Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Integrating Planning and Control for Efficient Path Planning in the Presence of Environmental Disturbances

Sandip Aine and P. B. Sujit
Indraprastha Institute of Information Technology Delhi (IIIT-Delhi)
New Delhi - 110020, India.
sandip+sujit@iiitd.ac.in

Abstract

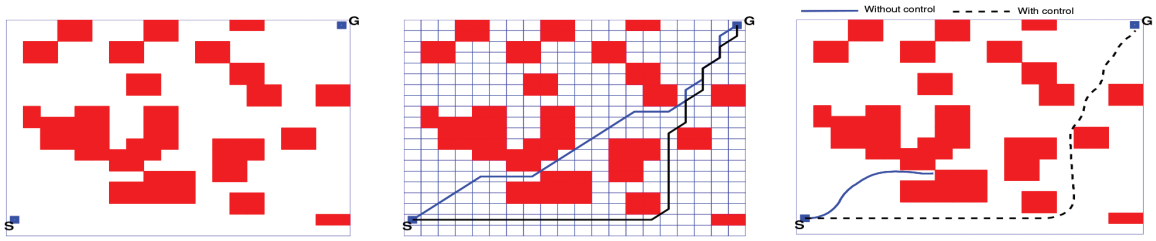
Path planning for nonholonomic robots in real-life environments is a challenging problem, as the planner needs to consider the presence of obstacles, the kinematic constraints, and also the environmental disturbances (like wind and currents). In this paper, we develop a path planning algorithm called Control Based A* (CBA*), which integrates search-based planning (on grid) with a path-following controller, taking the motion constraints and external disturbances into account. We also present another algorithm called Dynamic Control Based A* (DCBA*), which improves upon CBA* by allowing the search to look beyond the immediate grid neighborhood and thus makes it more flexible and robust, especially with high resolution grids. We investigate the performance of the new planners in different environments under different wind disturbance conditions and compare the performance against (i) finding a path in the discretized grid and following it with a nonholonomic robot, and (ii) a kinodynamic sampling-based path planner. The results show that our planners perform considerably better than (i) and (ii), especially in difficult situations such as in cluttered spaces or in presence of strong winds/currents. Further, we experimentally validate the approach using a quadrotor in the outdoor environment.

Introduction

Path planning is an important problem in mobile robotics, where, a robot needs to find a collision free path between a given source and goal positions in the presence of obstacles. Several applications like surveillance, monitoring, etc. need collision free paths to perform theirs tasks successfully. Over the last two decades, several flavors of path planning techniques have been developed, catering for different sensor, environmental, and motion constraints. As the robots have kinematic constraints (such as minimum turning radius), determining control policies to steer them towards the goal in the continuous domain with obstacles is often difficult (LaValle 2006). Planning becomes even more complicated when there are environmental disturbances, such as winds and currents, that may considerably impact the control policies. In this work, we focus on the planning problem where a robot needs to compute a minimum cost collision free path adhering its kinematic constraints in presence of environmental disturbances.

The generic path planning algorithms in the literature can be broadly classified into three main categories, i) optimal control based approaches (Barraquand and Ferbach 1994; Kimmel, Amir, and Bruckstein 1995; Mitchell and Sastry 2003), ii) search based approaches (Likhachev, Gordon, and Thrun 2004; Likhachev and Ferguson 2009) and iii) sampling based approaches (Karaman and Frazzoli 2010; Kavraki et al. 1996; Lavalle and Kuffner 2000). Unfortunately, all these approaches have deficiencies when we consider the presence of external disturbances in addition to obstacles and kinematic constraints. Most optimal control based algorithms are time consuming and require significant computational power that the robots may not have, and hence are not suitable for large scale real life applications. On the other hand, most search based planners require an offline discretization of the planning space, either grid based (Likhachev, Gordon, and Thrun 2004) or lattice based (Likhachev and Ferguson 2009). Grid based search algorithms do not handle kinematic constraints efficiently. The lattice based planners rectify this problem by precomputing *motion primitives*, which are short, kinematically feasible moves. However, even this approach fails in presence of external disturbances as pre-computed *motion primitives* may not be universally viable (a motion primitive feasible for a particular wind speed and angle, may not apply for another wind speed). The sampling based planners do not require specific discretization of the environments, and thus are more suitable for planning in continuous domains. However, these planners generally focus on finding a feasible trajectory, rather than minimizing the path cost, and therefore, the paths computed are often sub-optimal and inconsistent. Moreover, in the presence of external disturbances, the dynamics of the environment can change substantially affecting the performance.

A naive approach to plan in presence of kinematic constraints and environmental disturbances can be to use a standard grid-based planner (such as A*) to generate a collision free path and ask the robot to follow the path using a control algorithm. Since the path generation was open-loop and the planner did not take the motion constraints of the robot nor the effect of environmental disturbances into account, the produced path may not be realizable, or the path cost may vary significantly from the cost computed on the grid. For example, consider the scenario as shown in Fig-



(a) (b) (c)

Figure 1: An example of path planning in presence of obstacles, kinematic constraints and environmental disturbances. Figure 1a shows the original environment with start and goal states. The vehicle velocity is 5m/s, and the current velocity is 2.5 m/s blowing from left to right. Figure 1b shows a discretization of the environment in 5m×5m grid, and two planned paths, one by grid based A* (path shown in blue) and the other using integrated planning and control (path shown in black). Figure 1c shows the resulting continuous control paths when the original planned paths were given to the actual controllers. The blue path is not realizable as the control policy cannot satisfy the constraints in the presence of wind, whereas the integrated strategy computes a feasible path.

ure 1a, where, the robot needs to navigate from 'S' to 'G' avoiding the polyhedral obstacles. The blue path shown in Figure 1b is computed by running an A* search on a discretized 8-connected grid obtained from the original environment (shown in Figure 1a). Now, while this path is cost optimal on the underlying grid, it is not a feasible path when we consider the kinematic and environmental constraints, as a result the controller cannot convert this discretized path into a control sequence that takes the robot to the goal and returns with a failure.

We present a relatively simple solution to this problem, we propose a search based planning algorithm called Control Based A* (CBA*) that works on an underlying grid to compute a path in terms of grid cells, but when performing the search, it uses a control algorithm to determine exact cost of the intended traversal considering all the factors, namely, kinematic constraints, environmental disturbances and presence of obstacles. target point (mid-point of the targeted grid cell), and attempts to find a feasible sequence of control steps that can take the robot from the current position to the intended position. If the robot can reach any point on the target grid (not necessarily the exact mid-point), we consider that to be a feasible trajectory and in that case, the controller returns the exact cost of this traversal to the search algorithm. On the other hand, if such a control sequence cannot be computed (due to constraints or collisions), the controller lets the planner know that this transition is not feasible by setting the cost to ∞. Figure 1 shows an example of integrated planning and control, the black line in Figure 1b shows a discrete path on grid computed by CBA*. As this path is computed considering all the constraints, it can be realized by the actual controller (as shown by the black contour in Figure 1c).

We also propose a second algorithm, called Dynamic Control Based A* (DCBA*), which follows the same principle as CBA*, but improves the chances of convergence by dynamically discovering additional feasible connections (not restricted to grid neighbors) and using them to search. DCBA* provides a more robust approach to compute a feasible path, especially for high-resolution grids where connecting to an immediate grid neighbor may not always be that easy due to kinematic/environmental constraints.

On the theoretical side, we show that the algorithms provide the usual guarantees on completeness/optimality (on the underlying graph), like most search based planners. In addition, the paths obtained by CBA* and DCBA* are always realizable by the actual controller with the exact same costs reported by the search. On the experimental side, we test the proposed algorithms for different environments parameterized by obstacle densities, environmental disturbances (wind velocities), and grid resolutions. Comparison with the naive grid search planner and a randomized kinodynamic planner (LaValle and Kuffner 2001), shows that our planners are more robust (in terms of success rates) and they generally produce better quality solutions, with the performance gap increasing in adverse conditions (harsh weather/cluttered spaces). Finally, we carry out a field experiment on a quadrotor to demonstrate the usability of our algorithm for real-world applications.

Background and Related Work

Earlier, we broadly classified the path planning algorithms into three categories. In this section, we review several techniques in each category relevant to the proposed approach.

Optimal path planning problem

Given a source location $X_0 = (x(0), y(0))$ and the goal location $X_g = (x(t_g), y(t_g))$ along with the obstacle space and region of operation, the problem of finding an optimal path from X_0 to X_g can be formulated as a optimal control problem. A typical performance index for the optimal control problem is to minimize travel time as we assume the velocity of the vehicle to be constant,

$$J = \int_{t_0}^{t_g} dt, \tag{1}$$

subject to the kinematic constraints given in equation 2 and the boundary condition that at time t_g, the vehicle position

$p = X_g$. The robot kinematic constraints and environmental disturbances are modeled using a Dubin car model (Dubins 1957)

$$\begin{aligned} \dot{x} &= v_a \cos\psi + v_w \cos\psi_w, \\ \dot{y} &= v_a \sin\psi + v_w \sin\psi_w, \\ \dot{\psi} &= u, \end{aligned} \tag{2}$$

where, v_a is the speed of the robot, v_w is the wind or currents speed at angle ψ_w (environmental disturbance), u is control that determines the desired steering angle rate and is constrained as $-\omega_{\max} \leq u \leq \omega_{\max}$, ψ is the current heading angle, and $p = \{x, y\}$ is the current location of the vehicle.

Solving the optimal control in the continuous domain is difficult and hence numerical techniques such as level sets (Kimmel, Amir, and Bruckstein 1995), fast marching methods (Mitchell and Sastry 2003), dynamic programming (Barraquand and Ferbach 1994), and navigation functions (Rimon and Koditschek 1992) are used. These techniques are typically computationally intensive and are not scalable for obstacle rich environments. Some modifications like using a pre-computed table for implementation can be performed (Barraquand and Ferbach 1994), but not always feasible for obstacle rich environments. Furukawa et al. (Furukawa et al. 2004) proposed a time optimal trajectory for UAVs by parameterizing control and time discretization, which is then solved using sequential quadratic programming technique. Yang and Kapila (Yang and Kapila 2002) propose a optimal planning technique to visit several target locations using a combination of arcs and straight lines. However, both the approaches in (Furukawa et al. 2004; Yang and Kapila 2002) do not consider the presence of obstacles and disturbances.

Search based planning

Heuristic search algorithms have been extensively used for path planning in robotics, as they are easy to understand and implement, and they guarantee important theoretical properties like completeness/optimality. The classical planning technique is based on A* (Hart, Nilsson, and Raphael 1968), which is a provably optimal algorithm. There are several variants of A*, like incremental (Koenig and Likhachev 2002; Aine and Likhachev 2013), anytime (Likhachev, Gordon, and Thrun 2004; Aine, Chakrabarti, and Kumar 2007), and multi-heuristic (Aine et al. 2014) searches, that are used for various applications, such as motion planning for ground (Likhachev and Ferguson 2009) and air (MacAllister et al. 2013) vehicles, high-dimensional manipulation (Narayanan, Aine, and Likhachev 2015), footstep planning (Hornung, Maier, and Bennewitz 2013; Zucker et al. 2011), and etc.

Heuristic search is essentially a discrete optimization algorithm. Thus, for motion planning problems (which are inherently continuous), the planners need to discretize the search space. The easiest way to discretize a space is to partition it into cells, forming a grid. This requires no knowledge about the kinematic constraints or the control strategies, and thus can be done without extra overheads. However, as shown in Figure 1 a path obtained on a grid may not be feasible (due to kinematic constraints/disturbances). In (Barraquand and Latombe 1993), the problem of satisfying kinematic constraints was (partially) addressed by using control based discretization to generate the search space, which then led to the development of lattice based planners (Likhachev and Ferguson 2009), where the actions used to get successors for states are a set of *motion primitives*, that are short kinematically feasible motion sequences. These motion primitives are pre-computed, and during planning they are applied directly to a robot configuration to generate the successor configurations. However, even this approach suffers from two shortcomings. First, it is non-trivial to compute an effective set of motion primitives, as it targets a balance between search space size and quality of results. More importantly, computing motion primitives offline becomes infeasible when we consider environmental disturbances such as winds and currents. As these parameters can only be known at runtime pre-computed motion primitives cannot be used directly to construct a feasible control sequence. In (Dolgov et al. 2008), a search algorithm is proposed (Hybrid State A*) that maps the continuous states to discrete buckets instead of specific points, we adopt the same policy in CBA*/DCBA*. However, our algorithms differ from Hybrid State A* in terms of i) exploration, ii) control models and iii) efficiency.

Sampling based planning

An alternative approach to path planning is adopted by the sampling based planners (Kuffner and LaValle 2000; Kavraki et al. 1996; Lavalle and Kuffner 2000). These planners do not require specific discretization of the environments, and thus are more suitable for planning in continuous domains, also these methods can easily be integrated with different kinodynamic controllers (LaValle and Kuffner 2001). In general, these planners focus on finding any feasible trajectory, rather than minimizing the cost of the solution with the exception of the RRT* (Karaman and Frazzoli 2010) (and related algorithms), which asymptotically converges to an optimal solution. As a result, the paths computed by these planners are often sub-optimal. More importantly, to our knowledge, none of these planners take the effect of environmental disturbances into account, which can significantly alter the dynamics of the system and thus can affect the underlying assumptions of the planning algorithms. For example, the RRT based algorithms like RRT-Connect (Lavalle and Kuffner 2000) (with or without kinodynamic constraints), RRT* (Karaman and Frazzoli 2010), and etc, attempt to connect a newly found point (in the free space) to the closest point in the explored horizon (the points to which a path has already been discovered). Now, this closest point in terms of physical distance may not be the *easiest* one to connect to when we consider a strong wind or current. On the other hand, to find the easiest point to connect to is not trivial to compute and may considerably increase the overhead. Therefore, while most of the sampling based algorithms can be applied seamlessly to the problem of path planning considering obstacles, kinematic and environmental constraints, their performance may degrade in presence of disturbances.

Algorithm 1 NLGL method (Park, Deyst, and How 2007)

Input: $x, y, \psi, C_{xy}, C'_{xy}, v_a, L, \Delta t, v_w, \psi_w$
Output: x', y', ψ' ▷ : Configuration after Δt
Determine $q = (x_t, y_t)$ ▷ : Target position
$\Theta = atan2(y_t - y, x_t - x), \eta = \Theta - \psi, u = 2v_a^2 \sin(\eta)/v_a$
if $u > 0$ **then**
    $u = \min(u, \omega_{\max})$
else
    $u = \max(u, -\omega_{\max})$
$x' = x + (v_a \cos\psi + v_w \cos\psi_w)\Delta t$
$y' = y + (v_a \sin\psi + v_w \sin\psi_w)\Delta t$
$v_g = \sqrt{(v_a \cos\psi + v_w \cos\psi_w)^2 + (v_a \sin\psi + v_w \sin\psi_w)^2}$
$\psi' = \psi + u * \Delta t / v_g$

Algorithms

In this section, we describe the proposed algorithms and their properties. The flow of our approach is fairly simple. For our planner, a given path computation problem is defined using a start configuration (x, y, ψ) and end configuration (x_t, y_t) for the robot, the obstacle co-ordinates, the vehicle's velocity v_a, and the wind (or current) velocity (v_w) and direction ψ_w. We assume an underlying grid where each cell is a square of certain size, a cell is assumed to be navigable if all the obstacles are out of its boundary, else it is termed blocked. All blocked cells are included in the obstacle set $\mathcal{O}$. We also use a time unit Δt, which is the resolution used by the control algorithm. Given this formulation, we design the planner with two components, search and control. The search component works on the underlying grid trying to compute a path (a sequence of cells) from the start to end configuration, whereas the control component works on the actual configuration space computing the exact cost to move the vehicle from a given cell to a target one, feeding the result to the search component.

Control Module for Connecting Cells

The controller determines the feasibility of constructing a path segment from the current cell C to the target cell C' and computes the cost of that path, i.e., it starts with a given configuration that belongs to the cell C and tries to find a sequence of control steps that takes the vehicle to a configuration belonging to the target cell C'. We treat this path segment as two way-points - $\{C_{xy}, C'_{xy}\}$, where C_{xy} determines the x, y cell center coordinates for a given cell C. In order to follow a path given by the way-points, we can use any path following controllers (for example (Sujit, Saripalli, and Sousa 2014)). In this paper, we use nonlinear guidance law (NLGL) due to its simple implementation (Park, Deyst, and How 2007). The path following controller can handle constant and random wind disturbances which has been tested in (Sujit, Saripalli, and Sousa 2014).

The NLGL method uses a virtual target technique, where a virtual target is placed on the path towards C'_{xy} and the vehicle must follow the virtual target. Consider the scenario as shown in Figure 2, where q is the virtual target and the angle η is the angle that the vehicle needs to move towards q. The virtual target q is obtained by drawing a circle of radius

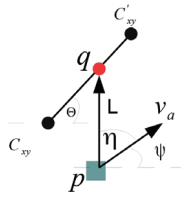


Figure 2: The geometry of determining η for a given L, v_a, ψ, C_{xy} and C'_{xy} using NLGL

Algorithm 2 PathCost Procedure

1: **procedure** FOLLOWPATH(x, y, ψ, C, C', cost)
2: Input: $v_a, L, \Delta t, \mathcal{O}$
3: Output: x', y', ψ', cost
4: $[x', y', \psi'] \leftarrow$ Execute Algorithm 1
5: xCell = Cell (x', y')
6: **if** $xCell \in \mathcal{O}$ **then**
7: cost= ∞
8: **else**
9: cost = cost $+ v_a \Delta t$
10: **procedure** PATHCOST(x, y, ψ, C, C')
11: Input: $v_a, L, \Delta t$
12: Output: x, y, ψ, cost
13: **while** $\|(x, y) - C'_{xy}\| \leq v_a \Delta t$ **do**
14: $[x, y, \psi$,cost] = FOLLOWPATH $(x, y, \psi, C, C'$, cost)
15: **if** cost $== \infty$ **then**
16: break

L from p and the circle intersects the path segment at q. The circle may intersect at another point also, in which case, the point that is closest to C'_{xy} is selected. The angle between the way-points is Θ.

Given a current and a target configuration (in terms of way-points), the NLGL algorithm in Algorithm 1 will update the position and heading angle of the vehicle for time Δt. However, to compute a path we may require multiple such collision-free steps. Moreover, for such a given path segment, we need the associated travel cost in terms of distance. The $PathCost$ routine (Algorithm 2), determines the path cost for such a path segment. It uses the $FollowPath$ function to determine the distance cost per Δt. The $FollowPath$ function executes the NLGL algorithm (Algorithm 1) to determine the new location and heading of the vehicle. The new vehicle position may be inside obstacles, hence we check if the new position is in the obstacle cell set $\mathcal{O}$ (line 6). If the new position is not in $\mathcal{O}$, then the cost is updated based on the distance traveled (line 9), otherwise ∞ is assigned to cost (line 7). Now, the $PathCost$ repeatedly calls the $FollowPath$ function with updated configurations (x, y, ψ) till it reaches the desired cell or hits an obstacle. If it reaches the desired cell (as close to the targeted midpoint as possible), the function returns the exact cost of this path (here, in terms of distance traveled, however any other cost function will be equally applicable), else the cost is set to ∞ indicating that such a path cannot be constructed following this control policy.

Algorithm 3 Control Based A*

```
1:  procedure EXPANDSTATE(s)
2:      C = Cell(s)
3:      for all neighbors C' of C on the grid do
4:          if (C' ∉ O) and (State(C') ∉ CLOSED) then
5:              [x, y, ψ, cost] = PATHCOST (x, y, ψ, C, C')
6:              if cost ≠ ∞ then
7:                  s' = CreateState(x, y, ψ, C')
8:                  g(s') = g(s) + cost
9:                  f(s') = g(s') + h(s')
10:                 bp(s') = s
11:                 if f(State(C')) > f(s') then
12:                     delete State(C')
13:                     State(C') = s'
14:                     update OPEN repositioning State(C')
15:                 else
16:                     delete s';
17: procedure CBA*
18:     OPEN = ∅; CLOSED = ∅
19:     g(s_start) = 0; bp(s_start) = null
20:     f(s_start) = h(s_start)
21:     insert s_start into OPEN with f(s_start)
22:     while OPEN is not empty do
23:         remove s with smallest f-value from OPEN
24:         insert s into CLOSED
25:         if s == s_goal then
26:             CONSTRUCTPATH; Return
27:         EXPANDSTATE (s)
```

Search Module for Constructing Paths

The search module computes a sequence of grid cells (path) that connects the start position to the goal position. The first algorithm (Control Based A*) we propose is a simple A* like search performed on the underlying grid (8-connected), the difference being that while the search is done on the grid the actual vehicle configurations (x, y, ψ) and the related costs are obtained using control algorithm (Algorithm 2).

In Algorithm 3, we include a pseudocode for Control Based A* (CBA*). CBA* follows the usual A* way of expanding states in an increasing order of their f-values (where $f = g + h$) till it expands the goal state (s_{goal}) at line 26 or there is no state to expand at line 22 (this means the search ends in a failure). In our case, each search state contains a vehicle configuration (x, y, ψ). From this configuration we can directly compute the grid cell $C(cur)$ the vehicle belongs to. When expanding a state we use the control algorithm to find out the exact cost of visiting a particular neighbor along with the configuration (line 5), and let the search use this cost.

The primary difference between a standard A* search and CBA* comes in the expansion part (lines 6-16). As we are using the configuration based control algorithm to connect two grid positions (say C and C') which essentially works in the continuous domain, we may have several states (with different configurations) for a given grid cell. This can easily blow up the search space, severely affecting the performance (memory/runtime). To counter this, in CBA*, we always store a single state corresponding to a particular grid cell, if a new state s_2 is discovered during the search for a given cell

Algorithm 4 DynamicPathCost Procedure

```
1:  procedure DYNAMICFOLLOWPATH(x, y, ψ, C, C', cost, V)
2:      Input: v_a, L, Δt, O
3:      Output: x', y', ψ', cost, V
4:      [x', y', ψ'] ← Execute Algorithm 1
5:      xCell = Cell (x', y')
6:      if xCell ∈ O then
7:          cost= ∞
8:      else if xCell ≠ C' then
9:          Store [x', y', ψ'] and cost in V
10:     else
11:         cost = cost + v_a Δt
12: procedure DYNAMICPATHCOST(x, y, ψ, C, C')
13:     Input: v_a, L, Δt
14:     Output: A vector V of configurations (x, y, ψ, cost)
15:     V = ∅‘
16:     while ||(x, y) − C'_xy|| ≤ v_a Δt do
17:         [x, y, ψ,cost,V] = DYNAMICFOLLOWPATH
        (x, y, ψ, C, C', cost, V)
18:         if cost == ∞ then
19:             break
20:     include [x, y, ψ, cost] in V
```

C which already has a valid state s_1 (i.e., $state(C) = s_1$), we compare the f-values of s_1 and s_2 and only store the state with minimum f-value (lines 11-16). This way we ensure that the possible number of states in CBA* is never more than the number of states that can be explored for an A* without any control information (for a given grid).

One thing to note here is that in A*, if two paths are discovered to a given state the path with best g cost is stored (using the bp pointer). However, in CBA*, many configurations can belong to a given grid, thus, we store the state with best f value. This can be seen as a generalization of A*, if the h values for two such states are the same (for example, if the h is computed on the grid) then this condition reduces to the A* way of storing best g, however, if the h values also differ (for example, if we use Euclidean distance as h), then we opt for the state with best f rather than the best g.

In CBA*, we integrate continuous control with discrete planning, but it does not maximally utilize the information obtained from the control component. For example, let us consider the case where we use Algorithm 2 to compute the cost moving the vehicle from one cell C to a neighboring cell C'. If this procedure can find a realizable control path to cell C' from the current robot configuration, it will return a finite cost, otherwise it will return ∞. However, it may happen that while trying to reach C', we reach another cell C'' (i.e., obtain a configuration that lies in cell C''). This intermediate cell information is not used in CBA*, although it can help the planner to construct a feasible path to s_{goal}. Such flexibility can be very useful when we are planning at a high resolution, where reaching the exact neighbors may be difficult for the control algorithm because of (strong) wind/current and/or turning radius constraints.

Next, we present another algorithm called Dynamic Control Based A* (DCBA*) which improves upon CBA* by expanding the grid neighborhood of CBA* to include the other cells visited while reaching the neighborhood cells. For this,

Algorithm 5 Dynamic Control Based A*

```
1:  procedure EXPANDSTATE(s)
2:      C = Cell(s)
3:      for all neighbors C' of C on the grid do
4:          if (C' ∉ O) and (State(C') ∉ CLOSED) then
5:              V = DYNAMICPATHCOST (x, y, ψ, C, C')
6:              for all C'' (with cost c'') in V do
7:                  if (c'' ≠ ∞) and (State(C'') ∉ CLOSED)
    then
8:                      s' = CreateState(x, y, ψ, C'')
9:                      g(s') = g(s) + c''
10:                     f(s') = g(s') + h(s')
11:                     bp(s') = s
12:                     if f(State(C'')) > f(s') then
13:                         delete State(C'')
14:                         State(C'') = s'
15:                         update State(C'') in OPEN
16:                     else
17:                         delete s';
18: procedure DCBA*
19:     OPEN = ∅; CLOSED = ∅
20:     g(s_start) = 0; bp(s_start) = null
21:     f(s_start) = h(s_start)
22:     insert s_start into OPEN with f(s_start)
23:     while OPEN is not empty do
24:         remove s with smallest f-value from OPEN
25:         insert s into CLOSED
26:         if s == s_goal then
27:             CONSTRUCTPATH; Return
28:         EXPANDSTATE (s)
```

we modify the FollowPath routine (line 1, Algorithm 2) to store information about the intermediate cells visited. Subsequently, the PathCost routine is now changed to return a vector of configurations (along with costs) that can be realized, instead of a single configuration and cost. The routine is included in Algorithm 4, where the important changes are in lines 8-9, which stores all the intermediate *reachable* cell information. The rest of the algorithm is essentially similar to the CBA* (Algorithm 5). The only changes are in lines 4-17, that show that in DCBA*, when we expand a state, we consider all the states that are visited during the call for finding cost, which may include state beyond its immediate grid neighborhood. Similar to the earlier case, here also we never store (or expand) more than one state per grid cell. However, in DCBA*, the number of successors of a state can be more than CBA*, as in CBA*, only the immediate neighbors are considered as possible successors.

The following theorems discuss some of the properties of CBA*/DCBA*. These state that CBA*/DCBA* mostly follow the properties of A* with a consistent heuristic function (Pearl 1984) in terms of solution quality and expansion/storage complexity (Theorems 1 and 2). In addition, the handshaking between the discrete search with continuous control ensures that the path computed using CBA*/DCBA* is realizable with the same cost as reported by the algorithm.

Theorem 1. *If the heuristic function is consistent, both CBA* and DCBA* produces an optimal solution on the discrete graph it constructs. In other words, the path computed by CBA*/DCBA* is optimal on the underlying grid if we consider the actual realization costs.*

Proof Sketch: If we consider the discrete space, this theorem directly follows from the properties of A* (Hart, Nilsson, and Raphael 1968; Pearl 1984) with a consistent heuristic function. Note that, if we replace the cost finding routine in CBA*/DCBA* with the grid distance, CBA*/DCBA* reduces to pure A* on grids. The control part helps the search ascertain accurate costs of the grid based transitions. □

Theorem 2. *In CBA*/DCBA*, a) at most one state per grid cell is stored in memory and b) a state is expanded at most once.*

Proof Sketch: Whenever a new configuration is discovered, both CBA*/DCBA* checks whether there is another state visited which belongs to the same grid cell in memory, if such a state exists and it is already expanded (i.e., in CLOSED), they do not consider the new state (checks at line 4, Algorithm 3 and lines 4-7, Algorithm 5). If the corresponding state is not expanded then CBA*/DCBA* either keeps the earlier state or replaces it with the new one depending on their f-values. □

Theorem 3. *In CBA*/DCBA*, a solution obtained on the grid is always realizable in the continuous space (if all other conditions remain unchanged) with the same cost as reported by CBA*/DCBA*.*

Proof Sketch: This is guaranteed by the integration of control along with the planning part, whenever CBA*/DCBA* sets the back-pointer for a state to another state (parent) (eg. line 11 in Algorithm 5), it ensures that there is a control sequence which translates the vehicle from the parent configuration to the state configuration with exactly the same cost as it reports. □

It should be noted that while Theorem 1 guarantees an optimal convergence on the discrete graph CBA*/DCBA* constructs (using both the grid resolution and also the f-values of the paths reaching a particular grid) and Theorem 3 guarantees that the cost information is accurate in the continuous domain, we cannot directly combine these results to guarantee optimality (or for that matter completeness) in the continuous domain. The optimality guarantee is on the underlying search graph obtained from the grid, but it can always happen that CBA*/DCBA* chose a particular configuration for a grid cell and that particular configuration does not lead us to the goal, or may be the goal is not at all reachable following the grid neighbors, in such cases the algorithm may return a failure in spite of the fact that there is a path to goal.

Experimental Results

We evaluated the performance of the proposed algorithms (CBA* and DCBA*) by simulation and on a quadrotor. For comparison, we selected two planning algorithms, i) a grid based A* algorithm (GrA*), where an optimal path is computed on the grid, and later the control algorithm attempts to realize the path (as mentioned in Introduction) and ii) a sampling based planner following the kinodynamic RRT-Connect algorithm (LaValle and Kuffner 2001) with suitable extensions for handling the environmental disturbances.

	GrA*	RRTConnect	CBA*	DCBA*
SR	22	76	90	91
SC	3487	4523	3446	3434
RT	1.06	2.91	8.04	10.20

Table 1: Comparison of GrA*, RRTConnect, CBA* and DCBA* for 500×500 grids with 5% obstacle density and $v_w = 0.5 \times v_a$. Legend: SR - Success Rate, SC - Average Solution Cost, RT - Average Runtime.

Simulation Results

We tested the algorithms in different environments, by varying the obstacle density (o_d), wind/current velocity (v_w) (both static and dynamically changing) and resolution of the grids (g_r). For each such parameter combination we generated a set of 100 random test-cases to create the benchmark suite. For the A* based algorithms, namely, grid based A* (GrA*), CBA* and DCBA*, we used the Euclidean distance as a consistent heuristic function.

In Table 1 we compare GrA*, CBA*, DCBA* and RRTConnect for 500×500 grids with 5% of its cells blocked (with regular shaped obstacles), the grid resolution is $5 \times 5(m^2)$ per cell, the velocity for the vehicle is $v_a = 5m/sec$ and the wind/current velocity is $0.5 \times v_a$. The time resolution (Δt) for the control algorithm is 0.1 sec. The start and goal positions are chosen randomly along the border of the grid. Each algorithm was allowed a maximum runtime of 30 seconds. The results clearly point to the advantages of using CBA*/DCBA* over the competitive algorithms both in terms of success rate and solution quality. The results show that success rate wise CBA*/DCBA* are considerably better compared to GrA* and slightly better than RRTConnect, and solution quality wise they are much better than RRTConnect. On the flip side, the runtime requirement for CBA*/DCBA* is more than both GrA* and RRTConnect. However, as shown, the average runtime is ≤ 10 secs, which should be acceptable for most real life scenarios. Next, we compare these algorithms' performance by varying the environment parameters, namely, obstacle density, wind/current velocity, grid resolution.

In Figure 3, we include the results with different obstacle densities (from 1% to 20%). The results show some interesting trends, Figure 3a highlights that the success rates of both GrA* and RRTConnect drop substantially with increasing obstacle densities. In comparison, both CBA*/DCBA* maintain a high success level even for cluttered environments. The solution cost results in Figure 3b strengthen our earlier observation that the search based algorithms usually produce better quality solutions. Considering Figure 3c, we observe that the runtime for RRTConnect increases substantially with higher obstacle densities, making CBA*/DCBA* better choices for cluttered environments.

Figure 4 depicts the results with different wind/current velocities, ranging from $0.1 - 0.9 \times v_a$ (for 500×500 grids, $g_r = 5m \times 5m$ and $o_d = 5\%$). The success rate results (Figure 4a) shows that both GrA* and RRTConnect suffer greatly with increasing wind/current velocity, whereas CBA*/DCBA* remains consistent. The impact of strong

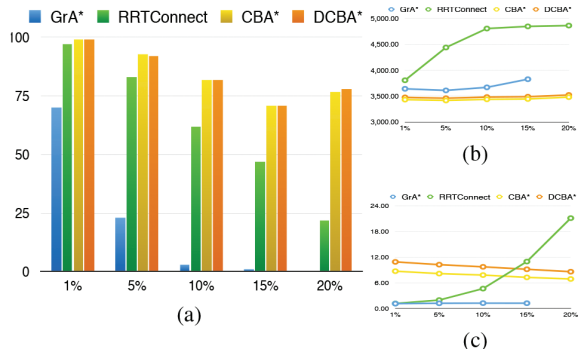


Figure 3: Experiments on 500×500 grids with varying obstacle densities ($1 - 20\%$), $g_r = 5m \times 5m$, $v_a = 5m/sec$, $v_w = 0.5 \times v_a$. (a) compares the success rates of GrA*, RRTConnect, CBA* and DCBA*, b) compares the average solution costs and c) compares the average runtimes.

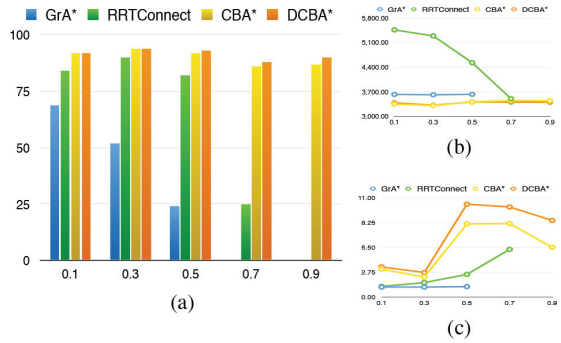


Figure 4: Experiments on 500×500 grids with varying wind/current velocities ($0.1 - 0.9 \times v_a$), $o_d = 5\%$, $g_r = 5m \times 5m$, $v_a = 5m/sec$. (a) compares the success rates, b) compares the average solution costs and c) compares the average runtimes.

wind/current on RRTConnect is more pronounced than the increase in obstacle rates, we see that a strong wind/current causes a close-to-exponential degradation (after a limit). Another interesting trend can be observed in Figure 4c, which shows the runtime for CBA*/DCBA* increases (moderately) with wind/current velocities, this is mainly due the control algorithm which takes more runtime in presence of strong disturbances.

In Figure 5, we present the results for different grid resolutions ($g_r = 1m \times 1m - 10m \times 10m$) for 500×500 maps with $o_d = 5\%$, $v_a = 5m/sec$ and $v_w = 0.5 \times v_a$. These results are included mainly to highlight the difference between CBA* and DCBA*. From Figure 5a we observe that in case of coarse grids, both the algorithms work fine, but CBA*'s success rate reduces dramatically for finer grids. This is primarily due to the fact the CBA* only considers the direct neighborhood of a given cell as potential children, when we have finer resolution grids, reaching such neighbors becomes exceedingly difficult due to both wind/current velocity and turning radius constraints. In contrast, DCBA*

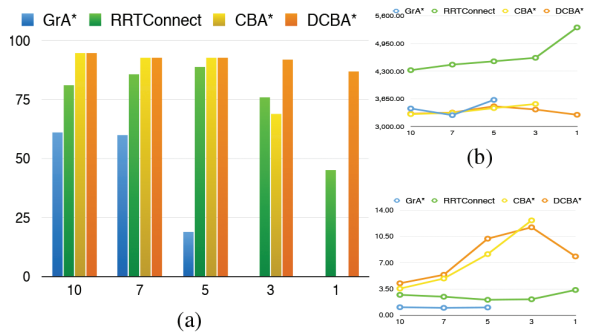


Figure 5: Experiments on 500×500 grids with varying grid resolutions ($1m \times 1m - 10m \times 10m$), $o_d = 5\%$. $v_t = 5m/sec$, $v_w = 0.5 \times v_a$.

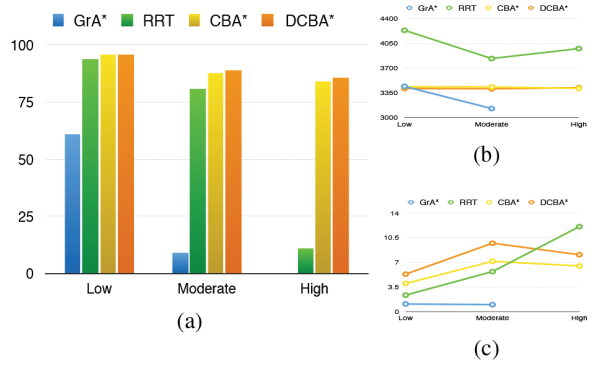


Figure 6: Experiments on 500×500 grids with $g_r = 5 \times 5m^2$, $o_d = 5\%$. $v_t = 5m/sec$, with randomly varying v_w (for low $0.25 \times v_a \leq v_w \leq 0.5 \times v_a$, for moderate $0.5 \times v_a \leq v_w \leq 0.75 \times v_a$, and for high $0.75 \times v_a \leq v_w \leq 1.0 \times v_a$).

performs quite well, as it does not limit the successors to grid neighbors only. It may be noted that the results here do not highlight the advantages of fine resolution planning, instead it shows that for a finer resolution DCBA* is a better choice. In general, it is good to use finer resolution for planning in cluttered environments as coarse grids may result in inflated obstacles (an entire cell is blocked even if the obstacle only intersects with a small fraction), making planning difficult.

Finally, we compare the performance of all the algorithms for dynamic environments where the magnitude and direction of the environmental disturbances (wind/current) vary randomly. To simulate this, we express v_w and ψ_w as a (pseudo) random function of time and space. We include the time information in the robot's configuration state so that the online control function can access the correct values for wind/current velocities using the robot's position and time. Note that, as v_w and ψ_w depend on time, we cannot use the bidirectional RRTConnect algorithm in this case. Instead, we use a uni-directional sampling based approach for comparison. We include the results of this simulation in Figure 6, the experiments were run for 500×500 maps with $o_d = 5\%$, $v_a = 5m/sec$ and $g_r = 5m \times 5m$). We used three kind of environments simulating low, moderate and high disturbance cases. For low, we randomly varied v_w between 0.25 to 0.50 of v_a, for moderate we varied v_w between 0.50 and 0.75 of v_a and for high, between 0.75 and 1.00 of v_a. In each case ψ_w was randomly chosen between 0 and 2π. The results show that both CBA* and DCBA* can adapt well for such dynamic conditions, in contrast the performance of the other algorithms (GrA* and RRT) degrades when compared to static conditions. For RRTs the degradation is mainly due to the fact that we cannot use the bi-directional search, whereas for GrA* more variability leads to more failures.

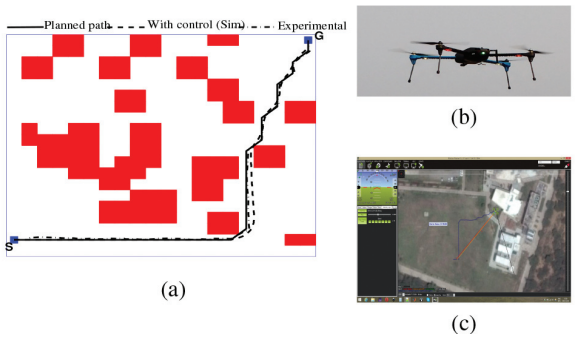


Figure 7: Outdoor experiment with a quadrotor robot. (a) Shows the path flown by the quadrotor along with the planned paths, (b) shows an image of the quadrotor used and (c) shows the path flown by the quadrotor in the mission planner software.

Overall, the results with changing obstacle rates and wind/current velocities depict the efficacy of CBA*/DCBA* in difficult scenarios (cluttered environments/harsh conditions), where integrated planning and control produces significantly better solutions compared to other algorithms.

Quadrotor results

In order to validate the approach in a realistic scenario, we performed an outdoor experiment using Arducopter quadrotor (Figure 7b). The plan generated by the CBA*/DCBA* was given as a set of way-points to the quadrotor. The quadrotor speed was set to $2.5m/s$ and the area of operation was $100m \times 100m$. The path flown by the quadrotor is shown in Figure 7a. The figure also shows the planned path and path taken by the robot during simulation. In figure 7c, we can see the quadrotor path taken from the mission log using Mission planner software. The results show that the vehicle was able to fly the path generated by CBA*/DCBA* algorithm nicely.

Conclusions

We presented two algorithms that integrates search based planning with continuous control, and thus, are able to efficiently compute realizable paths in presence of obstacles, environmental disturbances and kinematic constraints. The experimental results show that the proposed methods outperform the competitive algorithms by a fair margin, especially in cluttered environments/harsh conditions. It may be noted that although we have used grid based search throughout this

work, the algorithms are not restricted to grids (the grid cells are only used for selecting the target points). In fact, we can seamlessly apply these algorithms any discretization, control and collision resolution frameworks.

References

Aine, S., and Likhachev, M. 2013. Truncated incremental search: Faster replanning by exploiting suboptimality. In desJardins, M., and Littman, M. L., eds., *AAAI*. AAAI Press.

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2014. Multi-Heuristic A*. In *Proceedings of the Robotics: Science and Systems (RSS)*.

Aine, S.; Chakrabarti, P. P.; and Kumar, R. 2007. AWA* - A Window Constrained Anytime Heuristic Search Algorithm. In Veloso, M. M., ed., *IJCAI*, 2250–2255.

Barraquand, J., and Ferbach, P. 1994. Path planning through variational dynamic programming. In *Proc. of the IEEE International Conference on Robotics and Automation*, 1839–1846 vol.3.

Barraquand, J., and Latombe, J.-C. 1993. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica* 10(2-4):121–155.

Dolgov, D.; Thrun, S.; Montemerlo, M.; and Diebel, J. 2008. Practical search techniques in path planning for autonomous driving. *Ann Arbor* 1001:48105.

Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics* 497–516.

Furukawa, T.; Bourgault, F.; Durrant-Whyte, H.; and Dissanayake, G. 2004. Dynamic allocation and control of coordinated uavs to engage multiple targets in a time-optimal manner. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 3, 2353–2358 Vol.3.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Hornung, A.; Maier, D.; and Bennewitz, M. 2013. Search-based footstep planning. In *Proc. of the ICRA Workshop on Progress and Open Problems in Motion Planning and Navigation for Humanoids*.

Karaman, S., and Frazzoli, E. 2010. Incremental Sampling-based Algorithms for Optimal Motion Planning. In *Robotics: Science and Systems*. Zaragoza, Spain: The MIT Press.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces. *IEEE T. Robotics and Automation* 12(4):566–580.

Kimmel, R.; Amir, A.; and Bruckstein, A. M. 1995. Finding shortest paths on surfaces using level sets propagation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 17(6):635–640.

Koenig, S., and Likhachev, M. 2002. D* Lite. In Dechter, R., and Sutton, R. S., eds., *AAAI/IAAI*, 476–483. AAAI Press / The MIT Press.

Kuffner, J. J., and LaValle, S. M. 2000. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *ICRA*, 995–1001. IEEE.

Lavalle, S. M., and Kuffner, J. J. 2000. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 293–308.

LaValle, S. M., and Kuffner, J. J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5):378–400.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge university press.

Likhachev, M., and Ferguson, D. 2009. Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *I. J. Robotic Res.* 28(8):933–945.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

MacAllister, B.; Butzke, J.; Kushleyev, A.; and Likhachev, M. 2013. Path Planning for Non-Circular Micro Aerial Vehicles in Constrained Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 3933–3940.

Mitchell, I. M., and Sastry, S. 2003. Continuous path planning with multiple constraints. In *Proc. of the IEEE Conference on Decision and Control*, volume 5, 5502–5507.

Narayanan, V.; Aine, S.; and Likhachev, M. 2015. Improved multi-heuristic A* for searching with uncalibrated heuristics. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SOCS)*, 78–86.

Park, S.; Deyst, J.; and How, J. P. 2007. Performance and lyapunov stability of a nonlinear path-following guidance method. *AIAA Journal on Guidance, Control, and Dynamics* 30(6):1718–1728.

Pearl, J. 1984. *Heuristics: intelligent search strategies for computer problem solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Rimon, E., and Koditschek, D. E. 1992. Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on* 8(5):501–518.

Sujit, P.; Saripalli, S.; and Sousa, J. 2014. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. *Control Systems, IEEE* 34(1):42–59.

Yang, G., and Kapila, V. 2002. Optimal path planning for unmanned air vehicles with kinematic and tactical constraints. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 2, 1301–1306 vol.2.

Zucker, M.; Ratliff, N.; Stole, M.; Chestnutt, J.; Bagnell, J. A.; Atkeson, C. G.; and Kuffner, J. 2011. Optimization and learning for rough terrain legged locomotion. *International Journal of Robotics Research* 30(2):175–191.