Stable Trajectory Design for Highly Constrained Environments using Receding Horizon Control

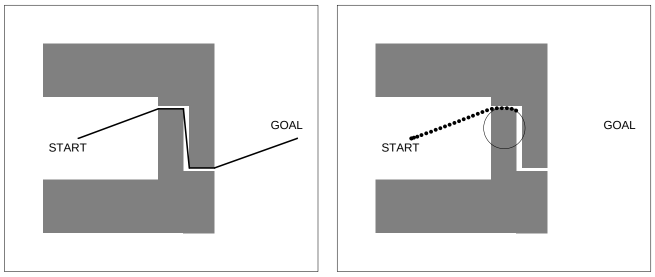
Yoshiaki Kuwata and Jonathan P. How
Space Systems Laboratory
Massachusetts Institute of Technology
{kuwata,jhow}@mit.edu

***Abstract*— This paper presents a new formulation of a stable receding horizon controller (RHC) for the minimum time trajectory optimization problem with a vehicle flying in a complex environment with obstacles and no-fly zones. The overall problem is formulated using Mixed-integer Linear Programming (MILP). The RHC uses a simple vehicle dynamics model in the near term and an approximate path model in the long term. This combination gives a good estimate of the cost-to-go and greatly reduces the computational effort required to design the complete trajectory, but discrepancies in the assumptions made in the two models can lead to infeasible solutions. This paper extends our previous RHC formulation to ensure that the on-line optimizations will always be feasible, while eliminating the binary variables associated with feasible turns. Novel pruning and graph-search algorithms are also integrated with the new MILP RHC, and these are shown to significantly reduce the computation time. A worst case analysis is performed to obtain an upper bound on the planning horizon, and the resulting controller is analytically shown to guarantee finite-time arrival at the goal.**

I. Introduction

Path planning problems have been well studied in the robotic field, and several approaches such as visibility graph, potential field, roadmap method, and cell decomposition have been proposed [1]. These traditional approaches are typically either computationally intensive and not suitable for real-time applications, or can produce suboptimal solutions that cannot guarantee arrival at the goal. These approaches includes the kinematics but further extension is required to capture the dynamics of the aircraft (*e.g.,* speed constraints, turn limitations).

Our approach address this problem by using a hybrid approach. Our receding horizon controller (RHC) uses a simple model of the aircraft dynamics [2], [3] model over the planning horizon in the detailed trajectory generation phase, and a simple kinematic model (*i.e.,* collision free straight line segments) beyond it [4]. Although replanning can typically find a dynamically feasible trajectory near the line segment path, the trajectory optimization problem can become infeasible if there is a large difference in the assumptions of the vehicle capabilities. Fig. 1 illustrates the case where kinodynamically infeasible straight line segments lead to an infeasible trajectory design [5]. One approach to avoid this situation is to place a turning circle at each corner when constructing a cost map, and enforce the rule that the vehicle moves towards the arc of the circle, not the corner. Ref. [5] used this approach and introduced binary variables to encode the *join* and *leave* events on the turning circles,



(a) Straight line path goes through the narrow passage (b) Infeasible problem in the detailed trajectory design phase

Fig.1: Trajectory in highly constrained environment

but these binaries complicate the MILP problem and make it much harder to solve.

A new approach presented in this paper ensures the existence of a *kinodynamically feasible* turn [4], [6] at each corner by introducing three turning circles per corner and by applying a modified Dijkstra's algorithm when constructing the cost map. When searching for the shortest path, this algorithm rejects a sequence of nodes if the turning circles cannot be suitably placed (*i.e.,* a kinodynamically infeasible sequence). The generated tree of nodes gives the shortest path from each node to the goal along the straight lines in the regions where a feasible path is guaranteed to exist. When optimizing the trajectory using RHC, the planning horizon is then extended beyond the execution horizon such that while executing the plan, the vehicle will always stay in the regions from which a feasible path to the goal exists. The key point is that this does not require pre-placed turning circles in the MILP optimization, which eliminates the additional binary variables used in Ref. [5]. Also, the feasibility check on the sequences of nodes enables us to prune infeasible cost points before the MILP optimization while still retaining the freedom to make decisions about which paths to take around the obstacles. This pruning algorithm is shown to significantly reduce the computational effort. This highly modified cost map can also be used to establish sufficient conditions on the planning horizon length. This leads to an analytic derivation of the minimum planning horizon length, which is one of the key results of this paper. This paper defines the "stability" of the trajectory design process as guaranteeing arrival at the goal, and then proves the stability of our RHC based on the existence of a feasible path in the detailed trajectory design phase. Finally, several simulation results are presented to verify the stability and performance of this new approach.

II. Modified Cost Map

This section constructs a reliable cost map that is used in the RH-MILP to estimate the trajectory beyond the planning horizon [2], [4]. Obstacles are modelled as rectangles, and nodes for the graph are the obstacle corners and the goal point. This is a good approximation, because with very fast turning dynamics, the shortest path from one node to another would consist of the straight line segments provided by the visibility graph. The visibility graph is first generated considering only kinematics. Then, Dijkstra's algorithm is used to search for the shortest path from each node. A circle placement algorithm is embedded in the Dijkstra's algorithm to incorporate vehicle dynamics. The modified Dijkstra's algorithm incorporates both kinematics and dynamics and yields a more reliable cost map for the RH-MILP.

A. Three Circles

The optimal trajectory of an aircraft flying at a constant speed with limited lateral acceleration is depicted as a series of straight line segments and arcs of the minimum turning circle with radius $r_{\min}$ [7]. Fig. 2 shows three turning circles ("leave circle", "corner circle", and "return circle"), the arcs of which comprise a path that is flyable by the vehicle when it changes from one straight line to another. The vehicle leaves the original straight line path at a "leave point", make a turn with the maximum lateral acceleration allowed, passes around the obstacle corner, and then aligns its heading to the next straight line at a "return point". The straight line segments in Fig. 2 are the connections in the visibility graph, guaranteed to be collision free. Two more conditions must be satisfied in order to construct a tree of kinodynamically feasible paths. First, the turning circles must be collision free (*i.e.*, not intersect any obstacles). Second, when constructing a tree backwards from a goal, a return point around a corner must not go over the leave point of its previous corner. This second condition requires that the vehicle must come back to the straight line and align its heading with it, before deviating from the straight line to make the next turn.

For the turning maneuver at each corner, the difference between the path length along the straight line and the one along the arcs can be analytically obtained as

$$\begin{aligned}\Delta J \;&=\; (2\beta + \pi/2 - \theta_1 - \alpha)\, r_{\min} \\ &\quad - \left[\left\{1 + \sin(\theta_1 + \alpha)\right\}\tan\beta + \cos(\theta_1 + \alpha)\right] r_{\min} \quad (1)\\ \beta \;&=\; \arccos\left\{\frac{1 + \sin(\theta_1 + \alpha)}{2}\right\}\end{aligned}$$

where α and θ_1 are shown in Fig. 2. Note that this path length along the two arcs gives an upper bound of the optimal path length.

B. Modified Dijkstra's Algorithm

This section extends the previous way of constructing the cost map to ensure the existence of kinodynamically feasible paths around the straight line segments. Proper placement

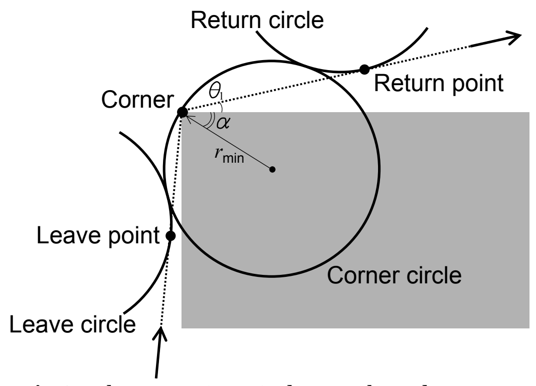


Fig.2: Three turning circles at obstacle corner

of turning circles is critical to obtain a less conservative estimate of the existence of a feasible turn. An analytical way of placing collision free circles is introduced in the Ref. [6]. This circle placement algorithm is embedded in a modified Dijkstra's algorithm presented in this section that accommodates the rejection criteria for joining two arcs when searching for the shortest path.

The original Dijkstra's algorithm [8] involves the labelling of a set of fixed nodes $\mathcal{P}$, of which the shortest node distances D_j from the origin node to each node has been found. Let w_{ij} denote the straight line distance between each pair of nodes. The weight w_{ij} is set to ∞ if node i is not visible from node j due to the obstacles. The goal is regarded as an origin node in the algorithm. The modified Dijkstra's algorithm presented here chooses a node which has a minimum distance label, and from which a kinodynamically feasible connection to the current node is guaranteed to exist.

Algorithm Overview: The index of the goal node can be set to be 1. The modified Dijkstra's algorithm then consists of the following procedure:

1. Set current node $i = 1$, and initialize:
$$\begin{aligned}\mathcal{P} \;&=\; \{1\}\\ D_j \;&=\; \begin{cases} 0 & (j = 1)\\ w_{1j} & (j \neq 1)\end{cases}\end{aligned}$$
2. Place the leave point for the goal on the goal.
3. Set node i as the successor of each node j that is visible from the node i (*i.e.*, $\{\, j \,|\, w_{ij} \neq \infty\}$). A successor node is the next node on the path toward the goal.
4. Find the next closest node from the set of unfixed nodes, and set it as a new current node i:
$$i := \arg\min_{j \notin \mathcal{P}} D_j$$
5. Place the return circle and the corner circle around node i, such that the distance from the corner i to the return point is maximized.
6. Fix node i, and update the set $\mathcal{P}$ of fixed nodes:
$$\mathcal{P} := \mathcal{P} \cup \{i\}$$
7. If all the nodes are also in $\mathcal{P}$, terminate.
8. For all $\{\, j \,|\, j \notin \mathcal{P}, w_{ij} \neq \infty\}$:
   (a) Place a leave circle and a leave point for i on a straight line connecting i and j.
   (b) Check the feasibility of the connection. If the leave point does not lie between i and j, then reject the

connection, pick the next j, and go to 8a. If the path from i to j along the corner circle, leave circle, and the straight line i-j is not collision free, then reject the connection, pick the next j, and go to 8a.

(c) Update the temporary labels

$$D_j := \min(D_j, w_{ij} + D_i)$$

(d) If D_j is updated with $w_{ij} + D_i$, then set i as the successor of node j.

(e) Pick the next j and go to 8a.

9. Go to step 4 for next iteration

This procedure produces a tree of nodes with the goal node at its end. D_j gives the shortest distance from j to the goal along the straight line about which a kinodynamically feasible path is guaranteed to exist. Fig. 3 shows a typical scenario where many obstacles reside between the vehicle location (marked as the $*$ middle left) and the goal (marked as a star in the upper right). The dotted lines are the visibility graph and the thick lines are the tree of shortest paths obtained by running the modified Dijkstra's algorithm. In this example, a minimum turning radius $r_{\min} = 2.5$ is used. The corner A is not connected to the corner C since the path A-C-D requires tighter than dynamically allowable turns in order to avoid collisions, and hence this sequence would be kinodynamically infeasible. Corner A is also not connected to the tree E-F-C-D because the leave point of the connection E-A does not lie between the two nodes E and A due to the tight turns required for the tree E-F-C-D.

C. Cost Points in MILP

Although the cost map obtained in the previous section provides a tree of nodes along which a kinodynamically feasible path is guaranteed to exist, not all the nodes can be passed to the MILP optimization process as cost points or candidate visible points because the resultant trajectory does not necessarily follow the precedence of nodes. Fig. 4 illustrates that the violation of precedence can lead to an infeasible problem. Turning circles introduced in Fig. 2 are also shown around each corner A, B, and C. In this example, the modified Dijkstra's algorithm guarantees that the tree A-B-C-D is a feasible sequence, but the vehicle should not aim directly for corner C following the dashed line because it cannot turn around the corner C with its minimum turning radius.

Taking into account the precedence of cost points when generating the cost map solves this issue. In order to form a list of feasible cost points, the following algorithm is applied before calling the MILP optimization solver:

Step 1. Find all the points visible from the initial location of the vehicle and include them in the set of candidate cost points.

Step 2. For each visible point, check if the connection is feasible by considering the circles placed around the corner. From the candidate cost points obtained in the previous step, eliminate points that are visible but are not connectable with a dynamically feasible path. The remaining points form a list of cost points.

Step 3. Calculate the distance between the initial location and each cost point. If it is shorter than the length of the planning horizon, add successors of the cost point (one at a time) to the list of cost points until the tree starting from the cost point has one and only one node beyond the planning horizon. This keeps the number of nodes in the list of cost points as small as possible, and prevents the vehicle from skipping ordered nodes.

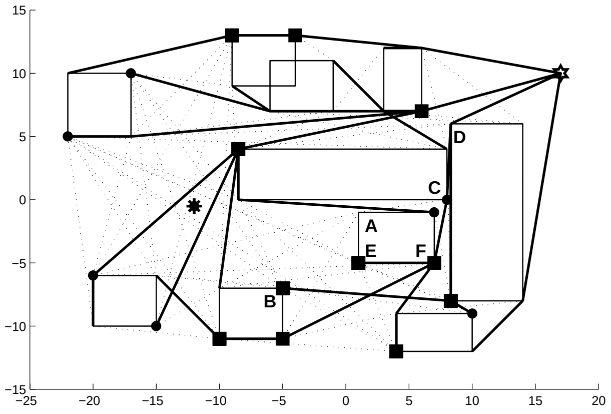


Fig.3: Typical scenario populated with obstacles, and the tree of shortest paths. Cost points used in MILP are marked with squares.

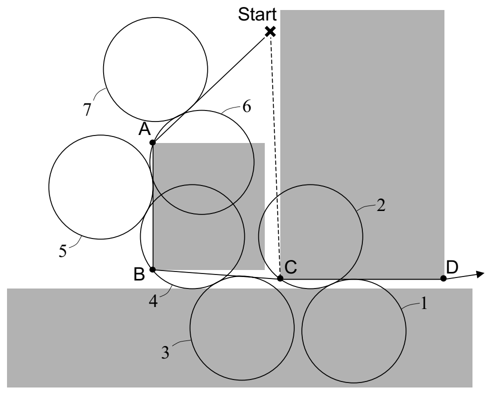


Fig.4: Precedence of cost points. The vehicle starts at the top ($\times$), and goes to the right of the figure by going through the narrow passage C-D. The corner circle (2) at C intersects the obstacle to its left, and cannot be directly connected to the start position.

Fig. 3 shows a resulting list of cost points after executing the algorithm above. Note that the points with $\bullet$ in the left of the figure are not considered to be feasibly connectable to the initial point, since a vehicle going from the initial point ($*$) to these nodes will require a sharp turn around the nodes in order to join the kinodynamically feasible tree. The planning horizon in this example has a length of 15 units. Point F has been added to the list in the operation of Step 3, but C was not included.

As described above, every time the receding horizon controller starts solving an optimization problem, only a limited number of points on each tree of cost nodes are extracted and used in the MILP. This ensures that the resultant trajectory is stable while retaining the freedom to choose the path along the way. Note that (for a static environment) the

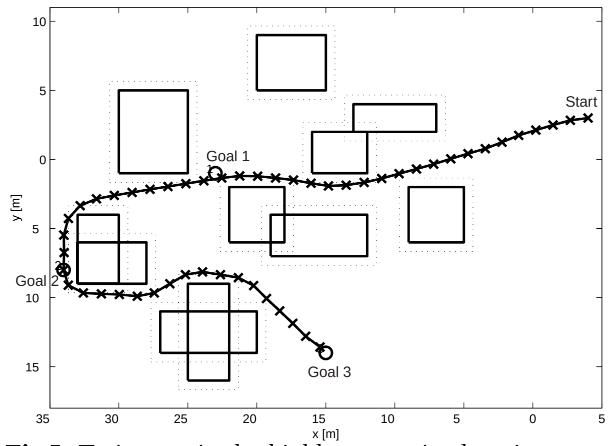


Fig.5: Trajectory in the highly constrained environment

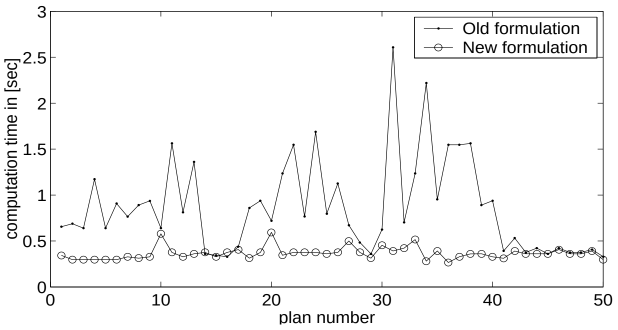


Fig.6: Comparison of the computation times.

stored cost map remains unchanged and this process is not computationally intensive.

D. Effect on the Computation Time

The node pruning algorithm presented above not only ensures that the problem is feasible, but it also reduces the computation load. Fig. 5 shows an environment densely populated with obstacles. The same optimal trajectory was obtained using the old and new (stable) formulations, and the vehicle goes through narrow passages. Fig. 6 compares the computation times of the two approaches. The plan reaches the final goal on the 50th steps. The line with $\cdot$ and the one with $\circ$ show the computation time of the old and stable formulations, respectively. As shown, there is a significant improvement in the computation time. Without pruning, there are 47 candidate nodes. If a node lies behind an obstacle, the visibility test rejects the node. This process involves a number of binary variables, and becomes computationally demanding as the number of obstacles increases. The stable formulation prunes most of these nodes before performing the MILP optimization, which results in a drastic reduction in the computation time. Note that this formulation only prunes the nodes that **will never be selected**, and still retains the freedom to choose the best path from the trees of nodes that remain.

III. Stability Proof

This section addresses the optimization process using RHC that designs detailed trajectories. In order to guarantee stability, several parameters of the receding horizon controller must satisfy the conditions, identified in Section III-A.

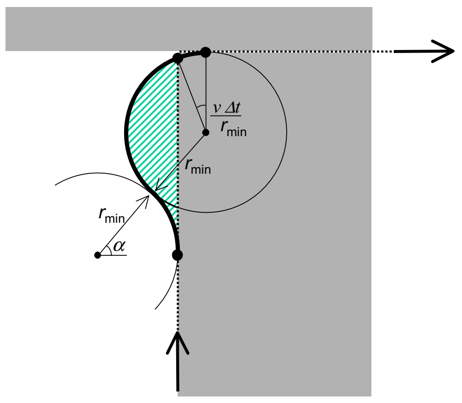


Fig. 7: Minimum margin to keep execution horizon in a feasible region (discrete time case).

A. Feasibility Criteria

The RHC has two horizons: the detailed trajectory, which is kinodynamically feasible, is designed over n_p steps of planning horizon; but only the portion in the execution horizon, which is the first n_e steps of the generated plan, is executed, and the RHC re-optimizes the trajectory beyond the execution horizon. However, since the straight lines used to approximate the cost beyond the planning horizon can be dynamically infeasible, this optimization process can fail [5]. This situation is successfully avoided by having a planning horizon that is relatively long compared to the execution horizon. The margin $(n_p - n_e)$ ensures the state on the execution horizon, which is the initial state of the next optimization problem, stays in the region from which a feasible path to the goal exists.

The margin $(n_p - n_e)$ has a lower bound in order to guarantee stability, but it should be minimized in order to keep the length of the planning horizon n_p small. This is because the complexity of MILP grows rapidly as n_p increases. The length of the execution horizon n_e also has a lower bound, which is derived from the computation time: the next plan must be generated before the vehicle finishes executing the current plan.

The minimum margin must be able to account for the discrepancy between the straight line approximation and the dynamically feasible path. The largest discrepancy occurs when the intersection of two line segments forms a 90° angle at the obstacle corner in our problems. Fig. 7 graphically shows the length of the minimum margin required to ensure that the state on the execution horizon never becomes infeasible if taken as an initial condition of the next plan. The vehicle enters from the bottom of the figure along the obstacle boundary and is aiming towards the right of the figure by going through the narrow passage. Once the vehicle passes through the thick line and enters the shaded portion, which is formed by two circles of radius $r_{\min}$, it cannot avoid a collision. The minimum margin $n_{m_{\min}}$ is geometrically calculated as the length of the thick line divided by the step size:

$$n_{m_{\min}} \geq \frac{r_{\min}}{v\Delta t}\left\{\frac{\pi}{2} + 2\arccos\left(\frac{1+\sin\left(\frac{v\Delta t}{r_{\min}}\right)}{2}\right)\right\} \quad (2)$$

The $n_{m_{\min}}$ is a function of the ratio of minimum turning radius $r_{\min}$ and step size $v\Delta t$. With the same step size, a larger minimum turning radius requires a larger margin since the vehicle is less agile. With the same minimum turning radius, a smaller $v\Delta\ t$ requires a larger number of steps as a margin since the waypoints on the trajectory are more detailed.

This analysis gives insight to the minimum length of the planning horizon that is often discussed but is rarely determined analytically. Note that a larger margin causes longer computation time; this represents a trade-off between the computation load and the resolution of the trajectory.

Simulation Result: The simulation result in Fig. 8 demonstrates that the optimization problem solved by the RHC remains feasible over the worst case turn. Original obstacles are depicted by solid lines, while dashed lines show obstacle boundaries expanded to account for the discrete time model. Solid lines with bullets show the detailed trajectory, and dotted lines with bullets show the waypoints of the previous plan. The vehicle starts at the bottom of the figure and goes to the upper right by passing through the narrow passage. From (a) to (d), the vehicle executes one step at a time. A circle introduced in Fig. 7 is placed to show the infeasible region. The following parameters are used in this simulation.

- n_p = 11
- n_e = 1
- $r_{\min}$ = 3.1
- $v\Delta t$ = 1
- $n_{m_{\min}} \geq 10.0$

The planning horizon of the next plan enters the narrow passage after executing one step, as shown in (a). However, it cannot proceed until the heading direction of the terminal step is aligned to the narrow passage as in (c). The resultant trajectory follows the circle of radius $r_{\min}$ as expected in Fig. 7. This result shows that with the minimum margin obtained from Eq. 2 the vehicle always stays in a region from which a feasible path to the goal is guaranteed to exist. Note that this scenario goes infeasible with the old formulation, or with a shorter margin.

B. RHC Stability

Finite Time Completion: The modified Dijkstra's algorithm ensures the existence of a kinodynamically feasible path around the line segments from each cost point to the goal. As shown in the previous section, when turning a corner, the vehicle will deviate from the straight line path. In order to obtain a feasible path, however, the vehicle is required to satisfy a condition that the vehicle joins the next straight line before the return point. The minimum margin $n_{m_{\min}}$ discussed above enables the RH-MILP to find the optimal trajectory while satisfying this condition. This discussion demonstrates that the receding horizon optimization problems are always feasible.

Although the UAV moves at a constant speed v, the point on the planning horizon can move less than $v\Delta t$, as shown in Figs. 8(a) to (c), when a planned trajectory does not follow the previous plan. This is caused by a discrepancy between the straight line approximation and the vehicle dynamics beyond the planning horizon. Once the heading discontinuities

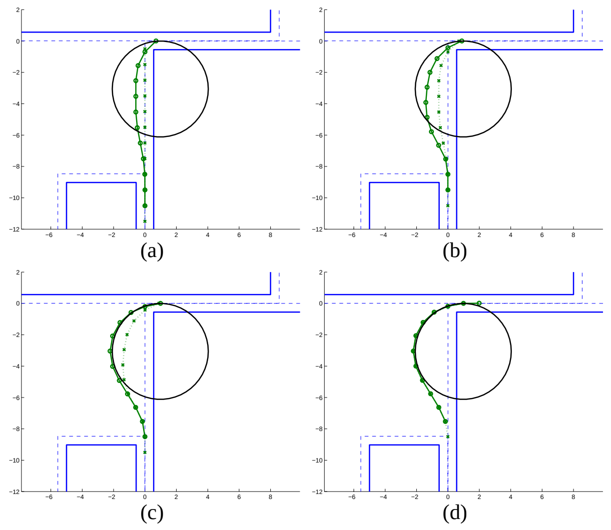


Fig. 8: Worst case turn. The straight line approximation requires a 90 degree turn at the corner.

are resolved, however, the point on the planning horizon starts moving towards the goal again (Figures 8(c) and (d)).

The foundation of the proof of finite time completion is obtained from the following arguments. First, the sum of the straight line lengths from the start point to the goal is calculated as the total path length. Then, all of the differences ΔJ between the arc lengths and the straight line lengths, as expressed in Eq. 1, are added to the total path length. This path length from the initial point to the goal along a feasible path gives an upper bound $J_{\max}$ of the length of the optimal trajectory. Since the vehicle moves at a constant speed v, it will enter a circle of radius $v\Delta t$ around the goal in at most $k_{\max}$ steps, where

$$k_{\max} = \text{floor}\left(\frac{J_{\max}}{v\Delta t}\right).$$

Simulation Result: Fig. 9 shows an optimal trajectory for a scenario where two tight turns are required and Fig. 10 shows the cost-to-go of each RHC optimization and the pre-calculated upper bound. Note that the difference between the two lines for the first plan is the sum of the difference ΔJ at each corner.

If the generated plan is the same as a straight line, then the cost-to-go decreases by $v\Delta\ t$ at each time step. When the planning horizon comes to the obstacle boundary $y = 0$, the horizon point cannot proceed until the heading direction at the terminal point is aligned to the direction of the straight line approximation. Thus, the decrease in the cost-to-go during plan numbers 6–8 in Fig. 10 is quite small. When the vehicle makes another turn in plans 16–18, the decrease in cost-to-go is also less than $v\Delta t$. However, the cost-to-go is bounded by the straight line $J(k) = J_{\max} - k(v\Delta\ t)$, which constantly decreases by $v\Delta\ t$ for each step and would eventually be less than zero. Having $J(k) < 0$ is a contradiction, so the cost-to-go must be less than $v\Delta t$ before the upper bound hits zero. This implies that the vehicle will enter inside a circle of radius $v\Delta t$ around the goal in finite time [5].

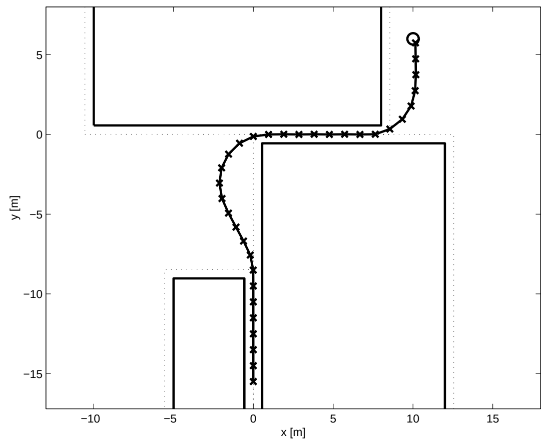


Figure 9: Trajectory

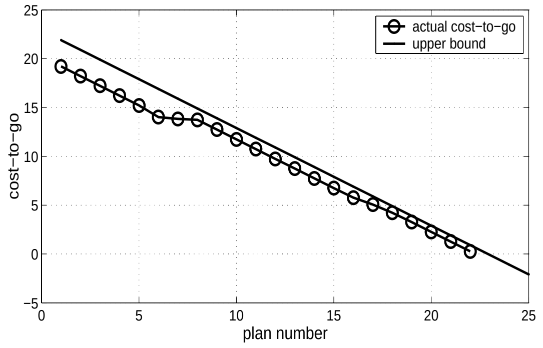


Figure 10: Decrease of cost-to-go

C. Selection of Better Path

Once there is a large discrepancy between a straight line approximation and a dynamically feasible path, the decrease of cost-to-go becomes smaller than the nominal $v\Delta t$ per step. In such situations, the stable RHC can choose another path on its way to the goal. Note that this formulation allows the controller to select only feasible paths, as shown in Fig. 3. In Fig. 11, the vehicle originally chooses to pass through the narrow passage based on a cost estimate using straight lines (trajectory points marked with $\times$). However, going around the corner prevents the planning horizon from proceeding and does not reduce the cost-to-go along the path. Then the controller makes a different decision on the selection of the visible point, and as a result the vehicle goes around the upper obstacle (trajectory points marked with $\circ$). Note that the controller selected another path simply because the cost-to-go along the new path is smaller than the first, and the cost-to-go of the actual vehicle trajectory is still bounded by the same straight line $J(k) = J_{\max} - k(v\Delta t)$.

IV. Conclusions

This paper presented a new algorithm that stably navigates the vehicle to the goal. The vehicle dynamics are taken into account as a minimum turning radius in the cost estimation phase. By placing turning circles around the obstacle corners, the modified Dijkstra's algorithm finds node sequences along which a feasible path to the goal exists. The pruning algorithm eliminates unnecessary nodes to form a stable cost map, without losing the freedom to choose better paths

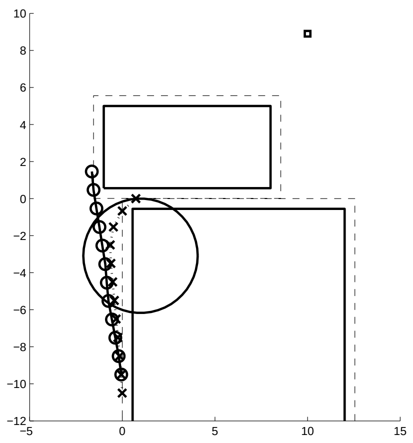


Fig.11: Selection of another path. The vehicle is at the lower left, and the goal is marked with a small rectangle shown in the upper right of the figure.

in the detailed part of the trajectory design. This process was demonstrated to significantly reduce the computation time. It was also shown that the receding horizon controller must extend the planning horizon beyond the execution horizon in order to guarantee the stability of the trajectory optimization. The lower bound of the margin required was analytically calculated. Combined with the stable cost map, this formulation proved that: 1) the RHC always has a feasible solution, and 2) the vehicle reaches the goal in finite time. The simulations verified these results.

References

[1] J. C. Latombe, *Robot Motion Planning.* Kluwer Academic, 1991.

[2] A. Richards and J. How, "Aircraft Trajectory Planning With Collision Avoidance Using Mixed Integer Linear Programming," in *Proceedings of the IEEE American Control Conference*, May 2002.

[3] T. Schouwenaars, E. Feron, and J. How, "Hybrid Model for Receding Horizon Guidance of Agile Maneuvering Autonomous Rotorcraft." 16th *IFAC Symposium on Automatic Control in Aerospace,* 2004.

[4] J. Bellingham, A. Richards, and J. How, "Receding Horizon Control of Autonomous Aerial Vehicles," in *Proceedings of the IEEE American Control Conference*, May 2002.

[5] J. Bellingham, Y. Kuwata, and J. How, "Stable Receding Horizon Trajectory Control for Complex Environments," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference,* Aug 2003.

[6] Y. Kuwata, "Real-time Trajectory Design for Unmanned Aerial Vehicles using Receding Horizon Control," Master's thesis, Massachusetts Institute of Technology, 2003.

[7] A. Bicchi and L. Pallottino, "On Optimal Cooperative Conflict Resolution for Air Traffic Management Systems," *IEEE Trans. on Intelligent Transportation Systems*, vol. 1-4, pp. 221–231, Dec 2000.

[8] R. G. Cleggs, "Bellman-Ford and Dijkstra's Algorithm," tech. rep., University of York.