implemented for multiple convex obstacles. While multiple obstacles would increase the complexity, the method scales very well to more obstacles. When tracing a geodesic path there will be at most two options per additional obstacle to continue on that instead of the current. This is because a geodesic path can have at most two supporting lines to another obstacle that are tangents to a helices or arcs. This means that when traveling on a geodesic path, an additional trace should be made just as an obstacle would appear on the horizon. If the obstacles intersect in CS, the current geodesic could be blocked, and the new trace would be the only way to continue. In other cases a new additional trace should be made just at the other obstacle leaves the horizon.

The method presented here could be used on polyhedron CS obstacles as well. Such obstacles would only consist of planar patches and the horizon would coincide with edges on the obstacle. Consequently, obtaining shortest paths would require a less complex approach, where multiple obstacles could more easily be supported. However resulting paths would not be continuous differential.

References

[1] N. J. Nilsson, A mobile automaton: An application of artificial intelligence techniques, In 1st International Conference on Artificial Intelligence, pages 509-520, 1969.

[2] T. Lozano-Perez, M. A. Wesley, An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles, Communications of the ACM, 22, 1979.

[3] J.-C. Latombe, Robot motion planning, Springer, 1991.

[4] J. S. B. Mitchell, A note on two problems in connexion with graphs, Handbook of Discrete and Computational Geometry, CRC Press, 1997.

[5] J. Hershberger, S. Suri, Efficient computation of Euclidean shortest paths in the plane, In Proceedings IEEE Symposium on Foundations of Computer Science, pages 508-517, 1995.

[6] J. S. B. Mitchell, Shortest paths among obstacles in the plane, International Journal Computational Geometry & Applications, 6(3):309-332, 1993.

[7] S. M. LaValle, Planning Algorithms, Cambridge University Press, 2006.

[8] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, MIT Press, 2005.

[9] J. Canny, J. H. Reif, New lower bound techniques for robot motion planning problems, In Proc. 28th Annu. IEEE Sympos. Found. Comput. Sci., 1987.

[10] C. H. Papadimitriou, An algorithm for shortest-path motion in three dimensions., Inf. Process. Lett. (1985) 259–263.

[11] F. Schøler, A. la Cour-Harbo, M. Bisgaard, Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning, Springer, submitted, 2012.

[12] S. Har-Peled, Constructing approximate shortest path maps in three dimensions, SIAM J. Comput., 2005.

[13] P. K. Agarwal, S. Har-Peled, M. Sharir, K. R. Varadarajan, Approximate shortest paths on a convex polytope in three dimensions, J. ACM, 1997.

[14] K. R. Varadarajan, P. Agarwal, Approximating shortest paths on an nonconvex polyhedron, In Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci., Miami Beach, Florida, 1997.

[15] M. Sharir, A. Schorr, On shortest paths in polyhedral spaces, SIAM J. Comput., 1986.

[16] J. Chen, Y. Han, Shortest paths on a polyhedron, SCG '90 Proceedings of the sixth annual symposium on Computational geometry, 1990.

[17] B. Kaneva, J. O'Rourke, An Implementation of Chen & Han's Shortest Paths Algorithm, Proc. of the 12th Canadian Conference on Computational Geometry, New Brunswick, 2000.

[18] S. Cao, S. A. Greenhalgh, Finite-difference solution of the eikonal equation using an efficient, first-arrival wavefront tracking scheme, Geophysics, 59, no. 4, 632-643, 1994.

[19] J. Sethian, Level Set Methods and Fast Marching Methods Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, Cambridge University Press, Monograph on Applied and Computational Mathematics, 1999.

[20] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1: 269271, 1959.

[21] R. Kimmel, J. A. Sethian, Computing geodesic paths on manifolds, Proc. Nat. Acad. Sci. 95, 1998.

[22] H. Tietze, Famous Problems of Mathematics: Solved and Unsolved Mathematics Problems from Antiquity to Modern Times, New York: Graylock Press, 1965.

[23] F. Hill, The Pleasures of 'Perp Dot' Products, Graphics Gems IV, Academic Press Professional, 1994.

[24] F. Schøler, A. la Cour-Harbo, M. Bisgaard, Generating Approximative Minimum Length Paths in 3D for UAVs, IEEE Intelligent Vehicles Symposium, submitted, 2012.

[25] F. Schøler, A. la Cour-Harbo, M. Bisgaard, Configuration Space and Visibility Graph Generation from Geometric Workspaces for UAVs, AIAA Guidance, Navigation, and Control Conference and Exhibit, 2011.

Paper E

Collision Free Path Generation in 3D with Turning and Pitch Radius Constraints for Aerial Vehicles

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

This paper was published in:
Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit, 2009

Revised to correct typographical errors

Collision Free Path Generation in 3D with Turning and Pitch Radius Constraints for Aerial Vehicles

Flemming Schøler*, Anders la Cour-Harbo†, and Morten Bisgaard‡

Aalborg University, Department of Electronic Systems, Section of Automation and Control, Denmark

In this paper we consider the problem of trajectory generation in 3D for uninhabited aerial systems (UAS). The proposed algorithm for trajectory generation allows us to find a feasible collision-free 3D trajectory through a number of waypoints in an environment containing obstacles. Our approach assumes that most of the aircraft structural and dynamic limitations can be formulated as a turn radius constraint and that any two consecutive waypoints have line-of-sight. The generated trajectories are collision free and also satisfy a constraint on the minimum admissible turning radius, while allowing faster flight if appropriate. The work has been carried out with reference to the Bergen Industrial Twin helicopter and the 3D path planner from the Autonomous Vehicle Group at Aalborg University. Simulation results for the trajectory generation are presented, which are obtained using a detailed model of the Bergen Industrial Twin helicopter.

Nomenclature

$\overline{W}_c$	Current waypoint
$\overline{W}_p$	Previous waypoint
$\overline{W}_n$	Next waypoint
$\overline{P}_a$	Approach path
$\overline{P}_d$	Departure path
$\overline{O}_w$	Waypoint arc location
$\overline{O}_a$	Approach arc location
$\overline{O}_d$	Departure arc location
R	Turn and pitch radius
A_w	Waypoint arc
A_a	Approach arc
A_d	Departure arc
Subscript	
W	Vector is located in $\mathbb{R}^3$
P	Vector is located in waypoint plane frame
x	X-component of vector
y	Y-component of vector
z	Z-component of vector

I. Introduction

UAS have become a preferred, indispensable, and increasingly used platform for many applications where manned operation is considered unnecessary, repetitive, or too dangerous.[1,2] Trajectory generation is a

*PhD Student, Department of Electronic Systems, Section of Automation and Control, Fredrik Bajers Vej 7C DK-9220, Aalborg East, Denmark.

†Associate Professor, Department of Electronic Systems, Section of Automation and Control, Fredrik Bajers Vej 7C DK-9220, Aalborg East, Denmark.

‡Assistant Professor, Department of Electronic Systems, Section of Automation and Control, Fredrik Bajers Vej 7C DK-9220, Aalborg East, Denmark.

fundamental area for UAS development in general[3] and is usually used to find a path through an environment under a set of constraints. However, such paths usually consist of a number of waypoints, which are connected by line-segments. This makes the paths difficult to follow considering the structural and dynamic capabilities of many UAS. Instead, the waypoints should be interconnected through a feasible and collision-free trajectory.

Efforts have been made to use Dubins ideas[4] for trajectory generation in 2D[5,6] and 3D,[7] where the generated continuously differentiable trajectory satisfy initial and final conditions to position and heading angle, and it is assumed that the UAS has constant turn radius. However, these approaches does not account for trajectories being obstructed by obstacles, which makes the methods less suitable for flight in environments with many obstacles.

This paper describes an algorithm for generating a collision-free 3D trajectory under the assumption that most of the aircraft structural and dynamic limitations can be translated to a constant turn and pitch radius constraint. It is also assumed that any two consecutive waypoints are within line-of-sight and have enough spacing to allow the vehicle to return to this line before proceeding to make another turn.

A full state linear-quadratic feedback controller is used for trajectory tracking. It uses a guidance law similar to the line-of-sight guidance algorithm,[8] which relies on 'good helmsman' behaviour to asymptotically track the trajectory. This method performs well in simulation using a detailed model of the Bergen Industrial Twin helicopter.[9] The helicopter model is simplified to include 8 rigid body states; helicopter position and yaw angle, and helicopter velocities and angular rate of yaw angle.

A. Background

The 3D path planner developed at AAU[10] generates a configuration space based on triangulation of the Minkowski addition of the UAS bounding sphere radius and the hull of work space obstacles, making it possible to operate with obstacles of any shape. In Figure 1 a cross section of a configuration space is shown. For one obstacle, it is shown how the bounding sphere radius expands the cross section of workspace obstacle into that of a configuration space obstacle. The actual workspace obstacle is slightly larger in order to reduce size of obstacle description.

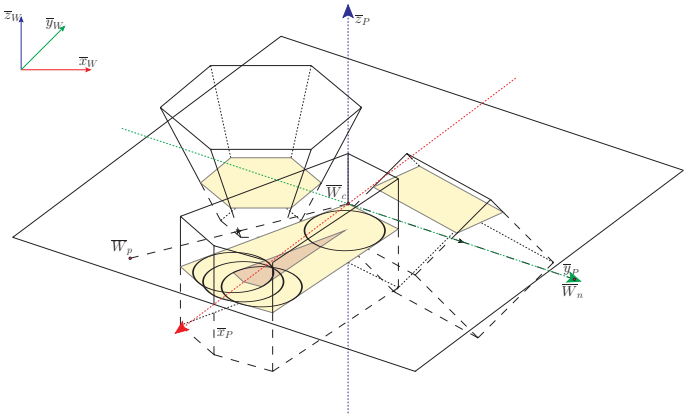


Figure 1. Cross section of three configuration space obstacles, with workspace shown for one obstacle. Three waypoints located in cross section are drawn in line-of-sight.

A visibility graph is constructed from this configuration space in which the shortest path can be constructed as a series of connected waypoints in line-of-sight. We aim at finding the shortest path, since this approach is model independent, but other criteria for optimization are supported by the search algorithm. Since the visibility graph is constructed in 3D, it is based on approximation through a finite number of points located on surfaces of the configuration space obstacles. Thus, the line-of-sight trajectory is only locally shortest among the vertices in the visibility graph and not globally shortest as it would be in the 2D case.

The implementation of the path planner generates a path based on the convex hull. As a consequence,

if non-convex regions of the configuration space should potentially be included in a solution, non-convex obstacle must be described as several convex obstacles.

II. Method

A continuous but non-differential path is difficult for a UAS to track closely, since it would require instantaneous changes in velocity at waypoints where the path changes direction. A path will well approximate a smooth curve by having a rather large bounding sphere radius, since configuration space obstacles will then have rather small angles between any two adjacent surfaces. In an effort to generate an efficient path this is not considered here. Instead, a differentiable curve is found for each separate waypoint, according to the flight approach path $\overline{P_a}$ and departure path $\overline{P_d}$ at the waypoint, see Equation 1 and 2. While this reduces the trajectory generation problem to two dimensions, it also results in an $O(n)$ complex algorithm, with n being the number of waypoints.

$$\overline{P_a} = \overline{W_c} - \overline{W_p} \quad (1)$$
$$\overline{P_d} = \overline{W_n} - \overline{W_c} \quad (2)$$

In Section A we first describe how a plane is aligned with the approach and departure path vectors at the waypoint of interest. In Section B, we then describe a method that tries to place an arc that intersects the waypoint but also avoids obstacles in this plane. This arc intersects lines formed by $\overline{P_a}$ and $\overline{P_d}$, and can subsequently be connected to them to give a continuously differentiable curve, consisting of three arcs, as described in Section C. To generate a complete position trajectory, the procedure is repeated for all waypoints and curves at each waypoint is connected by line-segments, as described in Section D. In section E we show how a full state trajectory is generated by determining velocities along the trajectory.

A. Waypoint Plane

When determining the location of the waypoint plane, which will contain the three connected arcs at each waypoint, a transformation matrix must be found that maps $\mathbb{R}^3$ to this plane, as seen in the Figure 1. The figure shows such a plane based on three waypoints where the location of the plane can be determined from $\overline{W_c}$, $\overline{P_a}$, and $\overline{P_d}$. The transformation matrix that maps the plane frame to $\mathbb{R}^3$ is given by the affine mapping $\mathbf{A}_P^W$:

$$\mathbf{R}_P^W = \begin{bmatrix} \overline{x}_P & \overline{y}_P & \overline{z}_P \end{bmatrix}$$
$$\mathbf{A}_P^W = \left[\begin{array}{c|c} \mathbf{R}_P^W & \bar{t}_P \\ \hline 0 & 1 \end{array}\right]$$

A top view of the plane can be seen in Figure 2. In general, a transformation matrix mapping from $\mathbb{R}^3$ to the plane frame $\mathbf{A}_W^P$ can be found by inverting $\mathbf{A}_P^W$. The vectors $\overline{x}_P$, $\overline{y}_P$ and $\overline{z}_P$ in the rotation matrix $\mathbf{R}_P^W$ are unit vectors and mutually orthogonal. This means that the transformation matrix mapping from $\mathbb{R}^3$ to the plane frame can be found transposing the rotation matrix and negating the translation vector $\bar{t}_P$:

$$\mathbf{A}_W^P = \left[\begin{array}{c|c} \left(\mathbf{R}_P^W\right)^T & -\bar{t}_P \\ \hline 0 & 1 \end{array}\right] \quad (3)$$

The transformation matrix used to transform vertices from $\mathbb{R}^3$ to the plane frame can be calculated from the following properties, also shown in Figure 1:

- The plane normal is aligned with $\overline{z}_P$. This means that any point on the plane will have a zero z-component in the plane frame.
- The vector $\overline{y}_P$ is aligned with the vector towards the next way-point. This means that any point on the vector towards the next way point will only be non-zero in its y-component in the plane frame.

- The vector $\overline{x}_P$ is normal to vectors $\overline{y}_P$ and $\overline{z}_P$.
- The plane frame origin $\overline{t}_P$ is at $\overline{W_c}$.

These properties result in the following equations, which allows for mapping between $\mathbb{R}^3$ and the plane frame.

$$\begin{aligned} \overline{z}_P &= \overline{P_{dW}} \times \overline{P_{aW}} \\ \overline{y}_P &= \overline{P_{dW}} \\ \overline{x}_P &= \overline{y}_P \times \overline{z}_P \\ \overline{t}_P &= \overline{W_{cW}} \end{aligned}$$

Once the plane frame has been found, the location of the waypoint arc in the frame can be specified as a two-dimensional vector.

B. Waypoint Arc

The waypoint arc (A_w) takes care of most of the turn, and its location can vary to accommodate for obstacles in the vicinity. It is located in the waypoint plane and intersects $\overline{W_c}$. Its location is bounded by $\overline{P_a}$, $\overline{P_d}$ and possible obstacles located within twice the pitch and turning radius. The waypoint arc is located on arc C_p in Figure 2, which shows bounds set by $\overline{P_a}$, $\overline{P_d}$. Figure 3 shows two different cases of intersecting obstacles. In one case the arc location is limited by a vertex and in the other case by an edge. Each case is treated separately.

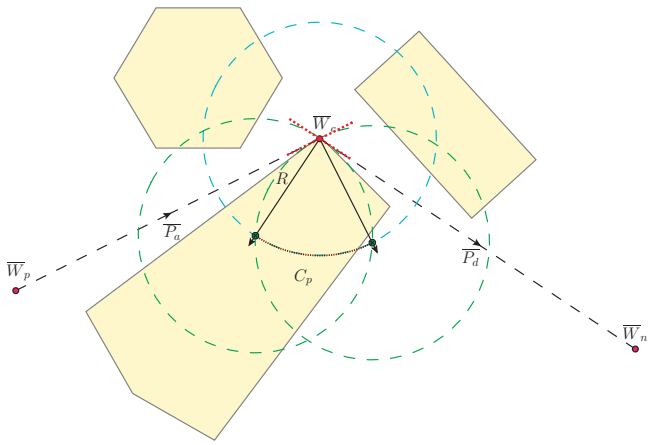


Figure 2. Waypoint plane with cross section of obstacles and waypoints in line-of-sight. The circles have pitch and turning radius R. The two circles through $\overline{W_c}$ mark the extreme locations of A_w on C_p when bounded by $\overline{P_a}$, $\overline{P_d}$.

In the former case, candidates for the bounded locations $\overline{O_v}$ of A_w based on a vertex $\overline{P_v}$ can be found from Equation 4. The scalar D_v is negative when the vertex is out of range.

$$D_v = R^2 - \frac{\left\|\overline{W_c} - \overline{P_v}\right\|^2}{4}$$

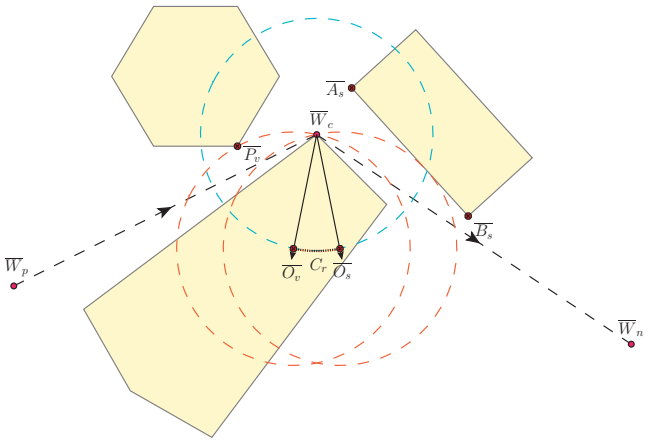


Figure 3. Waypoint plane with cross section of obstacles and waypoints in line-of-sight. The circles have pitch and turning radius R. The two circles through $\overline{W_c}$ mark the extreme locations of A_w on C_p when bounded by obstacles. The circle on the left intersects a vertex, while the right-most circle intersects a segment.

$$\overline{O_v} = \frac{\overline{W_c} + \overline{P_v}}{2} \pm \sqrt{D_v} \begin{bmatrix} \overline{W_{cy}} - \overline{B_{vy}} \\ \overline{B_{vx}} - \overline{W_{cx}} \end{bmatrix} \tag{4}$$

In the latter case, candidates for the bounded locations $\overline{O_s}$ of A_w based on segment end-points $\overline{A_s}$ and $\overline{B_s}$ can be found by solving the second order equation in Equation 5. In cases where end-points have same x-component, $\overline{O_s}$ can be found by interchanging the x and y-components of all vectors in Equation 5 to 8.

$$0 = \left(\alpha_s{}^2 + 1\right)\overline{O_{sx}}{}^2 + \left(2\alpha_s\left(\beta_s - \overline{W_{cy}}\right)^2 - 2\overline{W_{cx}}\right)\overline{O_{sx}} + \left(\beta_s - \overline{W_{cy}}\right)^2 + \overline{W_{cx}}{}^2 - R^2 \tag{5}$$

$$\overline{O_{sy}} = \overline{O_{sx}}\,\alpha_s + \beta_s \tag{6}$$

Where:

$$\alpha_s = \frac{\overline{A_{sy}} - \overline{B_{sy}}}{\overline{A_{sx}} - \overline{B_{sx}}} \tag{7}$$

$$\beta_s = \frac{\overline{A_{sx}}\overline{B_{sy}} - \overline{B_{sx}}\overline{A_{sy}} \pm R\|\overline{A_s} - \overline{B_s}\|}{\overline{A_{sx}} - \overline{B_{sx}}} \tag{8}$$

When multiple obstacles exist within the plane, the set of obstacles limiting C_r most is used. The exact location of the arc $\overline{O_w}$ is found by choosing the midpoint on C_r between the two most limiting locations of $\overline{O_v}$ or $\overline{O_s}$. This ensures that a vehicle following the path is as far from all obstacles as possible. If obstacles are too restricting on C_r and the waypoint cannot be passed, a solution might be found by reducing R by reducing speed, which might not be an option for all types of vehicles.

C. Approach and Departure Arc

In the general case, two additional arcs must be added to the waypoint plane to form a continuously differentiable path, as seen in Figure 4. In the figure, a vehicle must follow the approach vector $\overline{P_a}$ from $\overline{W_p}$

towards $\overline{W_c}$. When it reaches the approach arc A_a, it follows this until the waypoint arc A_w is intersected. The waypoint arc is followed onto the departure arc A_d leading the vehicle to the departure path $\overline{P_d}$ and on towards the next waypoint $\overline{W_n}$.

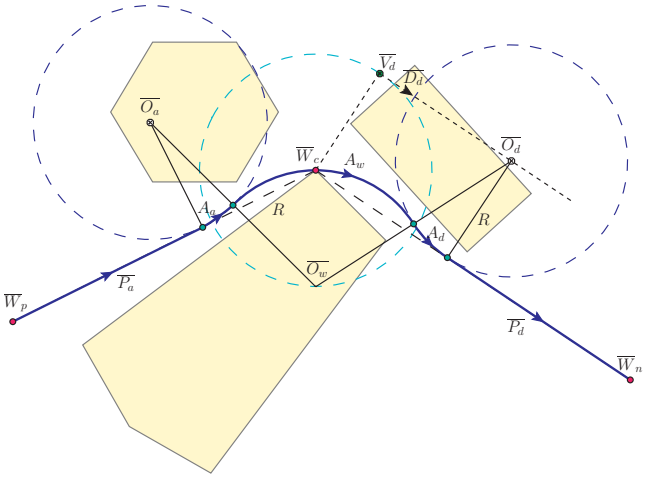


Figure 4. Approach, waypoint, and departure arcs for a waypoint. The thick curve shows the generated path.

Candidates for $\overline{O_d}$ of the departure arc A_d can be found by solving the second order equation in Equation 9. The location of $\overline{V_d}$ is shown in Figure 4. In cases where end-points have same x-component, $\overline{O_d}$ can be found by interchanging the x and y-components of all vectors in Equation 9 to 12.

$$0 = \left(\alpha_c^2 + 1\right)\overline{O_{dx}}^{\,2} + \left(2\alpha_c\left(\beta_c - \overline{O_{wy}}\right) - 2\overline{O_{wx}}\right)\overline{O_{dx}} + \alpha_c^2 + \overline{O_{wx}}^{\,2} - 4R^2 \quad (9)$$

$$\overline{O_{dy}} = \overline{O_{dx}}\ \alpha_c + \beta_c \quad (10)$$

Where:

$$\alpha_c = \frac{\overline{D_{dy}}}{\overline{D_{dx}}} \quad (11)$$

$$\beta_c = \overline{V_{dy}} - \alpha_c \overline{V_{dx}} \quad (12)$$

Candidates for $\overline{O_a}$ of the approach arc A_a can be found using similar approach. The approach and departure arcs must also be tested for intersection with obstacles, but are likely to be collision free since the combined trajectory remain close to A_w, and $\overline{P_a}$ or $\overline{P_d}$, which are all collision free.

D. Connecting Paths

To generate a complete trajectory, the approach, waypoint, and departure arcs are found for each waypoint, excluding the final and initial position. Consecutive waypoints are connected using line-segments that connect a departure path from a waypoint to the approach departure path of the next waypoint. This forms a continuously differentiable trajectory, since these vectors are aligned. If there is insufficient spacing between the waypoints, the path cannot be connected since the pitch and turning radius R is too great. In worst cases, such as if the vehicle is requested to fly to a waypoint and then proceed to the next waypoint in the opposite direction at two consecutive waypoints, the minimum spacing required to perform this operation is

$(2+\sqrt{12})R \approx 5.5R$. A way around this is to reduce velocity, although this might not be an option for all types of vehicles.

E. Full State Trajectory

The trajectory is made as a collection of paths describing the characteristics of each part of the trajectory. In order to generate a state trajectory, containing x, y, z, and yaw position and derivatives, velocities must be assigned along the trajectory. This is done by defining the desired functions for acceleration and deceleration and maximum velocity along each path in the trajectory. These may be depending on the flight direction.

An algorithm is used to enumerate each path and assign entry and exit velocity, see Equation 13 with v a functions of local time and path. The equation contains three functions: v_{acc}, v_{max}, and v_{dec}, which define velocity when accelerating, flying at maximum velocity, and decelerating.

$$\begin{aligned} v(t,p) &= \min(v_{acc}(t,p), v_{max}(p), v_{dec}(t,p)) \\ \text{Where:} & \\ v(t_0,p_i) &= v(t_f,p_{i-1}) \\ v(t_0,p_0) &= v_s \\ v(t_f,p_f) &= v_s \end{aligned} \tag{13}$$

In order to get a consistent trajectory, the exit velocity of any path must match entry velocity of next path, and initial and final entry velocity at the trajectory is zero (helicopter is hovering). When a path with inconsistent demands is met, the algorithm back propagates to decrease velocity as required. Inconsistent demands could be an unreachable exit velocity, because the acceleration is too low to reach this velocity in due time. In cases where there is insufficient spacing between waypoints, maximum velocity is reduced accordingly in order to reduce required turn and pitch radius R. Yaw angle is set to flight direction, although this is not a strict requirement for the helicopter.

III. Results

The capabilities of the trajectory generation algorithm will be demonstrated through two tests. The tests also aim at determining how difficult it is for a tracking controller to track the trajectory. In the first test we generate a trajectory that takes the vehicle through a number of predefined manoeuvres, such as climbing, turning, and turning while descending. The second test brings the vehicle from one location to another through a number of waypoints generated by the 3D path planner in an environment with obstacles. In both tests a Bergen Industrial Twin helicopter model is used to simulate flight using the generated trajectory as reference, while a LQR control and a helmsman control strategy is used to track the trajectory.

A. Generating Trajectory from Waypoints

In the manoeuvre test, the helicopter will be taken to four waypoints. It starts off by climbing two meters. It then proceeds two meters forward, where it makes a 90° right turn, continues two meters, and makes an approximately 56° left turn directed slightly downwards. It then lands approximately four meters from the turn. Figure 5 shows a side view of the waypoints and the generated trajectory.

Results from tracking the trajectory can be seen in Figure 6 along with statistics in Table 1. The figure shows that the tracker receives a reference that is tracked in satisfactory manner. The max deviance of 0.14 m in the Table 1 is the furthest distance to the trajectory the helicopter had throughout the flight. This value should be added to the bounding sphere radius to avoid potential collisions with obstacles. The average position error is 0.05 m.

B. Generating Trajectory from Path Plan

A test is made to verify that a system is able to track a trajectory generated based on a typical scenario with data from the Path planner. The purpose is to test if a trajectory can be travelled by a helicopter model controlled by a trajectory regulator. The helicopter must stay clear of obstacles in the area.

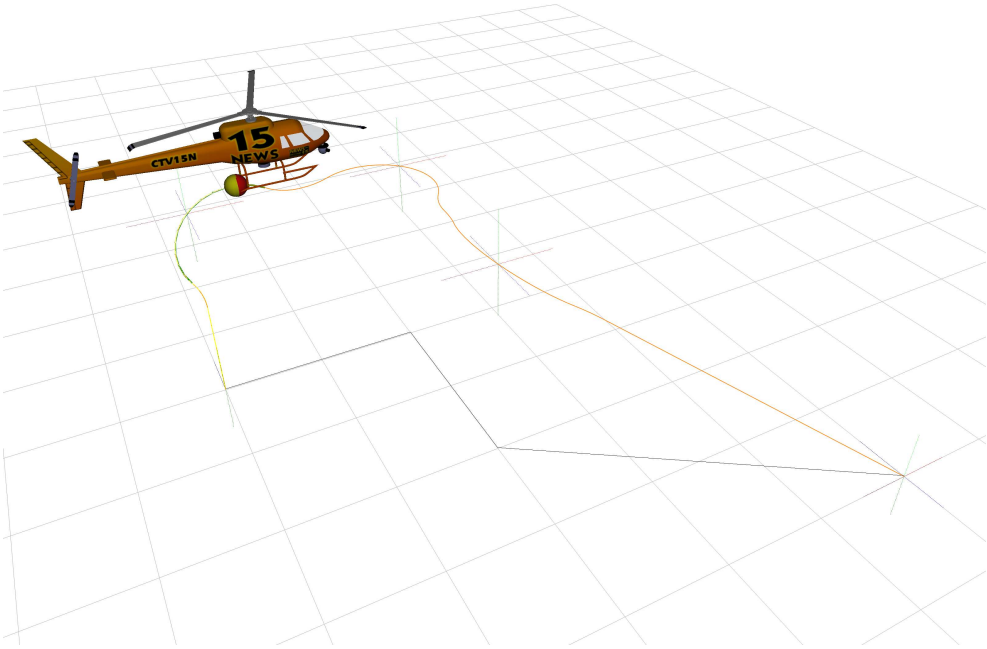


Figure 5. The curve shows the trajectory. Waypoints are projected onto the surface and plotted in line-of-sight.

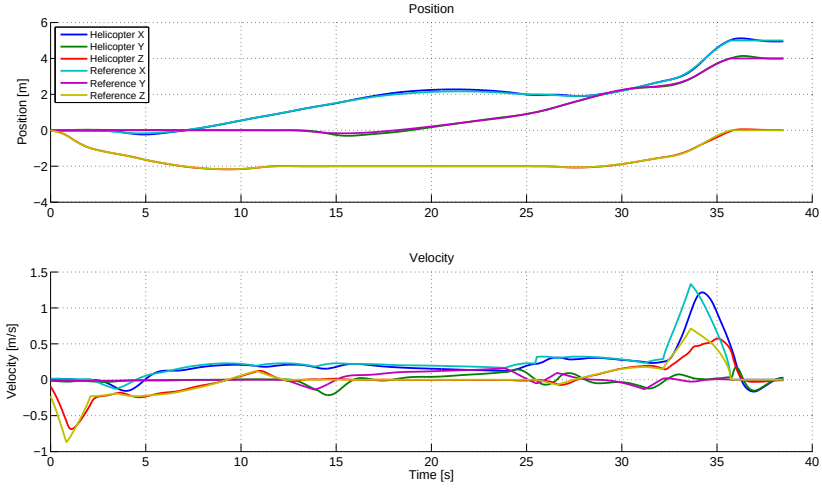


Figure 6. Helicopter states and references. Negative z-values correspond to an increase in height.

Mean [m]:	0.05
Max [m]:	0.14
Duration [s]:	36.26
Traj. Distance [m]:	10.49

Table 1. Mean and max values are the average and maximum distance of the helicopter to the trajectory, when its position is projected onto the trajectory. The table also shows duration of flight and length of trajectory

A test course has been constructed using the path planner. A trail run is performed on a test course containing several obstacles. A picture of the test course can be seen in Figure 7. There are three different sections in the environment. The starting position is located in a tunnel, from where it moves out in an open area and then into an s-shaped cave, that contains the target position. The cave requires tight manoeuvring to access. The helicopter starts off from hovering and moves in a direction opposite to its heading. A trajectory is generated using the described algorithm.

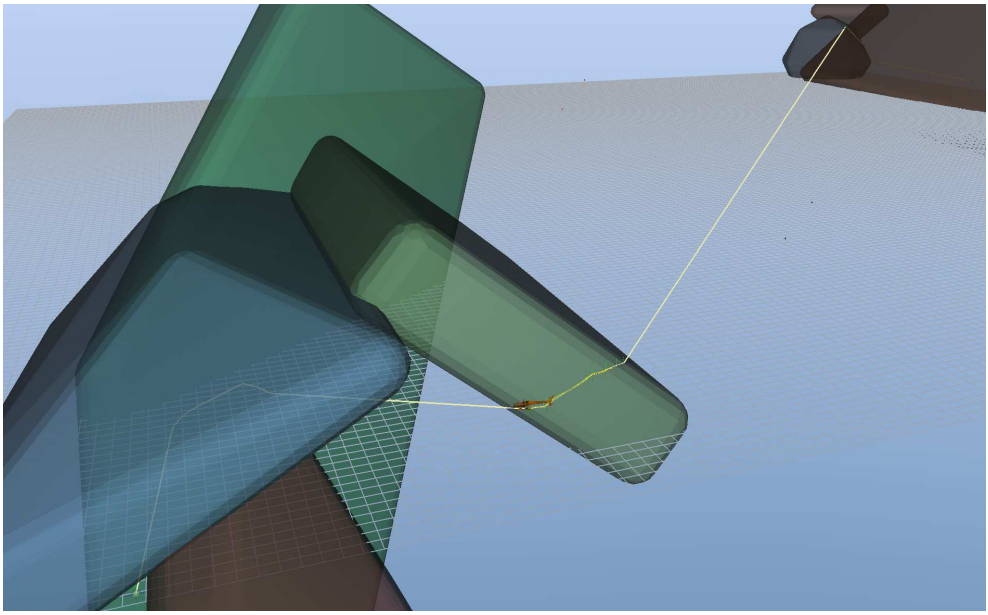


Figure 7. Side view of obstacle course. The curve shows the trajectory. Waypoints are projected onto the surface and plotted in line-of-sight.

Results from the simulation in Figure 8 show similar performance to the previous scenario. Table 2 shows that the vehicle has a maximum error of 0.12 m with an average trajectory error of 0.02 m. This shows a slight improvement to the previous case since a sharp turn rarely occurs in paths generated by the path planner. Also waypoints are often located on together near edges, results in lower speeds around obstacles.

IV. Conclusion

In this paper an approach for generating a trajectory for an aerial vehicle was described. The generated trajectory is continuously differentiable and it is assumed that the vehicle has the ability to turn in any direction with constant turning and pitch radius. A test showed that the trajectory generator was capable of generating collision free trajectories. In cases where sufficient space is available for turning, flight speed

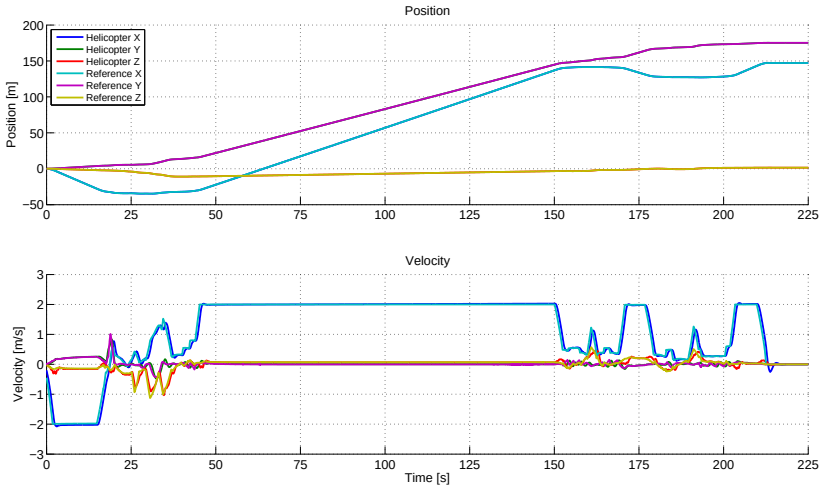


Figure 8. Helicopter states and references. Negative z-values correspond to an increase in height.

Mean [m]:	0.02
Max [m]:	0.12
Duration [s]:	213.02
Traj. Distance [m]:	318.32

Table 2. Mean and max values are the average and maximum distance of the helicopter to the trajectory, when its position is projected onto the trajectory. The table also shows duration of flight and length of trajectory

can be reduced. The approach used by the algorithm treats each waypoint separately. This ease the process of generating a trajectory, but some optimization is possible by evaluating multiple waypoints at once. This includes improvements to the location of the waypoint arc, and special handling of waypoints located on same vertex or edge of the workspace obstacle. It might not be possible to generate a trajectory from a path without reducing the turning radius and hence speed.

The generated trajectory was tested in a test environment, and flight was simulated using a helicopter model and a trajectory tracking controller. The helicopter was able to travel the trajectory without intersecting obstacles.

References

[1]D. H. Shim, *Autonomous exploration in unknown urban environments for unmanned aerial vehicles*, Technical report, University of California, Berkeley, 2005.

[2]Office of the Secretary of Defence, *Unmanned aircraft systems roadmap 2005-2030*, August 2005.

[3]M. Pachter and P. R. Chandler, *Challenges of Autonomous Control*, IEEE Control Systems Magazine, vol. 18, no. 4, August 1998.

[4]L. E. Dubins, *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents*, Amer. J. Math, vol. 79, 1957.

[5]E. P. Anderson, R. W. Beard, and T. W. McLain, *Real-Time Dynamic Trajectory Smoothing for Unmanned Air Vehicles*, Ieee Transactions on Control Systems Technology, vol. 13, no. 3, May 2005.

[6]E. P. Anderson and R. W. Beard, *An Algorithmic Implementation of Constrained Extremal Control for UAVs*, AIAA Guidance, Navigation, and Control Conference and Exhibit, Monterey, California, 2002, AIAA 2002-4470.

[7]G. Ambrosino, M. Ariola, U. Ciniglio, F. Corraro, A. Pironti and M. Virgilio, *Algorithms for 3D UAV Path Generation and Tracking*, 45th IEEE Conference on Decision & Control, 2006.

[8]T. I. Fossen, M. Breivik, and R. Skjetne, *Line-of-Sight Path Following of Underactuated Marine Craft*, Proceedings of the 6th IFAC MCMC, Girona, Spain, 2003, pp. 244-249.

[9]M. Bisgaard, *Modelling, estimation and control of helicopter slung load system*, PhD Thesis, Aalborg University, Department of Electronic Engineering, 2007.

[10]F. Schøler, *Three-Axis Motion Planning and Control of UAV*, Master Thesis, Aalborg University, Department of Electronic Engineering, 2008.

Paper F

State-Control Trajectory Generation for Helicopter in Obstacle-Filled Environment using Optimal Control

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

This paper was submitted to:
International Conference on Unmanned Aircraft Systems, 2012

State-Control Trajectory Generation for Helicopter in Obstacle-filled Environment using Optimal Control

Flemming Schøler and Anders la Cour-Harbo and Morten Bisgaard

***Abstract*— We address the challenge of flying a helicopter in a constrained 3D environment. Helicopter state and control trajectories are generated for the AAU Helisim model by formulating an optimal control problem (OCP) which is solved through the pseudospectral optimization tool, DIDO. We focus on achieving different tasks such as obstacle and terrain avoidance, flying through pipes and following existing paths through path constraints. These path constraints are evaluated through identified distance functions that can be used with DIDO. The trajectories are used off-line in a state feedback/feedforward control scheme to demonstrate simulated flight.**

I. Introduction

Small-scale autonomous helicopters are often used to perform different forms of surveillance and inspection tasks. This includes a recent interest in the use of unmanned aerial systems (UAS) in agriculture and forestry. Examples for agricultural purposes include weeds and disease identification, detection of water and nutrition shortages, and crop and plant growth monitoring. Examples for forestry purposes include timber cruising, disease and pest monitoring, environmental monitoring and compliance, nursery and tree planting growth development, early forest fires detection, search and rescue operations, accident investigation, etc. Whereas some of these tasks can be accomplished using a fixed-wing aircraft flying at a higher altitude, the ability to stay at the same spot or operate close to obstacles is often beneficial.

F. Schøler is with Institute of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, Denmark fls@es.aau.dk

A. la Cour-Harbo is with Institute of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, Denmark alc@es.aau.dk

M. Bisgaard is with Institute of Electronic Systems, Aalborg University, Fredrik Bajers Vej 7C, Denmark bisgaard@es.aau.dk

A. Previous work

The problem of generating trajectories for a UAS has been the subject of substantial research in the past years and a wide range of solutions to waypoint tracking, path planning, and obstacle avoidance have been presented. The wide range of approaches arises from the associated variety of design goals. A sparse sampling of methods for trajectory planning includes approaches that focus on constraints imposed by the environment such as the terrain avoidance [1], and constrains from the capabilities of the UAV such as turn radius [2]. Many consider the need to plan trajectories in an environment with obstacles and different approaches to this have been suggested, ranging from Lyapunov functions[3] to potential fields[4], visibility graphs[5], evolutionary algorithms[6], or Rapidly-exploring Random Trees[7]. In [8] the planning problem of flying a helicopter with a slung load through a known space with obstacles in a combination of minimum time and control effort and with a minimum of slung load oscillations is addressed. This planning problem is cast as a nonlinear optimal control problem (OCP) and state and control trajectories are generated using a pseudo-spectral method. The trajectories are used as references to a state feedback controller. Good results are obtained for different advanced maneuvers, but two preconditions are required 1) that a rough outline of the position trajectory is available as an initial guess, and 2) that state points along the position trajectory where the system is capable of achieving a trim condition are available. In this research we follow a similar setup where state and control trajectories are generated by formulating a nonlinear OCP that is solved using the pseudo-spectral method. However our individual tasks are all solved (with one exception) without initial

guesses, and since we fly without slung load, it is less complex to obtain trim conditions.

B. Present work

It is not straight-forward to generate an efficient and feasible trajectory when flying over varying terrain and amongst multiple obstacles. However, using the proposed method it is possible to design state and control trajectories specifically to the AAU Helisim model using the optimal solver DIDO. The subject of this paper is these trajectories on which a set of constraints are imposed. This will allow the helicopter to perform both basic and more challenging tasks while flying between different locations in a constrained environment.

An example of the basic task include tracking a pre-determined path, or keeping the altitude of the vehicle between certain minimum and maximum values measured relative to both ground level and mean-sea level. Other constraints that enable perhaps challenging tasks include flying inside a pipe, amongst trees, or amongst moving surfaces and obstacles.

Our aim is hence to enable the helicopter to fly to a sequence of goals while obeying a number of constraints imposed by the dynamic capabilities of the helicopter, the boundary conditions given by the goals, and path constraints imposed by the environment. In this paper we focus on identifying path constraints that are imposed on the helicopter position to avoid different types of obstacles in the environment. This includes time varying constraints.

In section II-A we briefly describe the helicopter model. The formulation of the OCP and the principles of solving it is presented in Section II-B. We present in Section II-C a means to decompose the trajectory into shorter parts with different path constraints. In Section II-D the general constraints are formulated in the OCP framework. Amongst the constraints we specifically focus on the path constraints in Section II-E. That is, for each task a set of distance functions indicate how well a specific task is performed. The final part of the OCP is the performance index presented in Section II-F. The generated state and control trajectories are used as reference to the feedback controller and for feedforward, respectively. The implementation of the controller is discussed in Section II-G. Simulation results are shown in Section III. This includes solutions to the OCP for a simulated flight through a test scenario for each task, and evaluation on how well tasks are handled at a given time.

II. Method

A. Helicopter Model

The process model is the AAU Helisim model[1], a first principles model of the Bergen industrial twin helicopter. It includes second order actuator dynamics and second order flapping dynamics for main rotor and stabilizer bar. Flapping is derived using blade element analysis and solved as steady state. Forces and torques for the main and tail rotor are also derived using blade element analysis. A uniform inflow model is used which is derived using momentum theory and it is solved using an analytical solution scheme. The rigid body model is derived using a redundant coordinate formulation based on Gauss' Principle of Least Constraint using the Udwadia-Kalaba equation. For the optimization process the model is simplified to 12 rigid body states (position and Euler angles and body velocities and angular rates) with steady state solutions for flapping and actuator dynamics. This is due to the fact that actuator dynamics, inflow dynamics, and flapping dynamics are of little interest from a control point of view and furthermore these states are faster than what is feasible to include in the controller considering the available computational resources. Included in the model is the capability to trim and linearize the helicopter model in any flight condition given by a speed, a sideslip angle, a yaw rotation rate, and an angle of attack to the flight plane. The model has four control signals. For a complete description of the model see Bisgaard [9].

B. Optimal Control Problem

The general idea of optimal control is to generate a control trajectory that takes the system from an initial condition to a final state while minimizing a performance index subject to a set of constraint. The challenge of finding a feasible trajectory for the helicopter can be formulated as

[1]The AAU Helisim model can be freely downloaded at www.UAVLab.org

such an OCP, where the performance index is dependent on the specific type of maneuvers.

In this paper a direct spectral algorithm known as DIDO is used for solving the optimization problem outlined below. It is not strictly dependent on an initial guess and has the distinct advantage over the majority of direct methods that a generated solution will satisfy necessary optimality conditions. This method discretizes the problem and approximates the states and control variables using Lagrange interpolating polynomials [10]. A presentation of the theory can be found in Bollino et al.[11].

The general optimal control problem is formulated as follows. Let $\mathbf{x}(t)$ and $\mathbf{u}(t)$ be state and control functions, respectively, and let

$$J(\mathbf{x},\mathbf{u},t) = E(\mathbf{x}_0,\mathbf{x}_\mathrm{f},t_0,t_f) + \int_{t_0}^{t_f} F(\mathbf{x}(\tau),\mathbf{u}(\tau),\tau)d\tau$$

be the performance index, where E is the boundary cost depending only on the first and last state $\mathbf{x}_0$ and $\mathbf{x}_\mathrm{f}$, and F is the running cost. The OCP is then to find that state-control pair that minimizes J subject to

$$\text{dynamic constraint} \quad \dot{\mathbf{x}}(t) = f(\mathbf{x},\mathbf{u},t) \quad (1)$$

$$\text{boundary conditions} \quad \mathbf{e}_\mathrm{l} \le e(\mathbf{x}_0,\mathbf{x}_\mathrm{f}) \le \mathbf{e}_\mathrm{u} \quad (2)$$

$$\text{path constraint} \quad \mathbf{g}_\mathrm{l} \le g(\mathbf{x},\mathbf{u},t) \le \mathbf{g}_\mathrm{u} \quad (3)$$

where f is the nonlinear helicopter model, e is the boundary condition function, and g is the path constraint function depending on the state and control functions as well as time. Both f, e, and g should be twice continuously differentiable, i.e. in $\mathbf{C}^2$, for better performance from DIDO. It should also be noted that the constraints (1) – (3) may only be satisfied at the collocation points. Even if all collocation points are inside accepted values a trajectory might still exceed constraints in-between collocation points. That is because a trajectory needs only fulfill constraints at the collocation points to be deemed optimal by the OCP formulation. While it is difficult to ensure that any generated trajectory is feasible everywhere, there are two approaches that seem to solve most issues 1) increasing the number of collocation points and 2) restricting constraints beyond strict necessity. In this research we use a combination of both, although the former approach tends to drastically increase computation time.

C. Completing tasks

We compose the flight of a series of shorter trajectories stitched together such as to create a seamless longer trajectory. To do so, we will assume that a general flight plan in the shape of a set of consecutive desired positions is available and that we are able to determine a set of state points along the flight plan in which the helicopter system is capable of achieving a trim condition, that is, zero-acceleration condition. We will denote these points 'trim points' (TP). We will also assume that consecutive TPs are chosen in such way that there exists a feasible trajectory between them. The optimal control problem is then solved between pairs of successive TPs, and we will denote such a state-control trajectory (TTT). This basically means that the overall flight trajectory will be fixed through a number of points, and that we will compute the optimal state and control trajectory that takes the helicopter from one TP to the next.

D. Constraints

The dynamic constraint in (1) is given by the helicopter model, but to determine the individual TTTs it is also necessary to identify the constraint functions e and g in (2) and (3), as well as the constraining constants $\mathbf{e}_\mathrm{l}, \mathbf{e}_\mathrm{u}$ and $\mathbf{g}_\mathrm{l}, \mathbf{g}_\mathrm{u}$. All TTTs will be constrained by the boundary condition that the system must be in a predefined trim. That is, the trajectory must start and end at the designated TPs.

E. Path Constraints

We focus mainly on identifying a number of task-dependent path constraint functions. These functions ensure that a TTT will fulfill certain tasks, such as, avoiding the obstacles, or staying near some path. The functions should be $\mathbf{C}^2$ which should give the best performance with DIDO. They should also be scaled properly and preferably monotone. However, not all properties will fit on all the distance functions described below. We identify functions that give distance to obstacles, terrain, pipes and paths.

In some cases, we will need to consider the physical size of the helicopter. A rough, but safe way to do this is to introduce a bounding sphere around the vehicle and ensure that this sphere avoids obstacles. The Bergen industrial twin helicopter has a bounding sphere radius of 1.7 meters.

1) Obstacle distances: The distance function for obstacles must not only determine if the vehicle is inside the obstacle, but also be a function with preferable a higher order differentiability that determines how close to intersection the vehicle is. It is difficult to identify such a function that can operate on obstacles shaped as polyhedrons. It can more easily be done if we represent each geometric obstacle by a bounding ellipsoid superset. We distinguish between work space (WS) ellipsoids and configuration space (CS) ellipsoids, where the latter are based on the Minkowski sum of the WS ellipsoid and the vehicle bounding sphere. This is done to transform the more difficult problem of finding a distance function that ensures that some vehicle shape does not intersect a WS obstacle, into the easier problem of ensuring that a point does not intersect the CS ellipsoid. The problems are not exactly dual, since the Minkowski sum of an ellipsoid and a sphere does not in general give an ellipsoid. Therefore the CS ellipsoid is chosen as the minimum volume tight ellipsoid containing this Minkowski sum. Thus, a solution found in CS will be valid in WS. The WS obstacle can be build from any set of vertices such as a point-cloud from sampled data. The described method works in N-dimensions spaces for N greater than one.

a) Work space ellipsoid: The WS bounding ellipsoid is the minimum volume ellipsoid that contains all points in a given cloud of vertices $\mathbf{v}_i \in R^n$. The bounding ellipsoid in 3D is uniquely defined by 9 extreme points. Determining which points to use analytically is an unsolved problem. However, finding the minimum volume bounding ellipsoid can be posed as a convex linear matrix inequalities problem. We let ε be an ellipsoid

$$\varepsilon_{ws} \equiv \mathbf{v}_i \mid (\mathbf{v}_i - \mathbf{a})^T \mathbf{Q} (\mathbf{v}_i - \mathbf{a}) \leq 1, \qquad (4)$$

where $\mathbf{Q} \in R^{N \times N}$ is its symmetric shape matrix and $a \in R^N$ is the origin, such that $\varepsilon_{ws} \supseteq \text{conv}(\mathbf{v})$. Since collision with any point of an obstacle must be avoided, all vertices of an obstacle must be within the ellipsoid. That is, all vertices of an obstacle in WS must be within the unit sphere after being translated by $\mathbf{a}$, and rotated and scaled by a symmetric matrix $\mathbf{Q}$, denoted the shape matrix. Since the volume of ε_{ws} is proportional to $(\det \mathbf{Q})^{-1/2}$, minimizing $\log \det \mathbf{Q}^{-1}$ is equivalent to minimizing the volume of ε_{ws}. Hence by solving (4), the minimum volume ellipsoid is obtained, centered at the origin, that contains the points $\mathbf{v}_1, ..., \mathbf{v}_L$. This convex problem can be solved in polynomial-time in theory, and in practice very efficiently using interior-point algorithms, see [12] for details.

b) Configuration space ellipsoid: A CS obstacle O_{cs} can be generated from the Minkowski sum of its WS ellipsoid ε_{ws} and the vehicle bounding sphere ε_v. Though this geometric sum is generally not an ellipsoid, it can be tightly approximated by the parameterized families of external ellipsoids ε_{cs}^+ such that $\varepsilon_{cs}^+ \supseteq O_{cs} = \varepsilon_{ws} \oplus \varepsilon_v$. If fact, this family of external ellipsoids inherits both center and principal axes from the workspace ellipsoid ε_{ws}. In principle any ellipsoid from this family of external ellipsoids could be used for the CS ellipsoid, but identifying the minimum-volume ellipsoid would give us the largest free configuration space.

In order to find the minimum-volume tight ellipsoid in CS we first need to decompose the shape matrix into a rotation matrix, containing the principal axes, and a diagonal scale matrix, containing its equatorial radii. This decomposition is performed because the volume of an ellipsoid is a function of its equatorial radii. We also need an expression for the shape matrix of ε_{cs}^+. This can be found as a linear combination of the shape matrix of each ellipsoid in the sum. It is known from [13] that the shape matrix of the external ellipsoid along some direction $\mathbf{l} \in R^n$, where the geometric sum and tight ellipsoid touch, is generated by the geometric sum of two ellipsoids with shape matrix $\mathbf{Q}_1$ and $\mathbf{Q}_2$, and is

$$\mathbf{Q}_l^+ = \left[\underbrace{\sqrt{\langle \mathbf{l}, \mathbf{Q}_1 \mathbf{l}\rangle}}_{k} + \sqrt{\langle \mathbf{l}, \mathbf{Q}_2 \mathbf{l}\rangle}\right] \times \left[\frac{\mathbf{Q}_1}{\sqrt{\langle \mathbf{l}, \mathbf{Q}_1 \mathbf{l}\rangle}} + \frac{\mathbf{Q}_2}{\sqrt{\langle \mathbf{l}, \mathbf{Q}_2 \mathbf{l}\rangle}}\right]$$

where k gives the radius of an ellipsoid with shape matrix $\mathbf{Q}_1$ along some direction $\mathbf{l}$.

Using ε_{ws} for $\mathbf{Q}_1$ and the sphere ε_v for $\mathbf{Q}_2$ the shape matrix of the external ellipsoid of an obstacle and vehicle can be simplified to

$$\mathbf{Q}_l^+ = (k + r)(\mathbf{Q}_1/k + r\ \mathbf{I}) \tag{5}$$

since the vehicle has radius r in any direction. It can be shown from (5) that the external ellipsoid will take the same semi-axes and center as the obstacle ellipsoid ε_{ws}. There exist a tight ellipsoid for any direction of vector $\mathbf{l}$, but we are only concerned with a direction that results in a minimum volume. Because the semi-axes of the external ellipsoid match that of the obstacle ellipsoid, this translates to finding the radius k where minimum volume occurs. Separating the rotation and scaling components of the shape matrix makes this easier. Eigendecomposition can be used to transform the shape matrix $\mathbf{Q}_1$ into the product of an orthonormal rotation matrix $\mathbf{R}$ composed by eigenvectors, and a diagonal matrix $\mathbf{\Lambda}$ with eigenvalues along the diagonal, such that $\mathbf{Q}_1 = \mathbf{R}\ \mathbf{\Lambda}\ \mathbf{R}^T$.

By this alignment of the CS ellipsoid with the semi-axis of the WS ellipsoid, (5) is rewritten, so we must find a diagonal scale matrix

$$\mathbf{R}\ \mathbf{Q}_l^+ = (1 + r/k)(\mathbf{I}\ k\ r + \mathbf{\Lambda}) \tag{6}$$

that minimizes volume. The volume of an ellipsoid is proportional to the product of its eigenvalues. In our case this is the diagonal elements of $\mathbf{Q}_l^+$.

The volume V reaches an optimum when its derivative with respect to k is zero

$$V'(k) = \underbrace{\frac{2\pi^{N/2}}{N\Gamma(N/2)}}_{a} \underbrace{r\left(\frac{r}{k} + 1\right)^N}_{b} \underbrace{\prod_{i=1}^{N}(k\ r + \lambda_i)}_{c} \times \underbrace{\left[\sum_{i=1}^{N}\frac{1}{k\ r + \lambda_i} - \frac{N}{k\ r + k^2}\right]}_{d},$$

where Γ is the Gamma function.

The first term a is a positive constant with regards to k and can be disregarded. It is also apparent from b that V' is zero when r is zero or $r = -k$. Obviously, none of these solutions are feasible since both r and k must be positive, so these terms are also disregarded. The product-term c is always positive, since also eigenvalues are positive and this term is also disregarded. The remaining terms in d is a polynomial fraction that has a numerator polynomial with exactly one positive root (at least for values of N up to 4). One may assert this by applying Descartes' rule of signs. The positive root can be substituted into (6) to finally obtain a volume minimizing shape matrix.

c) Distance function: Once a minimum volume shape matrix is found, the distance function becomes

$$d_\varepsilon(\mathbf{w}) = \|\mathbf{R}\ \mathbf{Q}_l^{+T}(\mathbf{w} - \mathbf{q})\| \tag{7}$$

for N-dimensional obstacles ($N > 1$) that are in $\mathbf{C}^2$. Because of the non-uniform scaling of $\mathbf{Q}_l^+$ this distance is not Euclidean, but any boundary point will still satisfy $d_\varepsilon(\mathbf{w}) = 1$ while a point outside satisfies $d_\varepsilon(\mathbf{w}) > 1$, which is exactly what is needed for solving the OCP.

2) Terrain distance: A requirement exist that the vehicle position $\mathbf{w}$ must stay within a defined distance d above ground level (AGL) Let p be a function that gives the height of the surface over the point $\mathbf{w}$. Then

$$d(\mathbf{w}) = \mathbf{w}_z - p(\mathbf{w}_x, \mathbf{w}_y, t)\ . \tag{8}$$

Typically, a 2D regular grid digital elevation model (DEM) is used to represent the heights at discrete samples of a continuous surface. While such a model only contain elevation information at each grid point, an elevation at any point can be estimated through grid-interpolation. We seek a $\mathbf{C}^2$ function p that gives an estimate of the surface height at $\mathbf{w}$ using vertices in the grid near $\mathbf{w}$. We choose a 16-point bicubic interpolation function for p as this method well-approximates a wide range of surfaces [14], while being fast to calculate and produces a $\mathbf{C}^2$ surface. Bicubic interpolation is the lowest order 2D interpolation procedure that maintains the continuity of the function and its first

derivatives (both normal and tangential) across cell boundaries [15].

Suppose the function values p and the derivatives p_x, p_y, and p_{xy} are known at the four corner vertices $(0,0)$, $(1,0)$, $(0,1)$, and $(1,1)$ of the unit square in a 2D Cartesian grid. In a 2D Cartesian grid the elements are unit squares, and the vertices are integer points. The interpolated surface at the square can then be found using bicubic interpolation

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{c}_{ij} x^i y^j \ ,$$

where $\mathbf{c} \in R^{n\times n}$ is some constant.

Bicubic interpolation can be done by subsequent interpolation on each axis. For a single axis, interpolation is done on the unit interval (0,1), between a starting point a_0 at $t=0$ and an ending point a_1 at $t=1$ with starting tangent m_0 at $t=0$ and ending tangent m_1 at $t=1$, the polynomial can be defined by

$$\begin{aligned} p_s(t) = (2t^3 - 3t^2 + 1)a_0 + (t^3 - 2t^2 + t)m_0 \\ + (-2t^3 + 3t^2)a_1 + (t^3 - t^2)m_1 \end{aligned}$$

where tangents are found by centered differences (instead of exact derivatives) using neighboring grid points

$$\begin{aligned} m_0 &= (a_1 - a_{-1})/2 \\ m_1 &= (a_2 - a_0)/2 \end{aligned}$$

such that interpolation on one axis is done with the four grid points nearest the vehicle. Extending to bicubic interpolation, the 16 surrounding grid vertices a are needed. These can be found in the set of all grid vertices g in a Cartesian grid by truncating the vehicle position. For $i,j = -1, 0, \ldots, 2$

$$a_{(i,j)}(\mathbf{w}) = g_{(\mathrm{trunc}(\mathbf{w}_x)+i, \mathrm{trunc}(\mathbf{w}_y)+j)} \ .$$

The interpolation is first applied four times in the x direction to obtain each of the four points needed for the final interpolation in the y direction. The interpolation parameters t are given by the fractional part of the vehicle position. The surface height at position $\mathbf{w}$ then becomes

$$p(\mathbf{w}) = p_s\left(\mathrm{frac}(\mathbf{w}_y), \begin{bmatrix} p_s\left(\mathrm{frac}(\mathbf{w}_x), a_{(\cdot,-1)}\right) \\ p_s\left(\mathrm{frac}(\mathbf{w}_x), a_{(\cdot,0)}\right) \\ p_s\left(\mathrm{frac}(\mathbf{w}_x), a_{(\cdot,1)}\right) \\ p_s\left(\mathrm{frac}(\mathbf{w}_x), a_{(\cdot,2)}\right) \end{bmatrix}\right)$$

3) Path/pipe distance: The distance from the vehicle to a path or pipe, consisting of n segments between consecutive vertices $\mathbf{s}_i$, $i = 0, 1, \ldots n$, is given as the minimum distance to any point on any segment is the $\mathbf{C}^1$ function

$$d(\mathbf{w}) = \min_{i\in(0,n-2)} \frac{\|\mathbf{s}_i + (\mathbf{s}_{i+1} - \mathbf{s}_i)t(i) - \mathbf{w}\|}{r(i,t(i))} \ ,$$

where the closest point to one segment is

$$t(i) = \max\left[0, \min\left(1, \frac{(\mathbf{w} - \mathbf{s}_i)\cdot(\mathbf{s}_{i+1} - \mathbf{s}_i)}{(\mathbf{s}_{i+1} - \mathbf{s}_i)\cdot(\mathbf{s}_{i+1} - \mathbf{s}_i)}\right)\right]$$

and $r(i,t(i))$ gives the radius at a particular segment i at time $t(i)$. The function $r(i,t(i))$ is 1 when finding distance to paths, such that $d(\mathbf{w})$ express Euclidean distance. For pipes we might want the vehicle to stay inside the pipe ($d(\mathbf{w}) < 1$) or outside ($d(\mathbf{w}) > 1$).

4) Geodesic path distance: A geodesic path is here a $\mathbf{C}^1$ path composed by segments of different types. The path segments alternates between straight line segments and curved segments, where curved segments might be arcs or helices with an angular span of less than π. Schøler et al. [16] present a method that finds such optimal paths.

Section II-E.3 describes a distance function for the straight segments. On curved segments, convex optimization is used for identifying the point on the segment closest to $\mathbf{w}$. Finding the global minimum amongst all segments can be done as

$$d(\mathbf{w}) = \min_{i\in(0,n-1)} \|p_i(\mathbf{w}) - \mathbf{w}\|$$

F. Performance index

Determining a performance index is a compromise between time-optimality and how aggressively the helicopter is driven. We use the complete flight time as the boundary cost. The running cost is a combination of changes in control input and yaw offset $\mathbf{x}_{\text{yaw}}$ (from flight direction). The running costs are weighted by a scalar k and a diagonal matrix $\mathbf{M}$

$$\begin{aligned} E(\mathbf{x}_0, \mathbf{x}_\mathrm{f}, t_0, t_f) &= t_f \\ F(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau) &= \frac{d}{d\tau}\mathbf{u}(\tau)^T \mathbf{M} \frac{d}{d\tau}\mathbf{u}(\tau) \\ &\quad + k\mathbf{x}_{\text{yaw}}(\tau)^2 \end{aligned}$$

The reason for this is that while a pure time optimization might produce unnecessary large and

rapidly changing control signals, the other extreme might produce unreasonable slow movement. We also try to enforce that helicopter orientation should be in direction of flight. The reason we need to do this is because flying sideways or backwards have virtually no influence on the time it takes to fly a trajectory.

G. Closed-loop trajectory tracking

Once a state and control trajectory has been computed we want the helicopter system to actually follow these references. Since the OCP presented is too complex to be solved in real time, it is at present not possible to run the optimization in real time such as to facilitate a model predictive controller implementation. The major drawback of the off-line computation is the inability to adapt on the fly to changes in the flight conditions. Instead, we rely on the pre-computed control-state trajectory pair and use the feedforward/feedback control strategy shown in Fig. 1 to keep the system close to the state trajectory at all times. Although feedforward of the control trajectory will cause the helicopter to diverge from the state trajectory, the pre-computed control trajectory will be fairly close to the actual control signal, and the feedback controller will only have to make small corrections. This results in smaller state errors and, hence, better trajectory tracking. The feedback controller

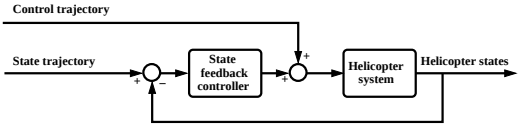


Fig. 1. Closed-loop trajectory tracking. The state and control trajectories are respectively used as reference to the feedback controller and for feedforward.

is designed using linear quadratic regulation (LQR) on the helicopter system in hover. This is treated in more detail in Bisgaard [9].

III. Results

The application of the OCP formulation to the challenge of designing an optimal state-control trajectory pair will be demonstrated through simulated flight. The example mission plans a trajectory that takes the helicopter through 9 TPs by stitching together 8 TTTs as discussed in Section II-C. The sideslip angles, angles of attack, and yaw rates are zero in all TPs. Different combinations of path constraints are used for the TTTs, while no changes are made to the performance index. The actual trajectory are found with the controller described in Section II-G. Results from the full planned and actual trajectory is discussed in Section III-I. Before this, each TTT is treated in details in Section III-A – Fig. III-H. Each section has picture that show a 3D view of each TTT. Here the gray spheres represent TPs interconnected by the calculated trajectory, with arrows showing the collocation points. According to type of path constraints used we also show one of three types of plots; altitude, obstacle, or distance plots. The altitude plots show surface height below the helicopter as a function of time. In all plots blue curves show results of distance functions from simulated flights. These usually cover red curves that show results of distance functions from calculated trajectories, but values at collocation points can still be seen as red dots. The collocation points have constraints that place them inside the green area. As discussed in Section II-B, distance functions might give values outside this range in-between collocation points. This does not necessarily mean that a trajectory is infeasible, since constraints can be inflated to avoid such issues. For altitude plots, a curve below the green area shows the location of the actual surface which obviously must not be intersected by either trajectory. All TTTs except TTT 3 are calculated without an initial guess.

A. TTT 1: avoiding a hill

The objective is to keep a safe distance to the terrain and staying inside a certain altitude AGL range. The path constraint is given from (8). The path is between two via points on opposite sides of a hill. The distance is ≈500 meter in beeline. Fig. 2 shows the setup and results of path constraints. The helicopter starts off from hover and must reach the target TP with a forward velocity of 3.5 m/s. The solution is a compromise between going directly above the hill and going completely around it. As the helicopter passes the hill, it stays near the surface. The helicopter slows down to make a final sharp turn to meet the target

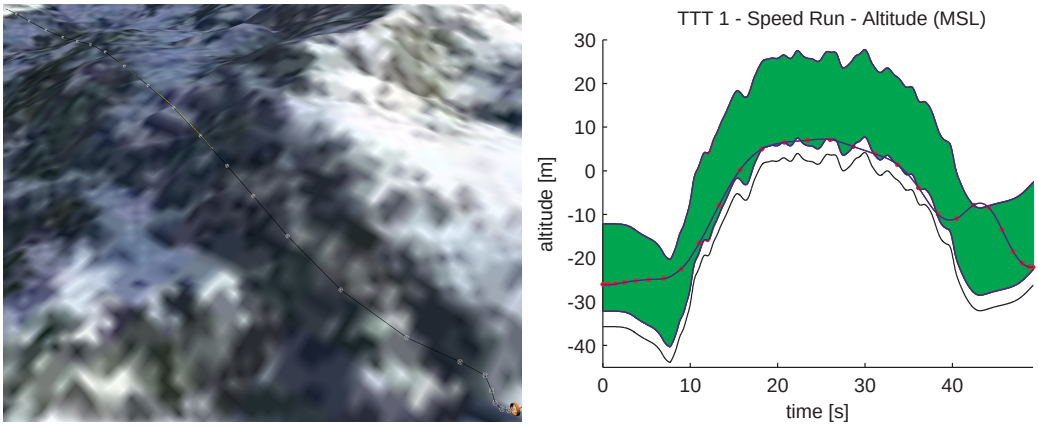


Fig. 2. Setup and altitude for actual state trajectory at TTT 1

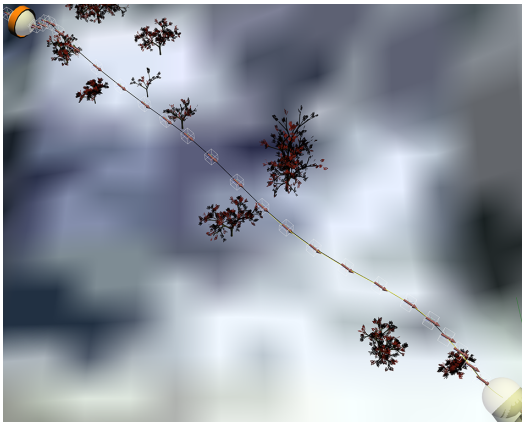
TP. This is done because of the performance index that penalizes an orientation different from the direction of flight. All collocation points are inside accepted values, the both calculated and simulated trajectories are above ground.

B. TTT 2: avoiding trees

The helicopter must reach its TP while avoiding the 9 trees and staying above the ground. We use a procedural method to generate realistically looking trees of different shape and size. When calculating distance to the trees, each tree is split into two bounding volumes. We find good consistency in approximating the crown of the tree by a bounding ellipsoid and the trunk by a bounding capsule. The bounding ellipsoid is generated from the vertices of the crown, using the method described in Section II-E.1. In order to not include points that are already inside the crown WS bounding ellipsoid, we only add vertices that have at least one edge with an end-point outside this ellipsoid. The radius of the capsule is determined by the maximum horizontal distance to the center of the capsule. We use the distance function from Section II-E.3 with r being the sum of the capsule radius and vehicle bounding sphere radius. The setup and result of path constraints on trees can be seen in Fig. 3. A low maximum allowed altitude removes the possibility of simple solutions, such as simply flying above all obstacles. We use (8) to include surface constraints. The result of the distance function is shown in the plot. As the helicopter approaches and departs from a bounding volume, each curve takes appearance as a second degree polynomial. The helicopter reaches a volume surface every time a distance curve reaches 1. There is some minor constraint violations in-between some collocation points. If necessary, these could be avoided by slightly increasing the vehicle bounding sphere radius.

C. TTT 3: fly through pipe

The helicopter must fly through a pipe. The pipe is shaped by a cubic spline from 9 control points that are points intersected by the spline. The first and last control point match the location of the TPs, and the pipe has the vehicle bounding sphere radius at these points. The remaining control points are chosen to give a challenging path. For these points, radius is twice the vehicle bounding sphere radius. Each section between a pair of control points of the spline consists of 15 line-segments, such that we can use the distance function in Section II-E.3. The results of this distance function and overall setup can be seen in Fig. 4. At this TTT the solver tends to come up with solutions that violate the pipe, although all collocation points are valid. The solution was to provide an initial guess to the solver. An initial guess was constructed by first placing a TP at each control point and generating a trajectory for each subsequent TP-pair. At each TP the velocity was set to zero,



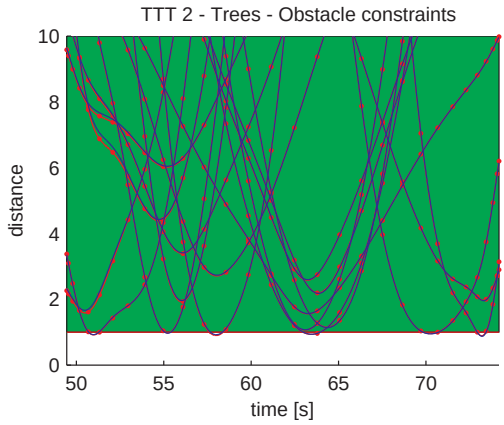
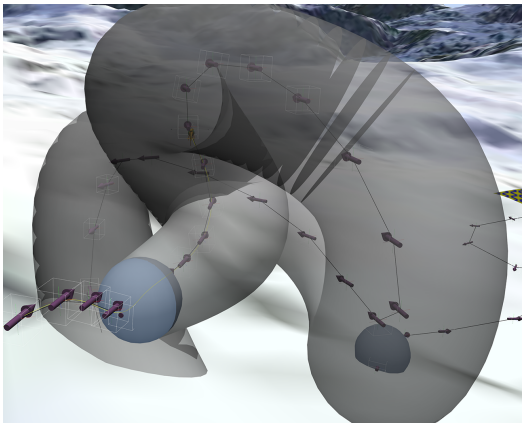


Fig. 3. Setup and output of distance function for actual state trajectory at TTT 2



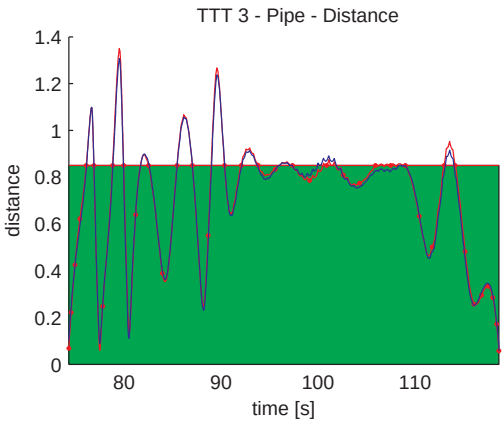


Fig. 4. Setup and output of distance function for actual state trajectory at TTT 3

which requires the helicopter to pass through each control point and hover there. These trajectories were then stitched together to form an initial guess. Although all collocation points are in green area, we would have further restrict the upper limit or increase the number to collocation points, to keep the helicopter at a safe distance from the pipe surface. The small fluctuations visible around 100 seconds might suggest increasing the number of segments per division.

D. TTT 4: rotate 360 deg

The helicopter must perform a full yaw rotation and descend 10 meters, while staying inside an altitude span of 5 meters. The setup and development in altitude can be seen in Fig. 5. The path constraint is given from (8). The increase of altitude and decrease x-velocity after 125 seconds seen in Fig. 10 is an effect of the performance index. It shows a compromise between flying as fast as possible and facing direction of flight.

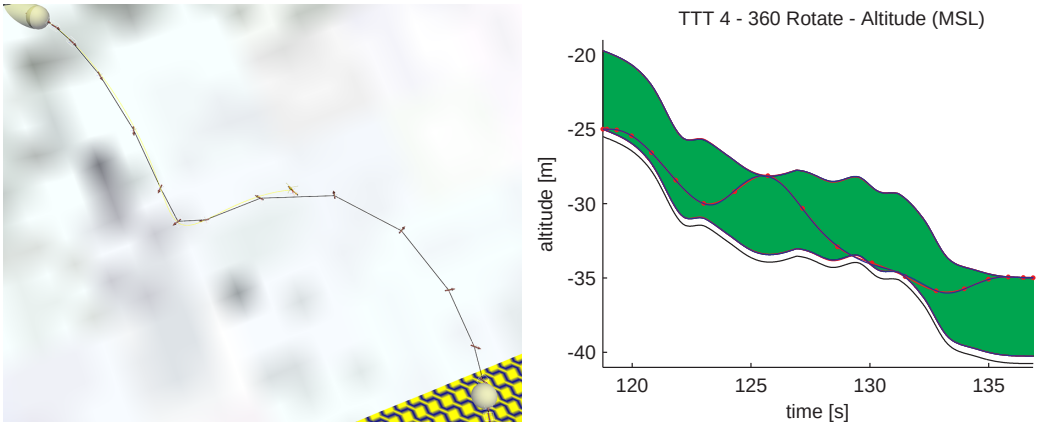


Fig. 5. Setup and altitude for actual state trajectory at TTT 4

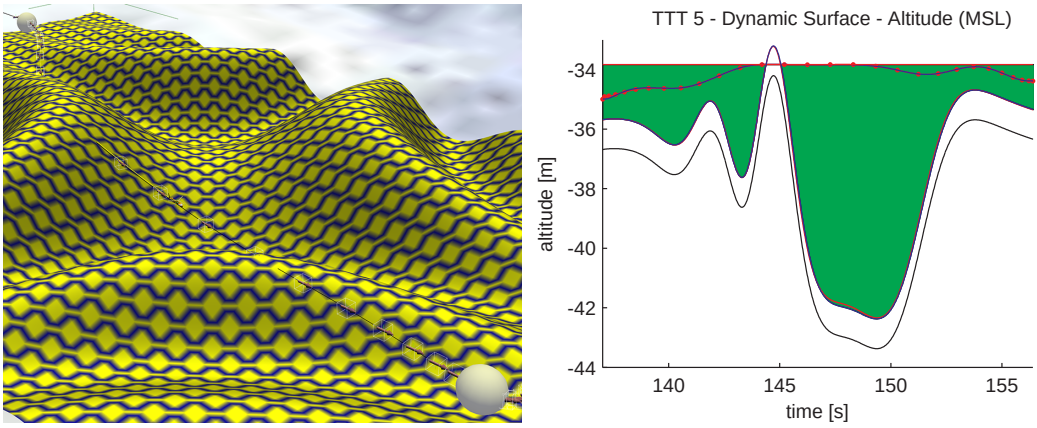


Fig. 6. Setup and altitude for actual state trajectory at TTT 5

E. TTT 5: fly above a dynamic surface

The helicopter starts from hover and must fly to its target TP, while above a dynamic surface. Both initial and target TP are collision free during a full periodic cycle, but there is no collision free feasible path between the TPs at any single time instance. The surface is smooth with a periodic movement that is generated by a cosine function. It is shown in Fig. 6 along with the result of the distance function. We use (8) for distance function but instead of a DEM, we use an explicit function to determine surface at a given (x, y)-position and time. Effective use of velocity makes it possible for the helicopter to fly nearly in beeline between TPs. We see that although the green area is separated at 145 seconds, all collocation points are still inside, and the helicopter does not intersect the lower surface curve.

F. TTT 6: avoid moving obstacles

The helicopter must fly through a chaotic environment with moving obstacles. Realistic obstacle trajectories are generated using the Newton Dynamics engine. Obstacles bounce against each other and the surface. The obstacles are shown in Fig. 7 along with the result of the distance

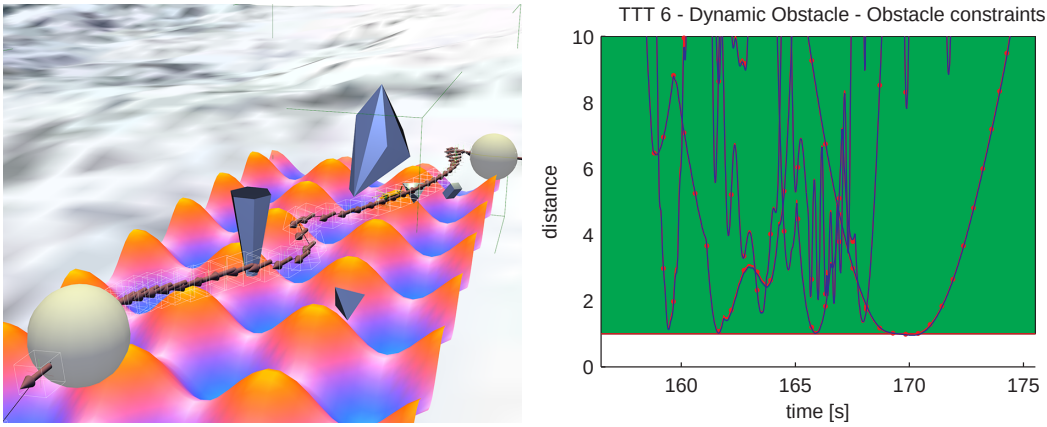


Fig. 7. Setup and output of distance function for actual state trajectory at TTT 6

function. A high number of collocation points are needed because of the obstacles move fast relative to the helicopter. We use (7) for the distance function, which gives an output that is quite different from the case with static obstacles. Curves for rotating non-spherical bounding volumes have a superimposed sinusoidal function, and are not continuously differentiable at points where obstacles collide. Despite this, all collocation points are still inside the green area, and the helicopter does not intersect any obstacles.

G. TTT 7: rotate 180

The helicopter must perform a half yaw rotation and descend 19 meters. The setup and development in altitude can be seen in Fig. 5. The path constraint is given from (8). The path takes shape as a half circle. It is less expensive to fly with a high velocity and facing the direction of flight, than the bee-line that would have two sharp turns.

H. TTT 8: follow an existing path

The helicopter must follow an existing geodesic path around a building. The path is generated using the method described in Section II-E.4 that also specifies the distance function. The results of this distance function and overall setup can be seen in Fig. 9. Here we wish each collocation point to be located close to the geodesic path. This allows following a minimum-length path, while optimizing the route according to the performance index. The discrepancy between the calculated and simulated trajectory and seems much larger than previous cases, though quite small considering the scale of y-axis. Both trajectories follow the geodesic path quite well, as the maximum distance between trajectory and path is ≈ 0.25 meters.

I. Full trajectory

The full planned and actual state trajectories through all 9 TPs are shown in Fig. 10. While the position and attitude plots show good tracking, the derived states show some difficulties a few places. Most notably are the oscillations on pitch after 50 and 180 seconds, that both occur while the helicopter is descending rapidly. These are seen more clearly on Fig. 11. This suggests the need for more focus on dampening pitch oscillation in the future control design and does not constitute an issue with the planned trajectory. There are also some tracking errors when the helicopter is exposed to high accelerations. This is i.e. visible from the drift on y-axis at the high initial forward acceleration.

IV. Discussion

We have presented an approach to generating optimal state and control trajectories for the Bergen industrial twin helicopter. The applied method is based on a formulation of an optimal control problem (OCP) with time and control effort

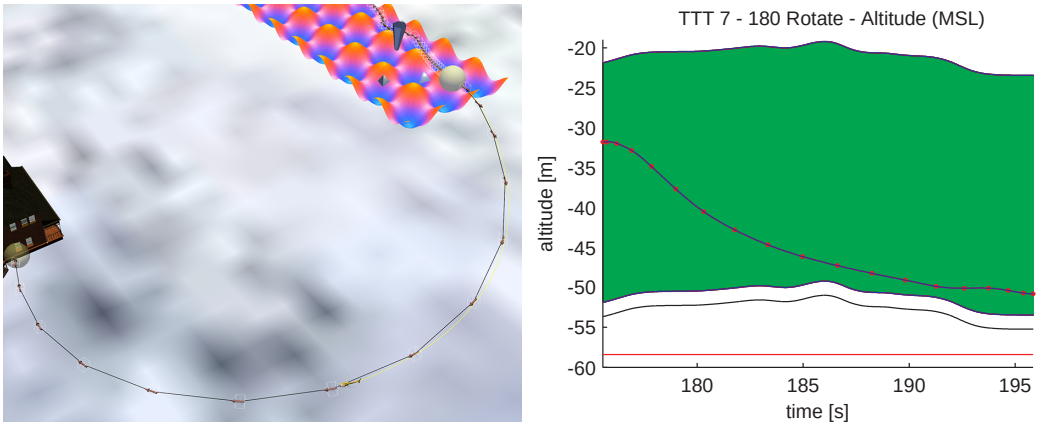


Fig. 8. Setup and altitude for actual state trajectory at TTT 7

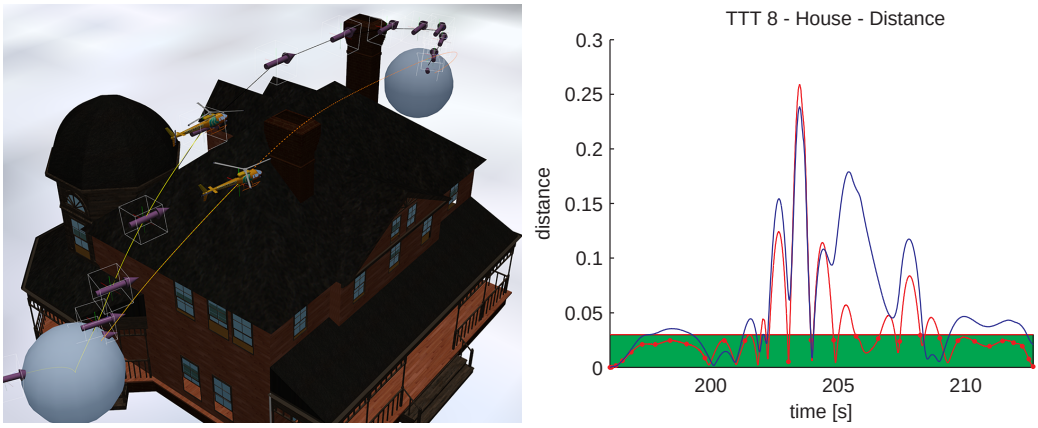


Fig. 9. Setup and output of distance function for actual state trajectory at TTT 8

as the primary components in the performance index. The AAU Helisim helicopter model was used and constraints in the environment were added as boundary and path constraints. The optimal solver DIDO was used to obtain a solution to the OCP. The solution was found in form of a state-control trajectory pair. The state trajectory was used as reference to a feedback controller and control trajectory for feedforward. Unfortunately the OCP presented is too complex to be solved in real time with DIDO, and runs off-line. It was shown that a range of different tasks could be completed satisfactory by introducing the appropriate path constraints in the OCP.

ACKNOWLEDGMENTS

Part of the work on this paper has been carried out while F. Schøler visited the Mechanical & Aerospace Engineering Department at Naval Postgraduate School. F. Schøler thanks Professor Michael Ross and Dr Mark Karpenko for hosting his stay and for valuable discussions.

REFERENCES

[1] Rahim, M. and Malaek, S. M., *Intelligent operation using terrain following flight in unmanned aerial vehicles*, In Proceedings of IEEE Aerospace Conference, 2007.

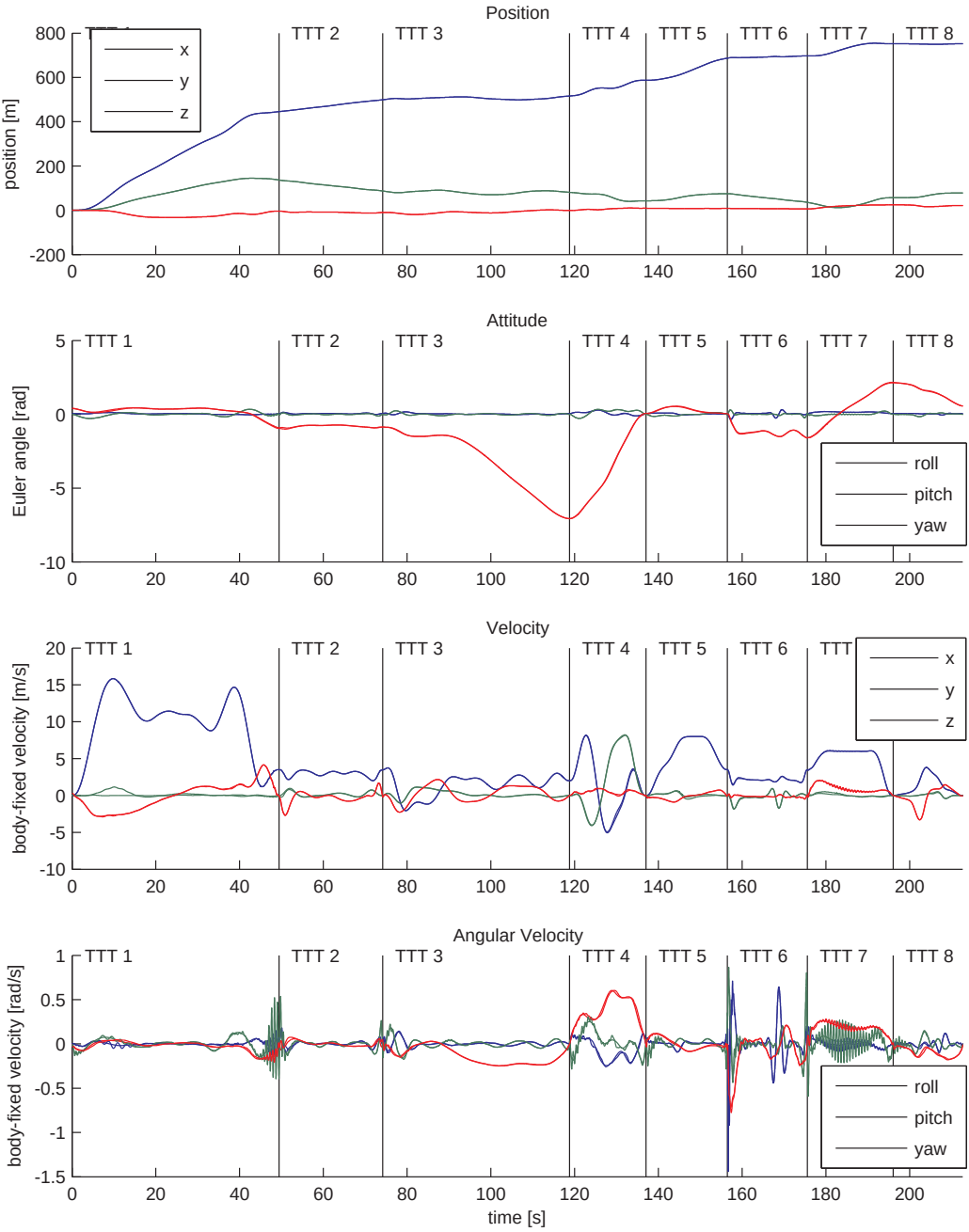


Fig. 10. Planned and actual state trajectory

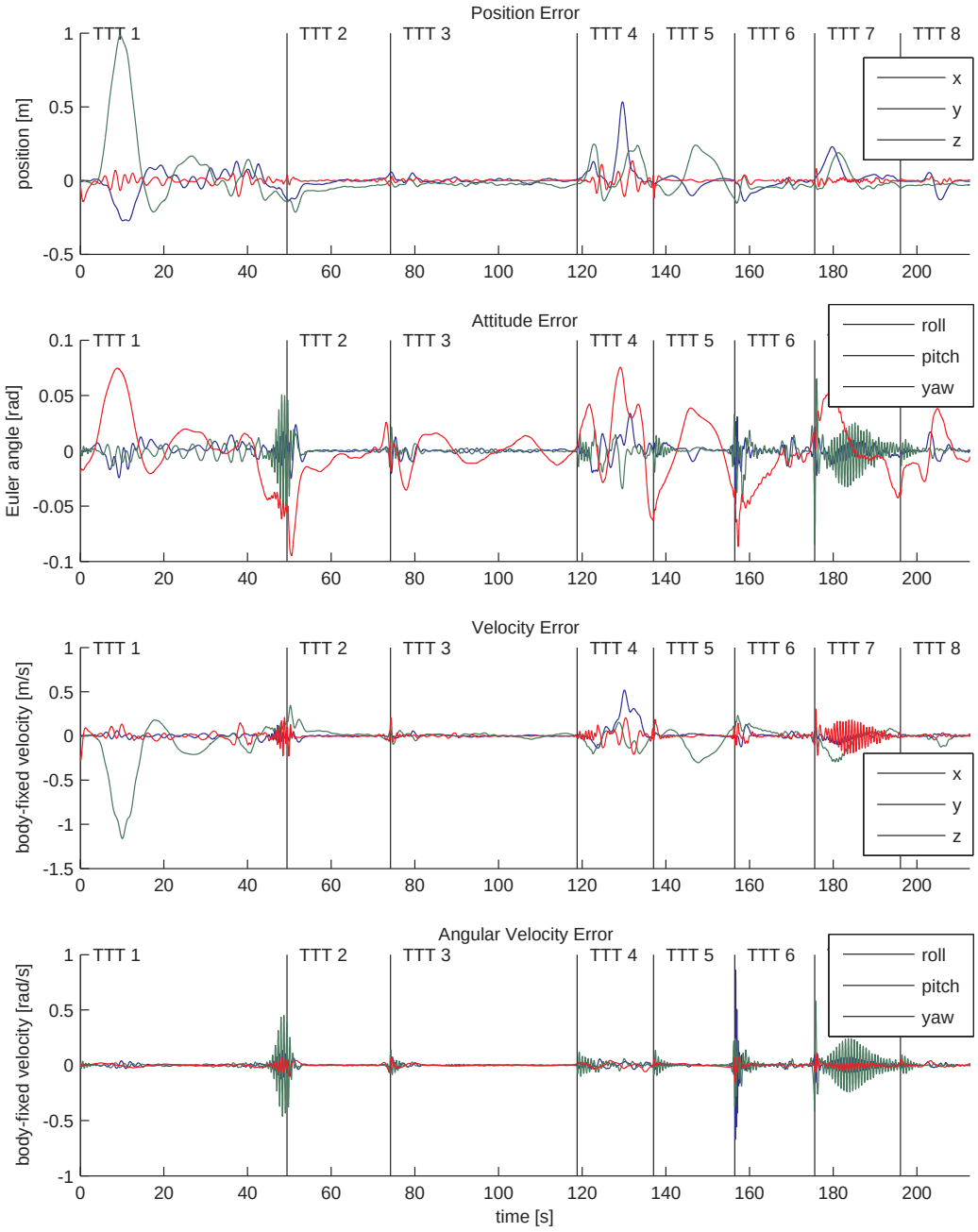


Fig. 11. Error between planned and actual state trajectory

[2] Ambrosino, G., Ariola, M., Ciniglio, U., Corraro, F., Pironti, A., and Virgilio, M., *Algorithms for 3D UAV Path Generation and Tracking*, Proceedings of the 45th IEEE Conference on Decision & Control, 2006.

[3] Frazzoli, E., Dahleh, M. A., and Feron, E., *Real-Time Motion Planning For Agile Autonomous Vehicles*, AIAA Conference on Guidance, Navigation and Control, 2000.

[4] Sigurd, K. and How, J., *UAV Trajectory Design Using Total Field Collision Avoidance*, AIAA Conference on Guidance, Navigation and Control, 2003.

[5] Schøler, F., la Cour-Harbo, A., and Bisgaard, M., *Generating Configuration Spaces and Visibility Graphs from a Geometric Workspace for UAV Path Planning*, Springer, Submitted, 2011.

[6] Rubio, J. C., Vagners, J., and Rysdyk, R., *Adaptive Path Planning for Autonomous UAV Oceanic Search Missions*, AIAA 1st Intelligent Systems Technical Conferencen, 2004.

[7] LaValle, S. and Kuffner Jr, J., *Randomized kinodynamic planning*, The International Journal of Robotics Research, 2001.

[8] la Cour-Harbo, A. and Bisgaard, M., *State-Control Trajectory Generation for Helicopter Slung Load System using Optimal Control*, Proc. of AIAA Guidance, Navigation, and Control, 2009.

[9] Bisgaard, M., *Modeling, Estimation, and Control of Helicopter Slung Load System*, Aalborg University, 2007, PhD Thesis.

[10] Ross, I. M., *User's Manual for DIDO: A MATLAB Application Package for Solving Optimal Control Problems, Tomlab Optimization*, Feb 2004.

[11] Bollino, K. P., Lewis, L. R., Sekhavat, P., and Ross, I. M., "Pseudospectral Optimal Control: A Clear Road for Autonomous Intelligent Path Planning," *Proc. of AIAA- Infotech@Aerospace Conference and Exhibit*, 2007.

[12] Boyd, S., Ghaoui, L. E., Feron, E., and Balakrishnan, V., *Linear Matrix Inequalities in System and Control Theory*, Society for Industrial and Applied Mathematics, 1994.

[13] Boyd, S. and Vandenberghe, L., *Convex Optimization*, Cambridge University Press, 2004.

[14] Kidner, D., Dorey, M., and Smith, D., *What's the point? Interpolation and extrapolation with a regular grid DEM*, GeoComputation 99, 1999.

[15] Russell, W. S., *Polynomial Interpolation Schemes for Internal Derivative Distributions on Structured Grids*, Applied Numerical Mathematics, Vol. 17, pp. 129-171, 1995.

[16] Schøler, F., la Cour-Harbo, A., and Bisgaard, M., *3D Path planning with geodesics on geometric workspaces*, Aalborg University, 2011.