International Micro Air Vehicle Conference and Flight Competition (IMAV2013)
17-20 September 2013, Toulouse, France

MAV Belief Space Planning In 3D Environments With Visual Bearing Observations

Laurie N. Bose[1]*and Dr. Arthur G. Richards[1]†

University of Bristol, United Kingdom
lb7943@bristol.ac.uk

Abstract

This paper examines the problem of path planning for a MAV in complex 3D environments without the use of GPS. Instead the MAV must rely on using sensor measurement of its surroundings for localisation. The quality of localisation and uncertainty in the MAV's estimated state will be determined by the path it takes through the environment. In order to minimize the probability of failure this state uncertainty must be incorporated into the planning process determining the path of the MAV. We present a path planner that can be used to produce paths which attempt to minimize the MAV's state uncertainty when operating in complex 3D environments. Simulation results are presented for a MAV equipped with a limited field of view camera sensor in environments featuring fixed localisation beacons from which, bearing measurements can be obtained and used for localisation.

1 Introduction

An autonomous vehicle operating in a GPS denied environment must rely on sensor measurements of its surroundings for localisation within the environment. The quality and availability of such measurement may vary greatly across the environment with some areas allowing for far more accurate localisation than others. The path the vehicle takes through the environment determines the measurements received by its sensors and hence its ability to maintain good localisation with low uncertainty in its estimated state. The vehicle is less likely to be able to accurately follow paths which involve high state uncertainty which results in a higher probability of experiencing a collision or other form of potential failure. It becomes necessary to account for this uncertainty in the vehicle's estimated state when conducting path planning in order to produce paths which do not have a high probability of such failures occurring.

Belief space path planners are designed to account for uncertainty in vehicle state when generating paths. They produce paths providing a trade off between reducing the time taken and reducing the vehicle's state uncertainty. One such planner, the Belief Road map [6], is based on a modified version of the Probabilistic Road map [5]. Simulated results demonstrated the planners ability to produce paths minimizing uncertainty by travelling within close proximity to beacons which the vehicle could use for localisation. The Rapidly-exploring Random Belief Trees algorithm presented in [7] iteratively constructs a graph in belief space. This graph is then used to determine a safe path for the vehicle. Simulated results showed the vehicle's path deviate in order to pass through state measurement areas reducing the state uncertainty and enabling safely passage through narrow areas further along the path. The Particle RRT algorithm [8] accounts for uncertainty by stochastic simulating each tree expansion multiple times with process noise, adding multiple new branches per expansion.

The belief space planner introduced in this paper is geared towards navigation for MAV class vehicles equipped with camera type sensors, operating in complex 3D environments featuring many obstacles. Fixed beacons within these environments are used for localisation. The vehicle is able to take relative bearing measurements to a beacon provided it is both within the limited field of view of the vehicle's camera and not occluded by any obstacle. This setup is used as a rough representation of a MAV using a visual SLAM system to navigate through an already mapped area (with the beacons representing visual landmarks within the SLAM systems map). One of the long term goals of this work is to produce a belief space planner that fully utilizes a visual SLAM system for autonomous navigation.

*PhD Student, Department of Aerospace Engineering
†Senior Lecturer, Department of Aerospace Engineering

The planner operates on a graph which is determined by the construction of an Octree partitioning of the environment. The Octree's nodes are used to determine the positions of vertices within the graph. This process as discussed in [10], [9], [12], [13] results in an efcient vertex layout for path planning with sparse vertices in large open areas and dense clusters of vertices close to obstacles. Graph edges are then added between pairs of vertices that are within line of sight (LOS) of one another and whose associated Octree nodes are in contact. This step involves the expensive test of determining if two points are in LOS of one another by checking if the line segment connecting the two intersects any obstacle. The computational cost of this step is reduced by using the partitioning Octree to determine a subset of obstacles with which a connecting line segment could potentially collide. The planning algorithm involves incrementally constructing potential paths through the graph from an initial starting vertex. The construction of a path involves conducting a simulation of the MAV attempting to follow that path. This simulated MAV uses a particle filter for localisation and state estimation (as discussed in [4], [3], [2]). This particle filter captures the growth of uncertainty in the MAV's estimated state as it performs actions along with uncertainty reduction due to sensor measurements. At the end of the simulation the path is evaluated based upon both its length and in terms of minimizing uncertainty in the MAV's state estimation. Potential paths are then chosen to be extended or rejected based on this evaluation.

Section 2 introduces the problem formulation and representation used for the MAV's environment. Section 3 describes the construction of the graph which the planner operates within. Section 4 describes the full planning algorithm. Section 5 then presents a set of simulated results where paths produced by the planner attempting to maintain low state uncertainty are compared to paths which attempt to minimize distance traveled.

2 Problem Formulation

The path planning problem involves finding a path from an initial position x_{start} to a destination x_{goal} which provides some desired trade off between minimizing distance traveled and maintaining low state uncertainty. The environment in which path planning is conducted is represented by a set of n convex obstacles S (non convex obstacles can be represented by decomposing into multiple convex) and a set of m fixed localisation beacons $B = \{b_1, ...b_m\}$ where $b_i \in B$ is the position of the ith beacon. The set $S = \{s_1, ...s_n\}$ consists of the convex polygon meshes representing each obstacle. For $s_i \in S$ we denote the set of vertices of the mesh $s_i.X$, the set of mesh edges $s_i.E$ and the set of mesh faces by $s_i.F$.

The MAV is taken to be equipped with a forward facing, limited field of view sensor which can take bearing measurements to the beacons of B. A measurement to a beacon $b_i \in B$ can be made provided b_i is both within the sensors field of view and is not obstructed by any obstacle in S. It is assumed that the MAV has access to a complete map of the environment. It can thus use the bearing measurements obtained from the beacons of B for localisation within this map. The map also gives knowledge of the obstacles in S allowing the MAV to account for them when path planning.

A graph across the environment is constructed as described in Section 3 consisting of a set of vertex positions V (including x_{start} and x_{goal}) and a set of undirected edges E dictating how the vertices are connected to one another. Potential paths through this graph are then represented by path nodes. A path node p has an associated vertex $p.x \in V$, a parent path node $p.p \in P$, current path length $p.l$ and an associated set of particle states $p.S$. Recursively iterating back through the parents of a path node p and examining their associated vertices traces out the whole path represented by node p. The set of particle states $p.S$ represents the belief of the MAV's true state upon travelling p's path and reaching $p.x$. Using a set of particle states to represent a probability distribution or "belief" for a true state is a standard technique used in Monte Carlo (particle filtering) localisation [2], [3], [8]. Path nodes also have a weighting $p.w$ which is generally based off both the node's path length $p.l$ and the uncertainty of its state belief represented by $p.S$, these two factors can be weighted differently based on the desired type of path (for example minimum distance paths would have zero weighting on uncertainty).

The planning problem then consists of finding a path through the graph connecting the vertex x_{start} to that of x_{goal} and which provides the desired trade off between distance and state uncertainty. The state uncertainty of the MAV upon reaching x_{goal} is dependent upon the path it traversed and the bearing measurements it took along the way which it used for localisation. The planner used to find such paths by searching the constructed graph is described in section 4.

Algorithm 1 Splitting Criteria for an Octree node

```
If node depth < max node depth
    For All $s_i \in S$
        For All edges $e \in s_i.E$
            If node intersects $e$
                return true
            End If
        End For
    End For
End If
return false
```

3 Graph Construction

This section describes the algorithm used to construct the graph on which the path planner operates.

3.1 Determining Graph Vertices

The environment is first partitioned via the construction of an Octree whose nodes are then used to determine the locations of graph vertices in V. The Octree is initialized so that its initial node encompasses the entire volume in which path planning is to be conducted. Each node is then recursively subdivided into eight equally sized octants provided the node furfills a specific splitting criteria given in Algorithm 1. This splitting criteria produces a partition as shown in Figure 1a which has high node density at obstacle edges required for pathing around them but which ignores flat surfaces. During construction each node of the Octree also stores which obstacles from S it intersects with in order to facilitate determining edges between the vertices of V in a later graph construction step.

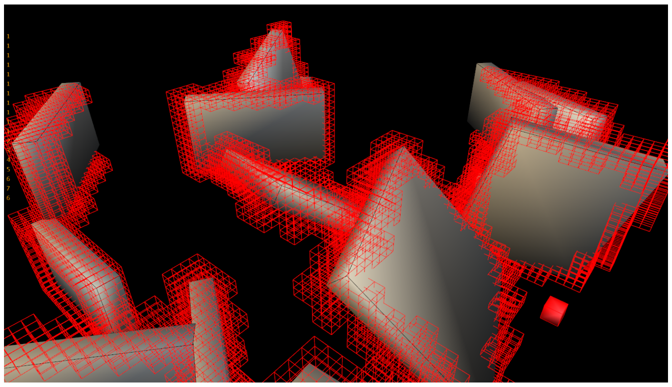
After construction of this Octree vertices are added to V at the centres of each of the Octree nodes (provided the node's centre is not inside any obstacle from S). Finally vertices are added at the initial start and goal positions $V = V \cup \{x_{start}, x_{goal}\}$. Note that the max node depth of the Octree in Algorithm 1 determines the level of detail to which the graph vertices V are placed throughout the environment. This affects the possible routes through the environment represented in the final graph. Important routes may not be represented if the max node depth is too low.

3.2 Determining Graph Edges

It is assumed that if any two graph vertices $x_A, x_B \in V$ are within direct line of sight then a safe path between them exists along the line connecting the two. This can be represented in the graph by the addition of an edge (x_A, x_B) to the set of edges E. However connecting every pair of vertices that are in line of sight leads to a huge amount of edges. Instead edges are added to E connecting pairs of vertices that are both within line of sight and whose associated Octree nodes are in contact with one another. This results in a graph of the form as shown in Figure 1b.

Determining if two positions x_A, x_B are within line of sight of each other requires checking that the line segment connecting the two does not intersect with any obstacle in S. A naive approach to performing this check is to simply check the line segment for intersection against each obstacle of S in turn until an intersection is detected or all obstacles have been checked against. However this method results in the number of intersection checks required simply growing with the number of obstacles present.

A preferable method involves using the constructed Octree to only perform intersection checks against a subset of the total set of obstacles. From the earlier graph construction in section 3.1, each Octree node stores a subset of obstacles with which it intersects. The method then consists of traversing the nodes of the Octree which the line segment passes through one by one and only performing intersection checks with obstacles that are contained within the subset of obstacles for


(a) The deepest nodes in a partitioning Octree showing how the splitting method biases the node density about obstacle edges

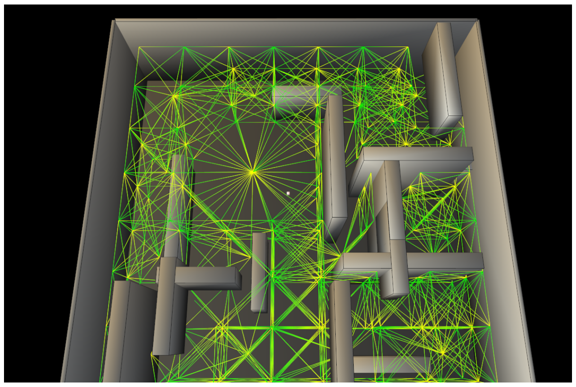

(b) Example of a graph constructed by the process described in Section 3. The partitioning Octree constructed with a low max node depth for clarity.

Figure 1

one of these nodes. Traversing between the cells of the tree is generally a relatively cheap process especially in open areas which can be partitioned by a few large cells. This method generally results in far fewer intersection checks being performed and resulting in large performance gains.

3.3 Intersection Checks

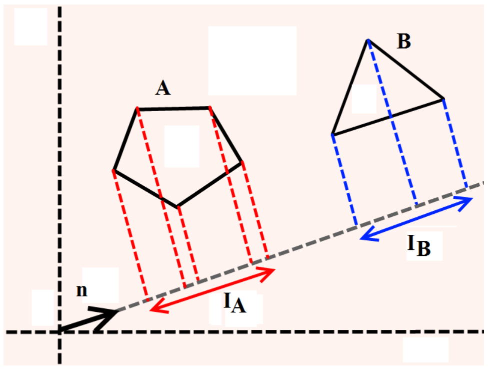
When constructing the Octree in Section 3.1 it is necessary to check for intersection between the cuboid nodes of the Octree and the convex obstacles of S. Intersection checks between convex objects, such as these are conducted with a method employing the separating axis theorem [11]. This simply states that if a plane can be placed between two convex objects such that each is fully contained on a different side of that plane then the objects are not intersecting. The normal to a such a separating plane is called a separating axis.

To determine if a specific direction $\hat{n}$ is a separating axis for two convex objects A and B, the projection of each of the convex objects onto $\hat{n}$ is calculated. This forms the projection intervals for each object I_A and I_B. If these intervals do not overlap then it can be seen that $\hat{n}$ is a separating axis for the object and hence they do not intersect. Figure 2 shows a 2D equivalent example of this.

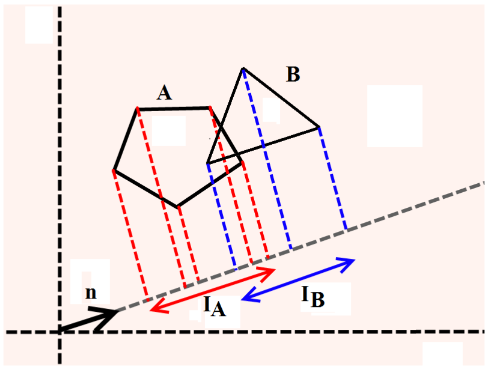
The interval formed from projecting a convex polygon mesh such as $s_i \in S$ onto a direction $\hat{n}$ is determined by calculating the dot product of $\hat{n}$ with each of the mesh's vertices $x \in s_i.X$. This then forms the set $D = \{x \bullet \hat{n} \mid x \in s_i.X\}$ and the interval formed by the projection is simply $I = [min(D), max(D)]$. If the convex mesh consists of a single edge (the object is a line segment) between two points p_1 and p_2 then its projection interval for a direction $\hat{n}$ is just $I = [min(D), max(D)]$ where $D = \{\hat{n} \bullet p_1, \hat{n} \bullet p_2\}$.

In order to fully determine if two convex meshes intersect, a number of directions must be checked to see if any provide a separating axis. This set of directions consists of the directions normal to the faces on each of the meshes along with all possible directions formed by taking the cross product between two edges, one from each mesh. Without checking the directions formed by the cross product of edges, separating axes such as those shown in Figure 2c would be overlooked. As soon as any of these directions are found to provide a separating axis the objects have been determined to not be intersecting and no further checks are necessary. On the other hand if no separating axis has been found after all these directions have been checked then the objects do intersect.

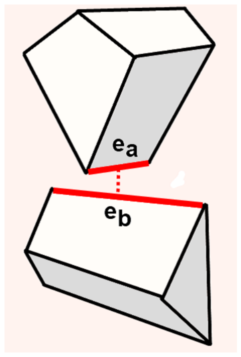
Note if a specific direction $\hat{n}$ has been checked to determine whether or not it is a separating axis it is unnecessary to check any other direction parallel to $\hat{n}$. For meshes such as those of the cuboid Octree nodes there will be faces whose normal direction is parallel to that of another face and edges parallel to others edges. It is important therefore to keep track of what directions have already been checked in order to avoid unnecessarily checking a direction parallel to one previous.



(a) Here the intervals I_A and I_B do not intersect indicating direction n is a separating axis for objects A and B and hence they do not intersect



(b) Here the intervals I_A and I_B intersect hence direction n is a not a separating axis however this alone is not enough to determine if A and B are truly intersecting



(c) Two objects with a separating axis shown by the dotted line whose direction is formed by the cross product of edges e_a and e_b

Figure 2: 2D Visualization of using vector projection to determine if a direction is a separating axis for two convex meshes

4 Path Planner

Paths that feature high levels of uncertainty in the MAV's estimated state are generally undesirable as the MAV will be less likely to accurately follow such paths. This may then result in a collision or other form of failure due to the MAV straying from the desired path. The path planning method in this section attempts to find paths through the graph constructed in Section 3 which provide some desired trade off between maintaining low state uncertainty and minimizing distance traveled.

In order to evaluate a potential path the planner numerically simulates a MAV attempting to follow the path from vertex to vertex using a particle filter and bearing measurements to the beacons of B for localisation. The sample variance and covariance of the particle filter's set of particles is calculated upon reaching each vertex and used to determine the current uncertainty in the MAV's estimated state. This uncertainty along with the distance traveled so far, is then be used to evaluate the path upon reaching a vertex. If the path is deemed inferior to an already existing path to the current vertex then the simulation is halted and the path discarded.

4.1 Algorithm

The planner searches for paths through the visibility graph constructed in Section 3 consisting of the set of vertices V and edges E. A set of path nodes P (as described in section 2) is used to store different paths through the vertices of the graph. $U \subseteq P$ denotes the set of path nodes which should be used in attempting to create new paths by extending their current path to another vertex.

The set of path nodes P is initialized by setting $P = \{P_0\}$ where the path node P_0 is at the starting vertex ($P_0.v = x_{start} \in V$) and has a set of particle states $P_0.S$ representing the initial belief of the MAV's state. The parent of this node is simply initialized as $P_0.p = P_0$ as it is the path node from which all others originate. U is then initialized as $U = \{P_0\}$.

The planning algorithm used to generate paths for this setup is listed in Algorithm 2. The algorithm iteratively creates new paths through the graph from the path nodes in U. This involves selecting a path node $p \in U$ and attempting to extend p's existing path to every vertex $v \in V$ which is connected to $p.x$ by an edge from E (every vertex v such that $(v, p.x) \in E$).

For each path extension from $p.x$ to some other vertex $v \in V$ the MAV is simulated attempting to travel from $p.x$ to v taking $p.S$ as the initial belief of the MAV's state. The result of this simulation is then used to form a new path node q representing the extended path. This path node is also assigned a weighting $q.w$ based on both its uncertainty and path

Algorithm 2 Planning method

```
While $U \neq \emptyset$
    $p =$minimum scoring element of $U$
    For all $v \in V \mid (v, p.x) \in E$
        $q = Propagate(p, v)$
        $q.l = Assign\_weighting(q)$
        If $!(\exists a \in P \mid (q.w < a.w \wedge q.x = a.x))$
            $P = P \cup p_{test}$
            $Insert(p_{test})$
        End If
    End For
    $U = U \setminus p$
End While
```

length, each of which, are weighted by different constants (w_{len} and w_{un}) depending on the desired type of path. If this new path node is then deemed to provide the current best path to its vertex $q.x$ then it is added to both P and U.

After all possible extensions of a path node have been attempted, it is removed from U. Once U no longer contains any path nodes there are no more potential paths to investigate and the algorithm terminates.

There are a number of functions defined in Algorithm 2 which are now explained in greater detail.

4.1.1 $Propagate(p, v)$

The function $Propagate(p, v)$ takes a path node $p \in P$ and a graph vertex $v \in V$ and returns a new path node q such that $q.x = v$. The function carries out a numerical simulation of the MAV travelling from vertex $p.x$ to vertex v using a particle filter for localisation taking $p.S$ as the initial set of particles. At each step of this simulation the subset of beacons in B that are within line of sight of the MAV (if any) and which can be brought within the MAV's limited field of view sensor is determined. These beacons are then further examined to determine which would provide a bearing measurement resulting in the greatest decrease in state uncertainty, the MAV is then made to face this beacon. The set of particles on the returned path node $q.S$ is that of the particle filter at the end of the simulation.

4.1.2 $Assign\_weighting(p)$

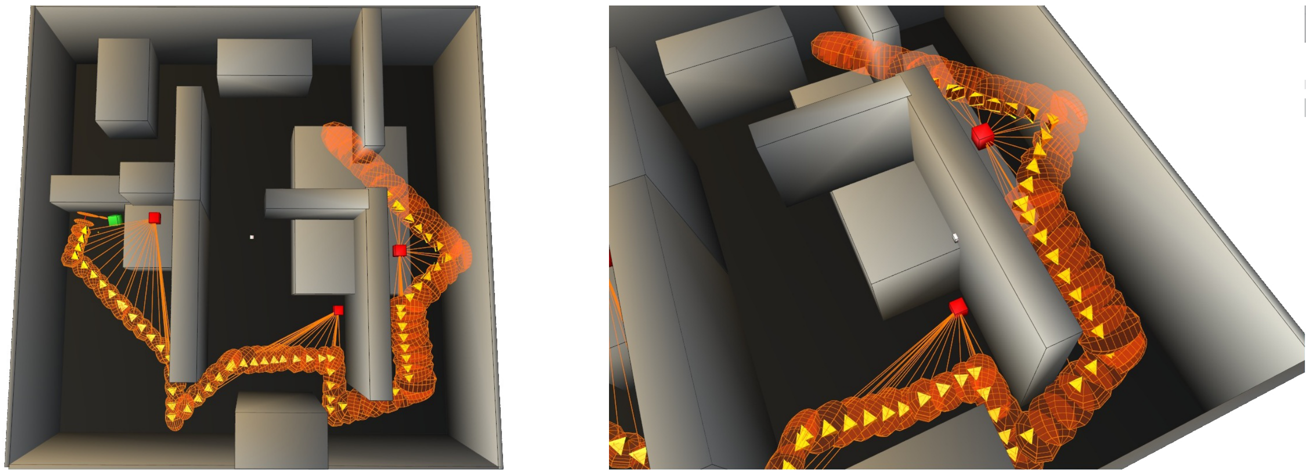
This function assigns a weighting to a path node p based on both it's path length $p.l$ and an uncertainty measure u associated with its set of particle states $p.S$. In the set of results given for the planner, this uncertainty measure u was simply taken to be the sum of the x,y,z sample variances in the positions of the particle states in $p.S$. The path nodes weighting is then assigned as $p.w = w_{len} \times p.l + w_{un} \times u$ where w_{len} and w_{un} are constants that can be adjusted depending upon the desired type of path. For example setting $w_{un} = 0$ would assign weightings based purely on path length resulting in the planner attempting to produce minimum distance paths. On the other hand, setting w_{un} to a value much greater than w_{len} results in the planner producing paths which are far longer but which maintain much lower state uncertainty by taking measurements to the localisation beacons of B.

4.1.3 $Insert(p)$

The function $Insert(p)$ inserts a path node $p \in P$ into the list of path nodes to update U. The position at which p is added to U is determined by its assigned weighting $p.w$ such that U maintains a list of path nodes ordered by their weightings. Low weighted path nodes in U can then be expanded first by simply choosing the last element of U for expansion. If we were instead not to order U, or simply select which path node to expand at random, it would lead to an extremely large number of cases whereby a new path would be found that was slightly preferable to an existing path but which would then itself, be replaced by another slightly preferable path found soon after. This would result in a great amount of time wasted creating and examining similar paths which are likely to be replaced.

5 Results

A number of paths produced by the planner in different environments are now presented in this section (each environment is fully enclosed, however the roof of each is not drawn). The uncertainty of the MAV's estimated state is visualized by error ellipsoids formed from the set of particle states of the path nodes as discussed in Section 2. In each setup, robust paths produced by the planner when attempting to maintain low uncertainty (drawn in orange using weighting constants $w_{len} = 0.01$, $w_{un} = 1$) are compared with the paths produced when only minimizing distance (drawn in blue using weighting constants $w_{len} = 1$, $w_{un} = 0$). localisation beacons are drawn as red markers and measurements taken of them at points along a path are indicated by lines and vision cones.


(a) Path produced by planning algorithm through complex environment
(b) Path of (a) shown from a different angle

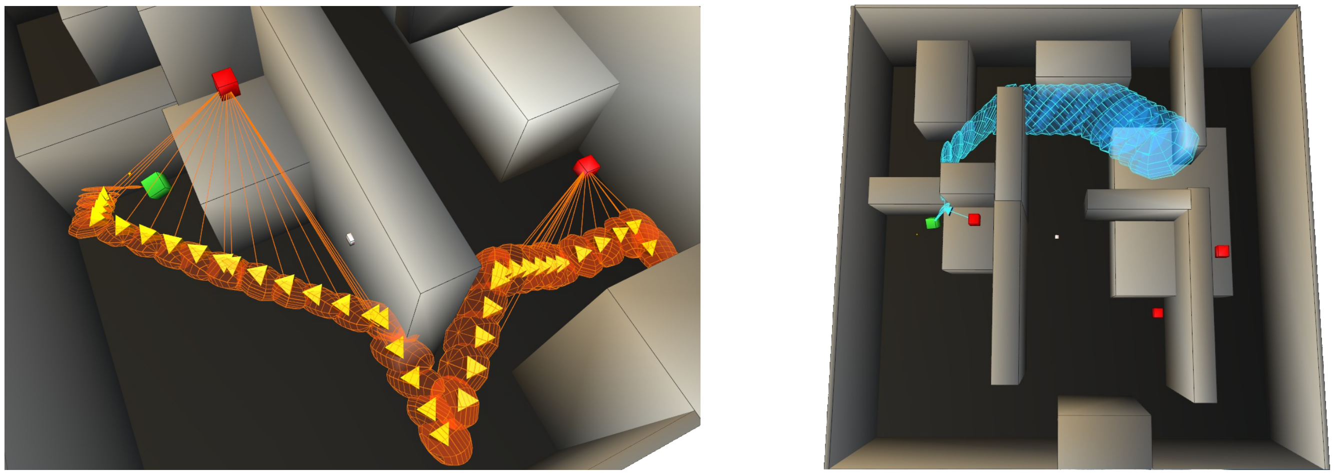

(c) Path of (a) shown from another different angle
(d) Path Produced by planner when attempting to minimize path length

Figure 3

An example of a path generated by the planning algorithm between two points in a complex environment is shown Figure 3. The minimum uncertainty path is seen to take a route which enables it to obtain numerous beacon measurements, resulting in better localisation by the internal particle filter compared to the minimum distance path.

Examples of the planner operating in simpler environments are shown in Figures 4, 5, 6. Figures 4 and 5 show the planner producing paths which take the MAV through areas in which localisation beacons are always visible. Figure 5 in particular highlights how the minimum distance path risk collisions, as the ellipsoid cuts deep into the wall near the destination. Figure 6 shows the planner producing a path taking a very specific route through an open space which maintains line of sight between the MAV and a set of beacons in order to retain localisation. Tables 7b and 7c show the computation times and properties of the robust paths and shortest distance paths. The uncertainty measure of the robust path is seen to always be lower than that of the shortest distance path. This indicates that in every scenario the MAV is more likely to successfully follow the robust path over the shortest distance path.

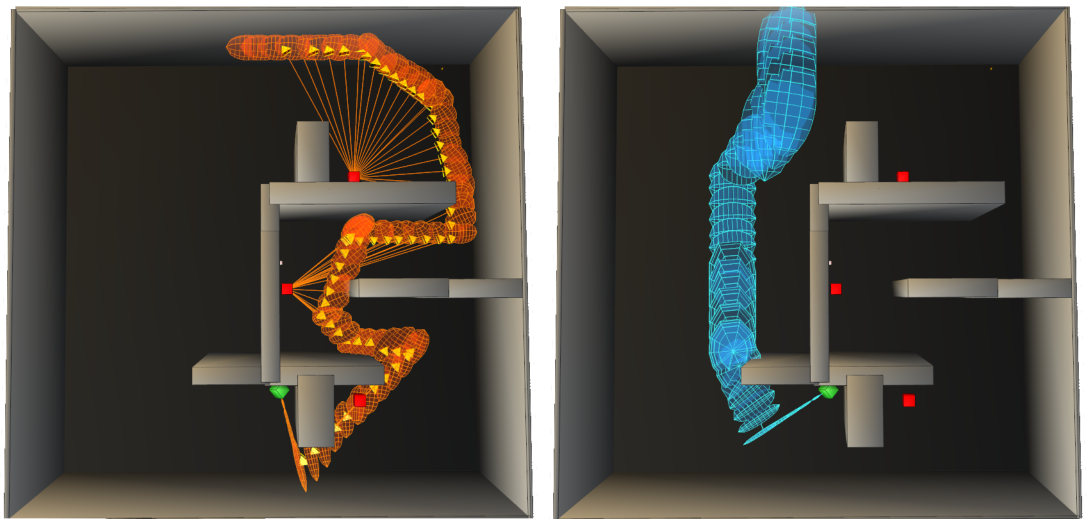
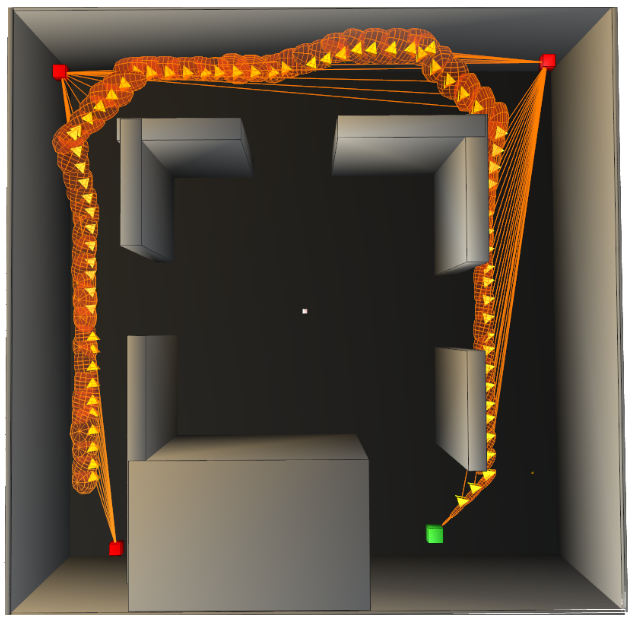

(a) (b)

Figure 4: (a) Shows the path produced by the planner which attempts to maintain good localisation. By comparision (b) shows the path produced when the planner simply trys to minimize distance traveled. The path shown in (a) goes through the corridor to the right resulting in the MAV taking a longer route but able to observe several localisation beacons along the way.

6 Conclusions

This paper has addressed the problem of path planning for a MAV-like vehicle with the presence of state uncertainty. A belief space path planning algorithm is presented in which a particle filter is used to represent the belief of the MAV's state and examine how the belief would evolve along potential paths through the environment. The algorithm was demonstrated in simulation producing routes in complex 3-D environments which effectively maintained good localisation minimizing the probability of the MAV straying from the path.

Much future work and examination remains, currently the planner does not explicitly take into account the probability of environmental collision when generating new paths, instead only performing comparisons to the current set of potential paths. Due to this omission many paths are generated which have a high likelyhood of collision. Though these paths are unlikely to be part of the final path outputted by the algorithm, significant computational time could be saved by culling such paths. By determining the probability of environmental collision at each step of path generation, the planner could also be formulated to produce paths that have a certain probability threshold of collision, rather than simply attempting to maintain a low state uncertainty.


(a)

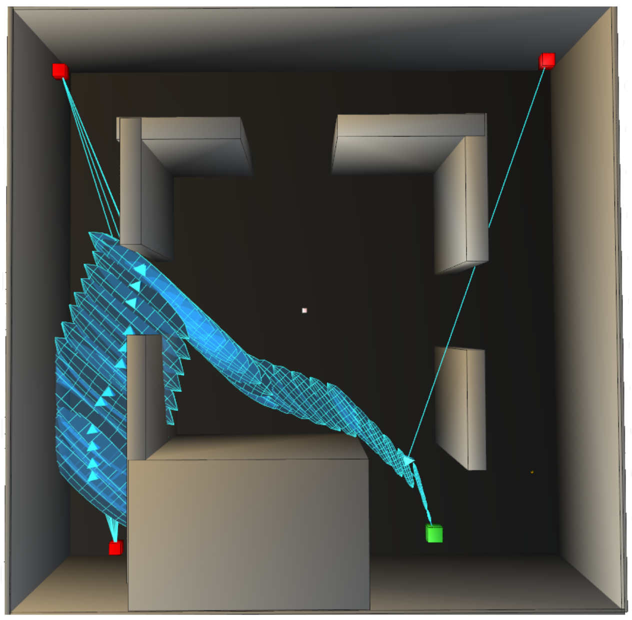
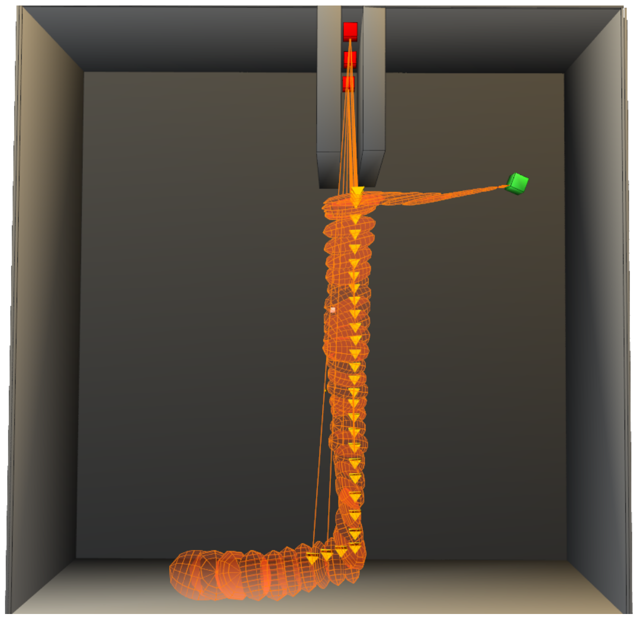

(b)

Figure 5: Here (a) shows the planner producing a path which takes a long indirect route to the destination but which always keeps the MAV in line of sight of a measurement beacon. (b) Shows the planners route minimizing only distance.


(a)

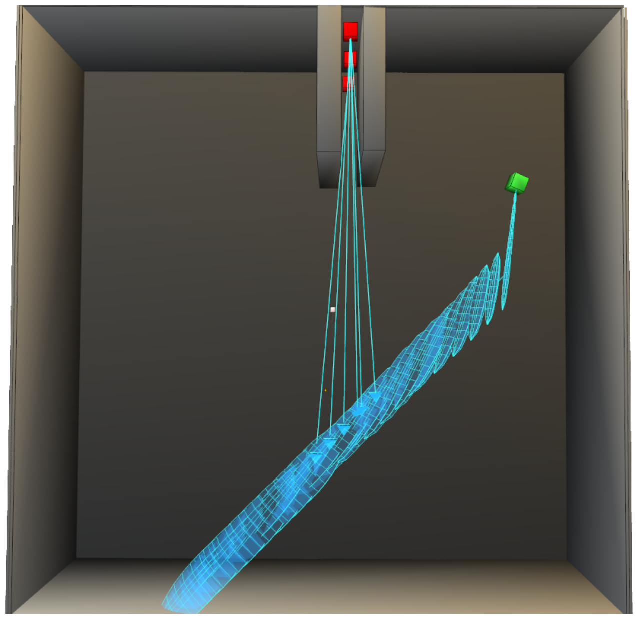

(b)

Figure 6: In this environment several localisation beacons are present placed at the end of a narrow corridor. The MAV is thus only able to observe them when it aligns itself with the corridor, bringing them within clear line of sight of its sensor. (a) Shows the planner producing a path diverging from the shortest route in order to align with this corridor and observe the beacons. (b) Shows the planners route minimizing only distance.

Figure	Time (s)
3	6.14
4	2.88
5	5.41
6	1.18

(a) Algorithm computation times

Final Uncertainty Measure	Path length
18.81	687.36
32.18	655.89
21.26	707.39
46.73	418.11

(b) Robust path properties

Final Uncertainty Measure	Path length
125.41	320.74
173.92	324.50
267.35	355.87
63.82	270.64

(c) Shortest distance path properties

Figure 7

References

[1] Salomon, B. (2004). Fast Line-of-Sight Computations in Complex Environments.

[2] Fox, D., Thrun, S., Bur-, W., & Dellaert, F. (1998). Particle Filters for Mobile Robot localisation, 470–498.

[3] Thrun, S. (2002). Particle Filters in Robotics.

[4] Thrun, S., Fox, D., Burgard, W., & Dellaert, F. (2001). Robust Monte Carlo localisation for mobile robots. Artificial Intelligence, 128(1-2), 99–141. doi:10.1016/S0004-3702(01)00069-8

[5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," IEEE Transactions on Robotics and Automation, vol. 12, no. 4, 1996.

[6] Prentice, S., & Roy, N. (2009). The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. The International Journal of Robotics Research, 28(11-12), 1448–1465. doi:10.1177/0278364909341659

[7] Bry, A., & Roy, N. (2011). Rapidly-exploring Random Belief Trees for motion planning under uncertainty. 2011 IEEE International Conference on Robotics and Automation, 723–730. doi:10.1109/ICRA.2011.5980508

[8] Melchior, N. A., & Simmons, R. (2007). Particle RRT for Path Planning with Uncertainty, (April), 10–14.

[9] Hirt, J., Gauggel, D., Hensler, J., Blaich, M., & Bittel, O. (n.d.). Using Quadtrees for Realtime Pathfinding in Indoor Environments.

[10] Kambhampati, S., & Davis, L. (1986). Multiresolution path planning for mobile robots. IEEE Journal on Robotics and Automation, 2(3), 135–145. doi:10.1109/JRA.1986.1087051

[11] Gottschalk, Stefan. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.

[12] Kuwata, Y., & How, J. (2004). Three Dimensional Receding Horizon Control for UAVs,

[13] Herman, M. (n.d.). Fast, three-dimensional, collision-free motion planning. Proceedings. 1986 IEEE International Conference on Robotics and Automation, 3, 1056–1063. doi:10.1109/ROBOT.1986.1087622