Proceedings of the
47th IEEE Conference on Decision and Control
Cancun, Mexico, Dec. 9-11, 2008

A Mission Planning Approach for UAV Applications

W. A. Kamal and R. Samar

Abstract— **In this paper, a 2-D mission planning approach is developed for UAV applications. The main contribution of the paper is the development of an extension to the Bellman Ford algorithm that enables incorporation of constraints directly into the algorithm during run time. The dynamical constraints of the vehicle, such as its angle of turn, can therefore be catered for. Furthermore, a procedure for computing a number of sub-optimal paths is developed so that a range of options is available to the user for selection. These sub-optimal paths are generated in an order of priority (optimality). An objective function is developed which models different conflicting objectives in a unified framework; different objectives can be assigned different weights. The objectives may include minimizing the length of the path, keeping the path as straight as possible, flying over areas of interest, etc. The algorithm is integrated into a software package and tested for complex mission objectives, and results are discussed.**

I. Introduction

The mission planning problem is about computing an optimum path between the initial (take-off) and terminal (landing) points, while satisfying user requirements without violating given constraints. The path must avoid 'no-go' regions or obstacles for operational reasons. It should however traverse through areas of interest (AOI) where aerial photography for instance may be required. The optimality of the path may be dictated by factors such as the length of the path, its straightness, angle of approach to the final (landing) point being close to a desired value, etc. These can be defined as weighted objectives to be minimized in an optimization framework. Vehicle (robot) constraints such as the turn angle also need to be incorporated into the optimization algorithm, thus providing the tracking controllers with feasible reference trajectories. Trajectories that do not comply with system dynamics and constraints can place impractical demands on the controller, and therefore must be avoided. We assume here that all data and constraint (global) information is available beforehand for path planning, which is done at the start of the mission.

A mixed integer linear programming (MILP) formulation of the route planning problem is proposed in [1], [2], [3], [4]. This strategy formulates the problem using integer variables and employs a branch-and-bound algorithm to optimize the route. The branch-and-bound algorithm (see for example [5]) tackles the problem by relaxing the integer restriction on the variables and iteratively solves for and modifies the solution space by adding special constraints. The integer variables however grow fast with problem size and hence this becomes computationally very expensive for larger problems. Further nonlinear objective functions and constraints are difficult to model in this framework.

Other methods entail transforming the route planning problem into an entirely different problem with more convenient methods of solution. Examples are the potential field method ([6] and [7]), the mass-spring-damper method ([7]) and the chain-link method [8]. In these formulations, the solution to the path planning problem is obtained by solving for the motion of a point mass (or a number of connected masses) under the influence of an artificial potential field. The variation in the field reflects the presence of obstacles and no-go areas; irregular shaped obstacles are however difficult to model by these methods. Another popular approach to path planning is graph-based search ([9], [10]); examples include rectilinear graph, visibility graph ([11], [12]) and Voronoi graph ([7], [13], [14]). These graphs can be searched using Dijkstra's algorithm, Bellman Ford algorithm ([15]) or A^* algorithm ([16]) to find the lowest cost path. In the visibility graph approach, the obstacles are usually treated as polygons, and the set of possible paths includes straight line segments formed by joining the vertices of these polygons (provided these segments do not intersect the no-go areas). In the Voronoi approach the obstacles are taken as point masses and a graph is constructed such that the edges of the graph are equidistant from a pair of masses. These approaches consider obstacle avoidance; however inclusion of areas of interest into the graph-based framework is not clear. Furthermore vehicle constraints cannot readily be built into the problem (both for graph-based and potential-field methods) and weighting different objectives against one another is not straightforward.

In this paper, a graph based path planning approach is presented that addresses the above limitations. Firstly an objective function is developed that models different conflicting objectives in a unified framework; different objectives can be assigned different weights. The objectives may include minimizing the length of the path, keeping the path as straight as possible, visiting areas of interest, etc. The main contribution of the paper is the development of an extension to the Bellman Ford algorithm that enables incorporation of constraints directly into the algorithm during run time. The dynamical constraints of the vehicle are therefore not violated when searching for the optimal path. Furthermore, a procedure for computing a number of sub-optimal paths is developed so that a range of options is available to the user for selection. The final path plan consists of a number of waypoints which define the path of the vehicle in terms of straight line segments.

The paper is organized as follows. The path planning

Author's are with National Engineering & Scientific Commission, Islamabad, Pakistan. Email: kamal@cesat.gov.pk

978-1-4244-3124-3/08/$25.00 ©2008 IEEE

problem is defined in section II. Section III discusses a particular grid generation scheme. No-go area avoidance is discussed in section IV, and the cost function is developed in section V. The optimization algorithm is presented in section VI along with the procedure for generating multiple paths. Simulation results are presented in section VII; section VIII concludes the paper.

II. Problem definition and assumptions

The aim is to compute a path between an initial and final point that minimizes the objective function over a path $\mathcal{P}$:

$$J(\mathcal{P}) = \int_{\mathcal{P}} C(\rho) d\rho, \tag{1}$$

where $C(\rho)$ is the cost per unit length, and $d\rho$ is the differential arc length at a point $\rho \in \mathcal{P}$. The graph being searched consists of edges and nodes and hence the path will be an ordered sequence of line segments. The path integral therefore becomes a discrete summation of edge costs and can be written as:

$$J(\mathcal{P}) = \sum_{j=1}^{n} E^j, \tag{2}$$

where n is the total number of edges and E^j is the cost of the j^{th} edge. The cost function will be discussed in detail in the following sections. The minimization of the objective function (2) will be subject to the following constraints:

1) No-go areas/obstacles must be avoided (hard constraints); these are modelled as circles or polygons.
2) Areas of interest should be visited (soft constraints), these are modelled as circles of given radii. The objective is to fly straight and level along a diameter of the circle so that best conditions for aerial photography (or reconnaissance) are maintained.
3) Vehicle dynamical constraints must not be exceeded, these include:
   - the vehicle's angle of turn, and
   - the total fuel available (or the maximum distance).

The following assumptions are made:

1) The initial and final points between which the path is to be planned are pre-defined.
2) The environment (no-go areas, areas of interest, vehicle constraints, etc.) is known and static.
3) The vehicle is equipped with an inertial navigation unit (INU) (which may be aided by GPS). The INU error growth rate is fixed and known. The same would be true for a mobile robot in which the estimate of position has an error that grows with time.

III. Grid generation

The process of generating a grid provides a graph consisting of nodes and edges that represents all possible paths connecting the given initial and final points. A grid may be generated in a number of different ways. The computational efficiency of the route planning algorithm, and the optimal path obtained depend on the grid. We will now describe one particular method of grid generation that we have found to be particularly useful and efficient.

A. Node generation

We shall first discuss generation of nodes. The process is explained below.

1) Regular nodes: These nodes are inserted by considering (imaginary) concentric semi-circles with F as the centre, and radii $R, 2R, 3R, \ldots, nR$. Where nR is less than or equal to the distance between initial and final point and R is the turn radius of vehicle (Figure 1). Each point on the semi-circle represents a possible node position, but the computational complexity increases with increase in the number of nodes in the problem. Here we choose to insert six node points on each semi-circle, three on either side of the line joining the initial and final points.

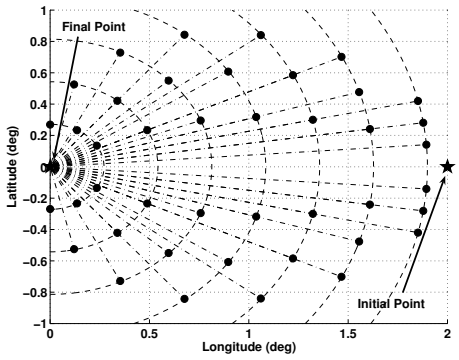

Fig. 1. Regular node generation scheme.

2) Nodes based on expanded no-go areas: No-go areas or obstacles are modelled as polygons, which are then expanded to cater for the navigation error build-up in the vehicle with time. The expansion of these polygons is done as follows.

- First the straight line distance from the initial point I to the farthest vertex of each polygon (no-go area) is computed. We call these distances d_i where i stands for the i^{th} polygon. The navigation error of the vehicle depends on the error of the inertial navigation unit (INU), which builds up with time. The farther a polygon is from point I, the more will be the navigation error of the vehicle as it reaches there. Although GPS integration can correct INU errors, but we will assume here that in the worst case GPS signals may not be available. The expansion of each polygon is therefore done in direct proportion to its distance from the initial point I. This will ensure that the vehicle does not enter no-go zones even in the presence of INU sensor drifts.
- The straight-line distances are multiplied by a safety factor f $(1.25 \leq f \leq 2.5)$ to cater for the deviation of the path from a straight line. The travel time of the vehicle to the polygons (no-go areas) is computed by dividing the distances by the average speed of the vehicle S, i.e., $t_i = d_i f / S$. The navigation error for the i^{th} polygon will be proportional to t_i.
- Each polygon is now expanded in proportion to its expected INU error. The expansion is done uniformly in all directions. This is done by moving all vertices outward along the bisectors of the vertex angles, by the

same amount. The polygons may be convex or concave, and so there is a need to decide on the outward direction at each vertex. The expansion direction is determined by taking a test point on the bisector close to a vertex. If the test point lies inside the polygon, then the expansion direction is opposite to the direction along which the test point was taken. The expanded polygons thus obtained incorporate into them the vehicle's navigation error, and therefore these are to be avoided during route planning. The vertices of these expanded polygons are stored as node points.

3) Nodes based on Areas of Interest (AOI): The user can define areas of interest for the robot to traverse through, or over which it may be desirable for a UAV to fly for operational reasons (such as aerial photography or reconnaissance). The areas of interest are modelled as circles whose centres are taken as graph nodes.

4) Manual nodes: Additional nodes can be inserted manually by the user.

5) Removal of unacceptable nodes: During the node generation process some nodes which violate mission planning constraints may have been generated, and therefore must be removed. We remove nodes:

- that lie inside a no-go area,
- whose distance from the initial point I is less than a certain value (this is done so that no nodes are inserted in the initial part of the flight, i.e., the take-off phase), and
- that lie outside the ellipse and satisfy the following equation

$$d(I,V) + d(V,F) > L_{max}, \tag{3}$$

where L_{max} is the maximum allowed path length (defined by the vehicle's maximum range), and V is any node. The maximum path length constraint is also checked for once the paths are computed.

B. Edge formation

The computational time for searching for the best path can be considerably reduced by connecting the nodes in an intelligent way. Nodes are connected to generate edges as follows.

- A reference point behind the initial point I on the line IF is found such that the distance between F and the reference point is L_{max}.
- The distance of each node from the reference point is computed.
- Nodes are sorted in ascending order with respect to the distances, initial and final points are then inserted at the start and end of the list. This generates a vector of nodes $[I, N_1, N_2, \ldots, N_N, F]$.

Each node in this sorted vector is a candidate for connection to all nodes ahead of it. So for example, the node N_i is connected to all nodes $N_{i+1}, \ldots, N_N, F$, provided these edges avoid all expanded obstacles/no-go areas. Checking for edge intersection with no-go areas is described in the following section.

IV. No-go area avoidance

Obstacles/no-go areas are modelled as polygons lying on the surface of the earth. Now each candidate edge is to be individually checked for avoidance from all obstacles. If for a given edge an intersection with an obstacle is detected, that edge is declared unfit. The correct way of doing this is to check for intersection or inclusion of an edge (which is an oblate arc on the earth's oblate surface) and the oblate polygon that models the obstacle. However since the obstacles and edges are small (of the order of several hundred meters) with respect to the radius of the earth, the problem can be simplified to lines and plane polygons instead of oblate geometrical shapes, without any loss of accuracy. The procedure involves the following steps:

Step 1 Using the coordinates (latitude, longitude) of edge and polygon vertices, a plane projection of the edges and obstacles is obtained on a two-dimensional cartesian coordinate system. Intersection of the candidate edge of the graph with each polygon edge is checked on this plane.

Step 2 In case of no intersection between the graph and polygon edges, it is checked whether the graph edge lies completely inside a polygon.

If both these conditions are false, the candidate edge is considered fit for route planning.

V. Cost function

Different objectives need to be considered when defining the edge cost. Often these objectives are conflicting, and thus there is a need to incorporate them into a single multi-objective function that satisfies mission requirements. In this study the objectives considered are:

- minimization of path length,
- maximization of traverse through (or flight over) areas of interest,
- minimization of en-route turns (waypoints), and
- minimization of deviation from a desired angle of approach to the final point.

The first objective can be modelled as L_E/L_{max}, where L_E the edge length is normalized by dividing by the maximum length L_{max}. The second objective of maximizing the visiting of areas of interest (AOI) can be modelled by adding $-D$ to the edge cost, where $0 \leq D \leq 1$ is the AOI cost parameter and is taken at the second (farther) node of the edge (which is also the centre of the area of interest). For edges which do not have an AOI as the second node, D is taken to be 0; a larger value of D depicts an AOI of greater importance. The objective is to select the edge which has an AOI at its farther end. The third objective of minimization of turns can be transformed into an equivalent objective of minimizing the number of waypoints. This can be modelled by inclusion of the term c/N_{max} in the edge cost, where c is a constant, usually taken to be 1 (the same for all edges), and N_{max} is the maximum number of waypoints used for normalization of the cost. The fourth objective is to approach the final point (which could be a landing strip, for example)

from a pre-specified direction, and to minimize the deviation in the approach angle from the desired heading; this is only included for edges connected to the final point F. The cost for this objective is modelled as $-k\left(1-\frac{|\psi-\psi_{des}|}{\Delta\psi_{max}}\right)$, where ψ and ψ_{des} are the actual and desired heading angles for the edge, k is a flag which is one for edges connecting node F and zero otherwise, and $\Delta\psi_{max}$ is the maximum allowed heading angle deviation and is used to normalize the heading angle cost component. The edge cost can now be written as:

$$E_{cost}=\frac{W_LC_1+W_DC_2+W_NC_3+W_HC_4}{W_L+W_D+W_H+W_N}, \quad (4)$$

where

$$\begin{aligned} C_1 &= L_E/L_{max}, \quad C_2=-D \\ C_3 &= c/N_{max}, \quad C_4=-k\left(1-\frac{|\psi-\psi_{des}|}{\Delta\psi_{max}}\right). \end{aligned}$$

Here E_{cost} is the edge cost and W_L, W_D, W_N and W_H are user defined weights for the four objectives.

VI. Route planning

A. Route optimization

The edge cost components discussed above can be negative, the choice of the optimization algorithm therefore needs to be done accordingly. These negative weights arise naturally in our problem formulation. The Bellman-Ford algorithm computes the shortest path between two given nodes in a weighted digraph where some of the edges may have a negative cost ([15]). Dijkstra's algorithm solves the same problem with a lower computational burden, but requires the weights to be non-negative.

Our goal is to select an algorithm that solves the minimum cost problem given in (4) above; furthermore we would like to work out a number of sub-optimal paths also, so that a range of options are available for selection. The Bellman-Ford algorithm in the process of finding the optimal path between the source and destination nodes, also finds lowest cost paths from the source node to all other nodes. This feature of the algorithm is exploited in the computation of a number of sub-optimal paths as discussed below.

We will modify the Bellman-Ford algorithm to our specific requirement and incorporate the turn angle constraint into the algorithm directly during run time. The optimization starts by assigning labels to every node and every edge. Each node label consists of two fields: the first field is the cost in going from the initial node I to that node, and the second field contains the name of the edge that connects this node to node I on the shortest route found so far. We also give labels to edges so that the turn angle constraint can be incorporated into the optimization process for all nodes of candidate paths. The edges are given labels similar to nodes: the first field is the cost from node I to the end node of the edge on the shortest path through that edge (found so far), and the second field is the name of the previous edge. The algorithm can be divided into four parts:

- initialization,
- forward computation,
- reverse computation, and
- multiple path computation.

For single path computation between two nodes, the algorithm is stopped after the forward computation step, but for multiple path computation the complete algorithm is run. Before describing the algorithm we first define the terminology used. The graph under consideration is a directed graph with an edge j starting from node $b(j)$ and ending at node $e(j)$ with cost E^j. Let $(\mathcal{R}_I^j,\mathcal{P}_I^j)$ be the label for the j^{th} edge, where $\mathcal{R}_I^j$ is the cost of the shortest path (found so far) from node I to $e(j)$ via the j^{th} edge, and $\mathcal{P}_I^j$ is the edge previous to j on that path. Let the label for the i^{th} node be (R_I^i,P_I^i) where R_I^i is the cost from node I to i for the shortest path (found so far), and P_I^i is the edge connecting node i on that path. N and n denote the total number of nodes and edges respectively, and α is the maximum angle by which the vehicle can turn. *'Treated'* is a flag used to avoid unnecessary computation in the algorithm logic. Now we give a complete description of each part of the algorithm.

Initialization: Assign a label $(0,0)$ to the initial node I, and the labels $(\infty,0)$ to all other nodes. Also assign labels $(\infty,0)$ to all edges. A zero in the second field indicates that either no preceding edge assignment has been made, or the edge originates from node I.

Forward computation: Pseudo-code for this part of the algorithm is given below.

$$\begin{array}{ll}
For & i \leftarrow 1\ to\ N \\
 & Treated \leftarrow False \\
For & j \leftarrow 1\ to\ n \\
If & b(j) = I \\
If & E^j < R_I^{e(j)} \\
 & R_I^{e(j)} \leftarrow E^j \\
 & P_I^{e(j)} \leftarrow j \\
 & \mathcal{R}_I^j \leftarrow E^j \\
 & Treated \leftarrow True \\
EndIf & \\
ElseIf & P_I^{b(j)} \neq 0 \\
 & \beta \leftarrow \angle\, b\left(P_I^{b(j)}\right) b(j)e(j) \\
If & \beta > 180-\alpha \\
If & R_I^{b(j)} + E^j < R_I^{e(j)} \\
 & R_I^{e(j)} \leftarrow R_I^{b(j)} + E^j \\
 & P_I^{e(j)} \leftarrow j \\
 & \mathcal{R}_I^j \leftarrow R_I^{b(j)} + E^j \\
 & \mathcal{P}_I^j \leftarrow P_I^{b(j)} \\
 & Treated \leftarrow True \\
ElseIf & R_I^{b(j)} + E^j < \mathcal{R}_I^j \\
 & \mathcal{R}_I^j \leftarrow R_I^{b(j)} + E^j \\
 & \mathcal{P}_I^j \leftarrow P_I^{b(j)} \\
 & Treated \leftarrow True \\
EndIf & \\
EndIf &
\end{array}$$

$EndIf$
$EndFor$
If $Treated\ is\ False$
$break$
$EndIf$
$EndFor$

Two *'for'* loops are run in the forward computation part of the algorithm. The outer loop is run N times (N being the total number of nodes), and the inner loop runs from 1 to n (the number of edges in the problem). In the start of the program we check to see if the start node of an edge is the initial node I, in which case the cost of the end node of the edge is compared with the cost of the edge itself. If the cost of the edge is smaller, the label of the end node is updated accordingly. If an edge does not originate from the initial node I, we check if a preceding edge assignment for the edge start node exists. If true, the angle (of turn) β at the edge start node is computed assuming the route includes the edge, and this is compared to the maximum allowed turn angle α. In case the turn angle constraint is not exceeded, the previously stored cost at the end node of the edge is compared to that accrued by going through this edge. In case of a lower cost, the end node label and the edge label are both updated; whenever there is an end node assignment, there is also an edge assignment (the reverse is not true). If however the cost at the end node while going through the edge is not lower, then this cost is compared with the cost stored in the edge label. In case of a lower cost, only the edge label is updated; this implies that this new path from I to $e(j)$ via the j^{th} edge results in a lower cost, and hence the label of the j^{th} edge now contains this cost.

Now for each edge j, the shortest path vector $j2I$ from $e(j)$ to the initial node I can be traced out by linking each node to its preceding node as follows:

$$j2I = \left[e(j),\ b(j),\ b\left(\mathcal{P}_I^j\right),\ b\left(\mathcal{P}_I^{\mathcal{P}_I^j}\right),\ b\left(\mathcal{P}_I^{\mathcal{P}_I^{\mathcal{P}_I^j}}\right), \ldots,\ I \right].$$

The shortest path vector $I2j$ from I to $e(j)$ can be obtained by reversing this; the corresponding cost is $\mathcal{R}_I^j$.

Reverse computation: The graph under consideration is a directed graph with an edge j starting from node $b(j)$ and ending at node $e(j)$. Reverse computation is done for computing multiple routes as discussed below. In this case we interchange the role of the nodes $b(j)$ and $e(j)$ and also of the nodes I and F. This gives optimal paths from the final point F to all edges of the graph.

Multiple route computation: Now the procedure of finding multiple (suboptimal) routes is described. The routes are generated in an order of priority (ordered according to cost); if one route does not meet user satisfaction for some reason, the next route can be considered, and so on.

- Find a vector of costs considering all edges, i.e., the vector $\left[\mathcal{R}_I^j - E^j + \mathcal{R}_j^F\right]$. The edge cost E^j is subtracted because it is included in both $\mathcal{R}_I^j$ and $\mathcal{R}_j^F$.
- Also find the corresponding list of route vectors, viz $[I2j \cup \overline{j2F}]$ where $\overline{j2F}$ is the shortest route vector from edge j to the final node F but without nodes $b(j)$ and $e(j)$ (these nodes are already included in the vector $I2j$).
- Sort the cost vector in ascending order of cost, and arrange the list of routes accordingly. Check all routes for total length and delete those that do not meet the maximum length constraint. The k best routes in order of priority correspond to the first k elements of the final sorted vector.

VII. Simulation Results

The above algorithm is coded into a software application for finding trajectories from given initial to final points. Obstacles/no-go areas (in the form of circles or polygons) can be inserted through click and drag operations. Mission-related parameters, constraints and mission areas can also be specified.

The mission area chosen for testing the algorithm is 3 degrees by 3 degrees in latitude and longitude, respectively. Figure 2 shows a scenario created by placing 8 obstacles and 20 areas of interest at different locations in the mission area. The initial (I) and final (F) points are shown as green and red stars respectively. Green circles indicate areas of interest, whereas red polygons indicate obstacles/no-go areas. Regular grid points are shown as blue dots, while expanded polygon nodes are shown as cyan dots. The reference ellipse shown as black dots is the boundary of the mission area, drawn using the maximum range constraint. The maximum turn angle constraint is set as 60 degrees.

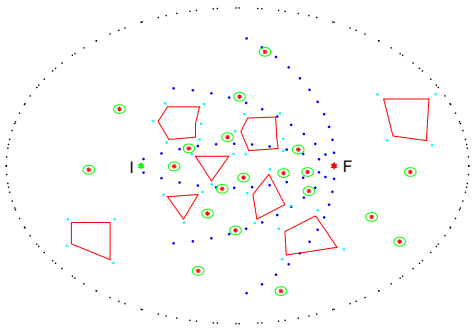

Fig. 2. Scenario created by placing 8 obstacles and 20 areas of interest.

Optimal paths (satisfying all constraints) are computed using the above algorithm; the effect of varying the weights is studied here. The objective of visiting maximum areas of interest can be given top priority by setting the weights as $W_L = 0$, $W_D = 1$, $W_N = 0$ and $W_H = 0$. The three best paths in an order of priority are shown in Figure 3 with red, blue and black colours, respectively. Each path selected four areas of interest with the first area being common for all paths.

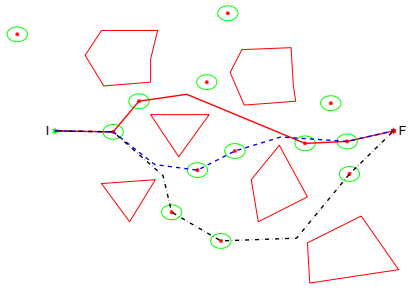

Fig. 3. Three best paths using weights $W_L = 0$, $W_D = 1$, $W_N = 0$ and $W_H = 0$.

We now incorporate the approach angle constraint with the desired approach angle having an azimuth of 120 deg, and set its weight as $W_H = 10$, with other weights kept unchanged. All three paths approach the final point with the specified angle, the optimal path (red) passing through three areas of interest whereas the other two paths go through four AOIs each (Figure 4).

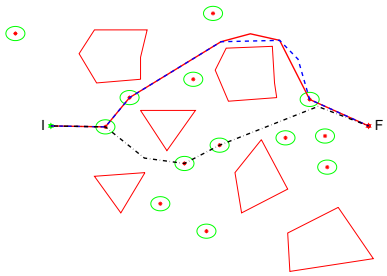

Fig. 4. Three best paths with 120 deg approach azimuth, and weights $W_L = 0$, $W_D = 1$, $W_N = 0$ and $W_H = 10$.

The number of turnings (waypoints) and the path length can be reduced by giving non-negative values to weights W_N and W_L respectively. The algorithm is run again with weights set as $W_L = 1$, $W_D = 1$, $W_N = 20$ and $W_H = 0$. The approach angle weight is made zero, whereas the waypoint and path length weightings are increased. This weight setting reduces the number of waypoints as shown in Figure 5. The optimal path has one turning, and passes through one area of interest.

VIII. Conclusion

A 2-D path planning algorithm is developed for UAV applivations. The algorithm can absorb different dynamical constraints (like maximum turn angle) into the computation process. Furthermore it provides multiple feasible paths that avoid obstacles and traverse through areas of interest by minimizing a multi-objective cost function. Weightings can be specified for important mission objectives, such as the length of the mission, the straightness of the path, desired approach to final point, and flight directly over areas of interest. The algorithm is tested for complex mission objectives on a UAV path planning example, and results discussed.

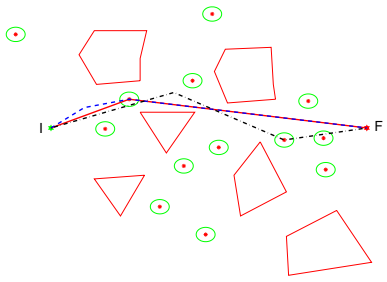

Fig. 5. Three best paths with weights $W_L = 1$, $W_D = 1$, $W_N = 20$ and $W_H = 0$.

References

[1] T. Schouwenaars, B. DeMoor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *Proceedings of the European Control Conference*, Portugal, September 2001, pp. 2603–2608.

[2] A. Richards, J. How, T. Schouwenaars, and E. Feron, "Plume avoidance maneuver planning using mixed integer linear programming," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2001.

[3] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the American Control Conference*, May 2002.

[4] W. A. Kamal, D. W. Gu, and I. Postlethwaite, "Real time trajectory planning for UAVs using MILP," in *Proceedings of the CDC-ECC Conference*, Seville, Spain, December 2005.

[5] H. A. Taha, *Operations Research: An Introduction*, 6th ed. Prentice-Hall, Inc., 1997.

[6] M. B. McFarland, R. A. Zachery, and B. K. Taylor, "Motion planning for reduced observability of autonomous aerial vehicles," in *Proceedings of the 1999 IEEE International Conference on Control Applications*, 1999, pp. 231–235.

[7] S. Bortoff, "Path planning for UAVs," in *Proceedings of the American Control Conference*, June 2000, pp. 364–368.

[8] T. W. McLain and R. W. Beard, "Trajectory planning for coordinated rendezvous of unmanned air vehicles," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2000.

[9] R. R. Murphy, *Introduction to AI Robotics*. MIT Press,, 2000.

[10] A. Sipahioglu, A. Yazici, O. Parlaktuna, and U. Gurel, "Real-time tour construction for a mobile robot in a dynamic environment," *Journal of Robotics and Autonomous Systems*, vol. 56, no. 4, pp. 289–384, April 2008.

[11] M. Neus and S. Maouche, "Motion planning using the modified visibility graph," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, October 1999, pp. 12–15.

[12] H.-P. Huang and S.-Y. Chung, "Dynamic visibility graph for path planning," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[13] O. Takahashi and R. J. Schilling, "Motion planning in a plane using generalized Voronoi diagrams," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 2, pp. 143–150, April 1989.

[14] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," *The 4th International Symposium on Voronoi Diagrams in Science and Engineering*, pp. 38–47, July 2007.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[16] N. J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publisher Company, 1980.