Blended Local Planning for Generating Safe and Feasible Paths

Ling Xu
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
lingx@cs.cmu.edu

Anthony Stentz
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
tony+@cmu.edu

***Abstract*— Many planning approaches adhere to the two-tiered architecture consisting of a long-range, low fidelity global planner and a short-range high fidelity local planner. While this architecture works well in general, it fails in highly constrained environments where the available paths are limited. These situations amplify mismatches between the global and local plans due to the smaller set of feasible actions. We present an approach that dynamically blends local plans online to match the field of global paths. Our blended local planner generates paths from control commands to ensure the safety of the robot as well as achieve the goal. Blending also results in more complete plans than an equivalent unblended planner when navigating cluttered environments. These properties enable the blended local planner to utilize a smaller control set while achieving more efficient planning time. We demonstrate the advantages of blending in simulation using a kinematic car model navigating through maps containing tunnels, cul-de-sacs, and random obstacles.**

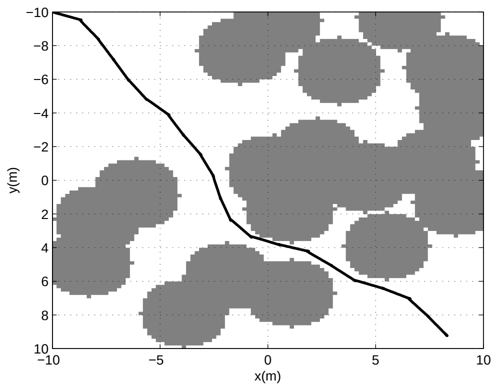


Fig. 1. Example of blended planner navigating a world with randomly-placed obstacles using the model of a kinematic car

I. INTRODUCTION

For many robotic applications, path planning can be divided into two stages: global and local planning. The global planner finds the optimal but coarse-grained route from start to finish, covering a broad area. Because of this large scope, the global planner must use a simplified model of the vehicle's kinematics and dynamics to maintain a small problem space (typically 2D). Local planners cannot afford this simplification because they are responsible for ensuring the safety of the vehicle. Therefore, a local planner focuses on a more limited planning scope to find a detailed path that follows the global path while satisfying the kinematic and dynamic constraints of the vehicle. To achieve both safety and goal acquisition, the two plans are merged.

While this two-tiered technique usually works well, in some cases the plans mismatch at their boundaries, resulting in combined paths that cannot be executed. One example of a mismatch occurs when the final heading segment of the local plan and the initial heading of the global plan segment fail to coincide, creating an disconnected transition at the boundary waypoint. Depending on the dynamics of the vehicle, the lower-level controller may not be able to smooth large heading mismatches particularly when operating on more complex systems traveling at high speeds. More serious examples of boundary mismatches happen when the local planner cannot find a dynamically-feasible path to achieve the global plan.

Additionally, constraining the local planner to follow one global path decreases path diversity, which is key to traversing cluttered environments. A local planner that uses the global path field (the set of all global paths) would allow for greater maneuverability and efficiency in environments containing a myriad of obstacles, cul-de-sacs and tight spaces such as in Figure 1. Our approach bridges global and local planning by creating a blended local planner that utilizes global planning information to generate high fidelity globally-feasible plans. Our blended local planner conforms a set of dynamically feasible paths to the global path field resulting in diverse, globally-informed plans.

Planning high fidelity paths that achieve global waypoints has been studied extensively in the context of motion planning. Methods such as randomized kinodynamic planners [1] and potential-based motion planners [2] sample a space based on vehicle dynamics and random target points. Those methods constrain the plans to end at these target points which may not lie in feasible regions of the planning space. Additionally, several potential field methods exist that utilize global information to reactively deform infeasible paths to be admissable [3], [4]. The potential functions can be highly complex, difficult to adapt to new environments, and subject to local minima.

Several deterministic methods have also addressed this problem. One technique known as ego-graphs [5] creates a static search tree of feasible path segments to achieve global waypoints. In order to densely span poses in the environment, the static search tree must contain a large number of paths which can be memory or space intensive. Another method uses boundary constraint solvers to plan optimal trajectories between global poses creating a lattice-like network [6]. The discretization of the control commands affects the completeness of the approach because it limits action set of the vehicle. Recently Howard, et al, [7] present an approach that samples control actions in the state space. The algorithm uses global information to guide the placement of their samples. However, the approach depends on selecting good end points that are both feasible and strategically placed – a difficult process for cluttered environments.

Our approach differs from these methods by blending a set of candidate control actions with the global path field by dynamically changing the control action online at each step of the path generation process. Our algorithm precludes the selection of sample points at which to connect the local paths to the global paths, since blending enables the set of control actions to naturally find the best points at which to join the two plans. Moreover, this approach avoids the complexities of computing reverse kinematics/dynamics or multiple forward kinematics/dynamics passes since the algorithm only requires a single forward propagation of control actions.

Our approach, the blended local planner, begins with a candidate set of velocity and heading commands. A diverse candidate set of commands is crucial, because some actions may align better with the global path than others. Moreover, the differing actions in the set enable the local planner to negotiate obstacles that were not (yet) included in the global planning process. Next, the planner searches a space by forward propagating the candidate set of commands using the model of the controlled vehicle. The generated path set encapsulates the capabilities of the vehicle but may contain inconsistencies with the global planner. To bring together the global and local plans, we blend the local path with the global path field by changing the set of heading commands in the forward propagation process to align with the space of global headings. Because the system invokes the local planner at constant time intervals, the local path changes as the environment and vehicle position change. The blending process enables the planner to initially spread out the paths to explore a space before conforming to the global path field. Due to the blending property, a blended local planner can cover a space efficiently with a fewer number of control actions than an unblended planner. The blended paths benefit from the path diversification and feasibility of the local planner as well as the completeness and efficiency of the global planner.

Our approach has several major advantages. First, it aligns local paths to the global path field. This decreases the potential of an infeasible transition between the two planners through the blending the control actions to meet the global field. It also increases path diversity due to alignment with the global field rather than a single global path. Second, the method allows the control set to naturally converge to feasible regions in the planning space instead of constraining the control actions to end at a given set of sample points.

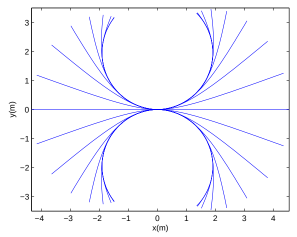


Fig. 2. Paths generated using the controller and model of a kinematic car, v_i = 0m/s, v_f = ±1m/s, 20° separation

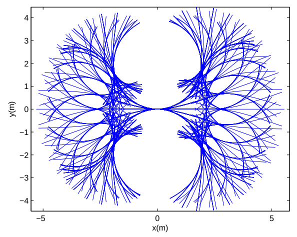


Fig. 3. Paths generated from two sequential control commands using the controller and model of a kinematic car, v_i = 0m/s, v_f = ±1m/s, 20° separation

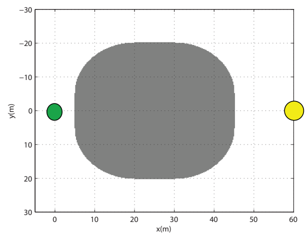


Fig. 4. Example of a C-space environment with one obstacle between the robot (left green circle) and the goal (right yellow circle)

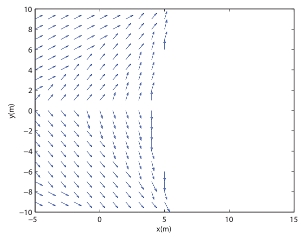


Fig. 5. The global heading field of the environment

Blending possesses the added benefit of limiting the path computation to using only forward kinematics and dynamics. Finally, the blended planner can produce complete paths with a smaller control set than an equivalent unblended planner. This property helps generate smooth paths quickly to support real-time operation. We present simulation results in a tunnel and cul-de-sac scenario as well as in environments with randomly-placed obstacles.

The paper is organized as follows. Section II describes the details of the blended local planner. Section III explains the experiments and results. The algorithm and results are discussed and analyzed in Section IV. Finally Section V concludes and presents future work.

II. Blended Local Planner

Algorithm 1 shows the algorithm for the blended local planner. The following subsections discuss the different components of the algorithm in more detail. First we describe the general underlying technique of forward propagation for path generation (Algorithm 1 lines 4-12). Next we explain the novel method of blending the local heading with the global heading field to create a new local heading (Algorithm 2). Finally we elucidate the terms in the cost function used to find the lowest-cost path (Algorithm 1 line 13).

Before we proceed to the algorithm details, we first define some terms related to the planner as well as outline the structure of the autonomous vehicle system. We define

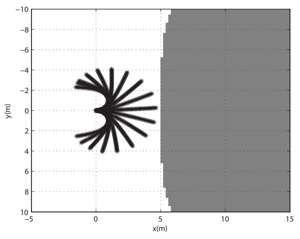


Fig. 6. Unblended set of paths with several directed towards the obstacle

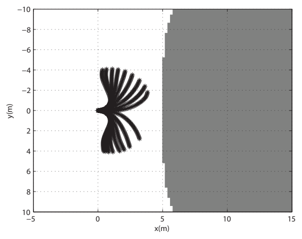


Fig. 7. Blended set of paths that bend around the obstacle like the global field

vehicle state as the composition of positional, rotational, and velocity components. Control commands consist of linear velocity v_i, heading θ_i, and local planning time T_i. A path is a sequence of vehicle states moving from a start to a goal position. The overall control system operates at a constant rate of 10 hertz or cycles every 0.1s (dt). To integrate newly acquired sensor data, the blended local planner is invoked every 15 time steps or cycles every 1.5s. Local planning time T_i equals planning cycle time (1.5s) plus an additional time buffer to ensure safety. At the end of a planning cycle, the algorithm returns its best, valid path within the planning horizon denoted by T_i.

```
Input: $s$,$g$, initial and goal states
       $W$, world model
       $dt$, system time step
       $N$, candidate command set $(v_i, \theta_i, T_i)$ where $v_i$
       = velocity, $\theta_i$ = heading, and
       $T_i$ = local planning time
Output: $P$, path found or empty if no path found
1  foreach $i \in N$ do
2      $PathSet(i) = \{\}$;
3      $c$ = $s$;
4      for $t = 0 : dt : T_n$ do
5          $\theta_b =$
           $ComputeBlendedHeading(W, c, g, \theta_i, t, T_i)$;
6          $c = ForwardPropagation(c, (v_i, \theta_b, t))$;
7          if $Valid(W, c)$ then
8              $Append(c, PathSet(i))$;
9          else
10             break;
11         end
12     end
13     $P = \min_i CostFunction(PathSet(i))$
14     return $P$
15 end
```

Algorithm 1: Blended local planner

```
Input: $c$,$g$, current and goal states
       $W$, world model
       $\theta_i$, candidate heading
       $t$, current time
       $T$, total time
Output: $\theta_b$, blended heading
1  $\theta_g = ComputeGlobalHeading(W, g, c)$;
2  $\beta = \frac{t}{T}$;
3  $\alpha = 1.0 - \beta$;
4  $\theta_b = \alpha\theta_i + \beta\theta_g$;
5  return $\theta_b$
```

Algorithm 2: ComputeBlendedHeading

A. Forward Propagation

The algorithm receives the candidate command set (N) as input. To ensure dynamically-feasible plans, the command set is a sampling of control inputs that spans the space of possible vehicle motions. The forward propagation operation uses a model of the controlled vehicle to simulate the control command applied to the current state to produce the next state (Alg 1, line 6). A control command (v_i, θ_i, T_i) corresponds to a single path, $PathSet(i)$, of the path set. The forward propagation operation ensures that the velocity and acceleration of the vehicle stays within certain limits.

Computing the set of possible paths can be computationally expensive due to complex equations in the vehicle's model and controller. Due to the time constraints inherent to on-line planning, we can precompute the set of paths off-line. In order to accommodate a wide array of initial and ending velocities, the paths span a candidate set of velocity pairs. Figure 2 shows an example of a path set that starts at rest and ends at ± 1 m/s with a 20 degree separation between the segments. Figure 3 illustrates a path set generated using two sequential commands. During runtime, the planner matches the current speed of the vehicle with the precomputed path segments that have the appropriate initial speed, ensuring that each of the potential segments can be safely executed. At the end of the planning process, the planner combines the precomputed segments in real time to produce whole paths. In this paper, we use the kinematic car model to define the vehicle motions, but the algorithm can be easily extended to use more complex models.

B. Blending Process

In order to incorporate information from the global planner, in our case Field D* [8], the algorithm alters the heading control command by modifying the candidate heading with a new global heading at each step of the path (Alg 1, line 5). Algorithm 2 begins by computing the global heading θ_g using the initial state and the world model. A linear weight function then assigns weights α and β to θ_i and θ_g, respectively. The weights are calculated from the current position in the path. The sum of the weighted headings $(\alpha\theta_i + \beta\theta_g)$ produces the new blended heading command θ_b. The new blended command (v_i, θ_b, t) generates a new state in the path. The blended heading equals the local heading at $t = 0$ and then blends to merge with the global heading field at $t = T_i$. This process repeats to create new path states. Additionally, the algorithm prunes paths that collide with obstacles in the environment.

We illustrate the difference between blended paths and unblended paths using a simple example. Figure 4 shows the C-space expansion of an environment where a single obstacle is between the robot and the goal. The gray cells indicate obstacle cells while the white cells indicate free space. Figure 5 displays the global heading field of the world. Note that the headings flow around the obstacle. Without use of this global information, the unblended path set (Figure 6) includes path segments that drive directly towards the obstacle. On the other hand, using the global headings as guidance, the blended paths (Figure 7) move around the obstacle in accordance with the global field.

C. Cost Function

Once a valid set of paths is created, the algorithm assigns a cost to each path (Alg 1, line 13). The minimum cost path is returned as the best plan. The cost used in our approach is a function of several variables. The variables include distance to the goal, distance to obstacles, a momentum term, and an alignment term as described in Equation 1. We use the D* cost to represent the distance to the goal and obstacles. This maintains cost consistency between the global and local planners. D* cost represents the sum of the cell costs along each path, where free cells have low cost, cells near obstacles have higher cost, and obstacle cells have infinite cost. The momentum term indicates how much the local heading changes. To avoid oscillation, we give preference to paths that keep the vehicle heading constant. Finally, the alignment term measures the difference between the local and global headings. Local paths that align more with the global path are lower cost. $\vec{w}$ is the set of weights for the variables, and the sum of the weighted costs equals the total cost of the path.

$$CostFunction = \vec{w} \cdot [dist(g), dist(o), \delta(\theta_l), \|\theta_l - \theta_g\|] \quad (1)$$

III. Experiments and Results

A. Robot Model

We tested the blended planner on a car-like vehicle in simulation. The car is modeled by the dynamic equations below. u_0 represents the vehicle speed, u_1 is the steering angle, and L is the length between the front and back wheels. The steering angle and speed are limited to 0.15π and 1m/s respectively.

$$\dot{\theta} = \frac{u_0 tan(u_1)}{L} \quad (2)$$

$$\dot{x} = u_0 sin(\theta) \quad (3)$$

$$\dot{y} = u_0 cos(\theta) \quad (4)$$

We model the vehicle as a point robot, but the vehicle dimensions can easily be more complex since the C-space expansion accounts for vehicle size. The vehicle has the ability to drive forwards and backwards, thus enabling it to maneuver in tight spaces. The vehicle controller converts the commanded control action to motion inputs using the equations below.

$$u_0 = v_i \quad (5)$$

$$u_1 = (\theta_i - \theta_c) \quad (6)$$

where θ_c is the current vehicle heading.

B. Tests and Results

In order to illustrate the properties of the blended local planner, we conducted three tests to compare the differences between the blended planner and the unblended planner. We show the capability of the blended planner to navigate tight spaces with a smaller path set than for the unblended local planner. The first two tests illustrate these properties by navigating the vehicle in a tunnel environment. The third test places the vehicle at one corner of a randomly-generated world where it must navigate to the opposite corner while avoiding obstacles.

The path set consists of both forward and backward paths. Each path is determined by a single candidate control command (θ_1, v_1, t_1) or a sequence of two candidate commands $[(\theta_1, v_1, t_1), (\theta_2, v_2, t_2)]$. Positive and negative velocities paired with a set of angles define the path set. The angles are generally evenly distributed around a circle with the spacing determined by the size of the angle set. While paths generated from single commands form a single level search tree, paths from coupled commands become a two-level search tree that includes segments with forward motion, backward motion, or both (Figure 3).

The global planner is Field D* with a cell resolution of 0.2m. The local planner finds the global heading by querying Field D* for the path from a particular location to the goal. Field D* runs in the unfocussed mode meaning the path search is undirected allowing for wider path cost propagation. In this manner, a global query becomes an easy lookup in the cost map, so we assume calls to the global planner to be negligible. The planner uses a predetermined lookahead point a distance 1.5 m down the global plath. Then, it calculates the angle between the current point and the lookahead point to be the global heading θ_g.

We measured the performance of the planner in two ways: path length and planning time. The path length indicates the distance the vehicle traveled to reach the goal, and the planning time is the total computational time spent planning until achievement of the goal. Because the planner can return a valid path before the end of a planning cycle, the computational time for one planning cycle could be much shorter than the allocated cycle time. Additionally, the larger the trajectory set, the more time is needed to compute a best, valid path. We ran the tests on a 2.13GHz Intel Pentium M with 1GB of RAM.

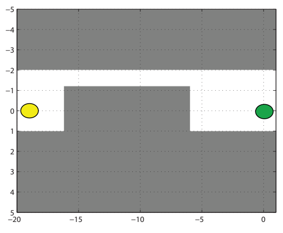


Fig. 8. Tunnel environment: the right green circle denotes the start position and the left yellow circle denotes the goal position. Gray represents obstacles and white free space.

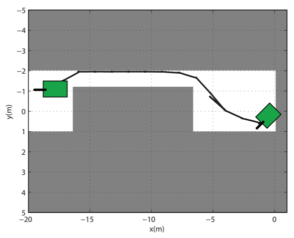


Fig. 9. Path result from using unblended planner with a set of 8 paths with a 90° separation

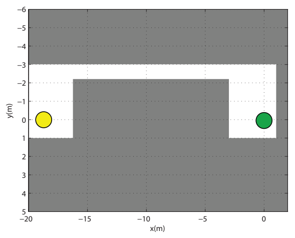


Fig. 12. Cul-de-sac environment: the right green circle denotes the start position and the left yellow circle denotes the goal position. Gray represents obstacles and white free space.

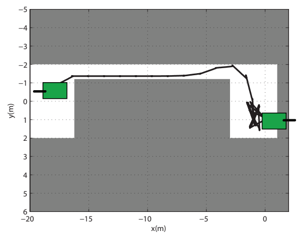


Fig. 13. Path result from using unblended planner with a set of 8 single and 64 coupled paths with a 90° separation

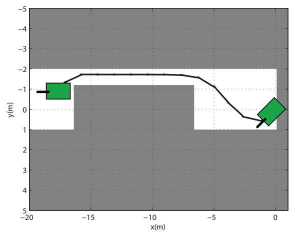


Fig. 10. Path result from using unblended planner with a set of 20 paths with a 36° separation

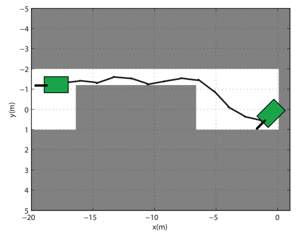


Fig. 11. Path result from using blended planner with a set of 8 paths with a 90° separation

1) *Tunnel:* One of the features of the blended planner is its ability to align with the global path field. This alignment property modifies the local paths making it possible to use a smaller path set during planning. We illustrate this feature by testing the planner in an enviroment with a 1m wide tunnel. As shown in Figure 8, the robot begins on one side of the tunnel at position (0,0) and navigates to position (-18,0) on the other side. At the beginning, the robot faces 45 degrees away from the goal ($135°$ heading). Due to the narrowness of the tunnel, the robot must align itself perfectly before it can proceed into the passage. The results are shown in the table below. C represent the size of the control set.

Tunnel	Plan Time(s)	Path Length(m)
Unblended-1 C=8	2.3	22.36
Unblended-1 C=20	3.38	19.57
Blended-1 C=8	1.768	19.65

During the test, we compared the blended local planner with the unblended planner while varying the size of the path set. With a path set of 8 single control commands, the unblended local planner found the path shown in Figure 9. In order to align the vehicle to the tunnel, the planner issued one backwards motion to ensure the vehicle heading was correct. By increasing the local path set, the unblended local planner succeeded in achieving the goal with no backup movement with 20 commands in the set (Figure 10). In contrast, with 8 commands, the blended planner had no problem aligning with the tunnel (Figure 11). Blending allowed the planner to find a smooth path in less time than the unblended planner using the same number of paths. Additionally, the same path generated using blending took nearly half the time and almost half the number of commands as that of the unblended planner.

2) *Cul-de-sac:* In addition to tunnels, we tested the ability of the planner to navigate tight spaces by finding plans that involved driving backwards. One example is a situation where the vehicle had to drive backwards in order to align itself to the goal. To illustrate this property, we used a similar tunnel environment and started the robot pointed away from the goal (0° heading). The robot could not move forward because it was pointed straight at a wall. Therefore, the planner was required to perform a turning maneuver to align the vehicle heading with the tunnel. Again, we varied the size of the path set to find the ideal number of paths for the vehicle to plan a feasible path to the goal. In addition to the path set of single command paths, we added coupled commands to create a search tree with two levels levels. This enabled the planner to look ahead one more path step. Results are shown in the table below:

Cul-de-sac	Plan Time(s)	Path Length(m)
Unblended-2 C=72	10.62	42.17
Unblended-2 C=274	12.84	25.31
Blended-2 C=72	5.29	23.93

The blended and unblended planners both reached the goal with a path set consisting of 8 single commands and 64 coupled commands. Although the unblended planner reached the goal, the path contained numerous forwards-backwards motions due to its inability to modify the path segments (Figure 13). In contrast, the blended planner reached the goal in half the time with a shorter path length (Figure 15). In order for the unblended planner to generate a comparative plan, it required a set of 18 single commands and 256 coupled commands (Figure 14).

3) *Random obstacles:* Finally we tested the planners in binary environments with randomly placed obstacles. The world was 20m by 20m with 30 single point obstacles at

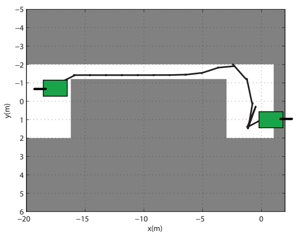


Fig. 14. Path result from using unblended planner with a set of 16 single and 256 coupled paths with a 45° separation

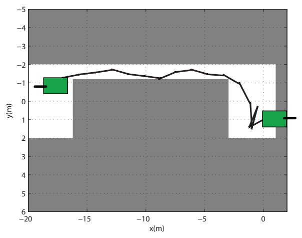


Fig. 15. Path result from using blended planner with a set of 8 single and 64 coupled paths with a 90° separation

random locations. A buffer of 2m in width was placed around each obstacle and obstacles placed near the start or goal were removed from the world model. The vehicle started at a random corner of the environment and navigated through the obstacle field to reach the goal at the opposite corner. We tested the blended and unblended local planners with both single-level and two-level search trees consisting of 8 commands and 72 commands, respectively. Ten random worlds and ten random start and end locations were generated. We ran each planner on each of the ten worlds and recorded the distance traveled and planning time. We then averaged the path length and planning time. The table below shows the results from the experiments.

Averages	Plan Time(s)	Path Length(m)
Unblended-1 C=8	6.7	51.02
Blended-1 C=8	5.14	41.11
Unblended-2 C=72	19.21	46.04
Blended-2 C=72	15.9	38.77

Test 3 shows that both the blended planners (single-level and two-level) performed better both in time and distance than the equivalent planner without blending. With 8 single command paths, the blended planner performed 1.64s better in time and 9.91m better in distance than the unblended planner. With 64 coupled command paths and 8 single command paths, the blended planner performed 3.31s better time-wise and 7.07m better distance-wise. However, these number do not tell the whole story. The unblended one-level planner failed on one of the tests and the unblended two-level planner failed on 3 of the tests. Figures 16 and 17 show the time and length results for each of the ten tests.

IV. Discussion

The tests indicate that the blended planner uses global information to reduce planning time. The benefits of blending increase in tight spaces where alignment to the global field is crucial in navigation. Blending also reduces planning time by limiting the set of controls considered during search. For instance, in the tunnel scenario, blending enabled the planner to bend the paths to conform to the global field through the tunnel with 8 paths. Without blending, the planner required an extra 12 paths to generate a similar path.

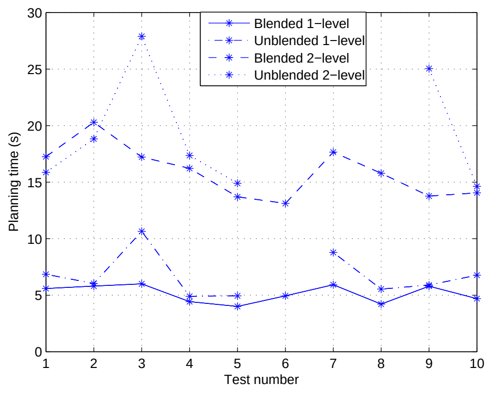


Fig. 16. Planning time for the random tests

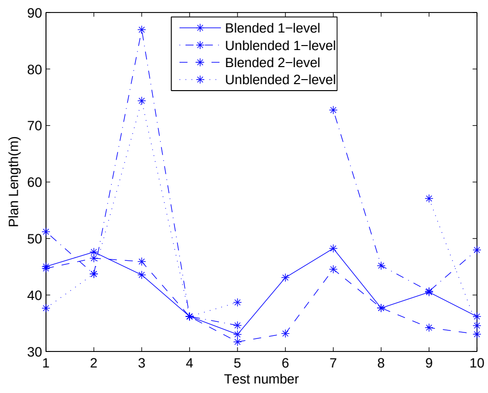
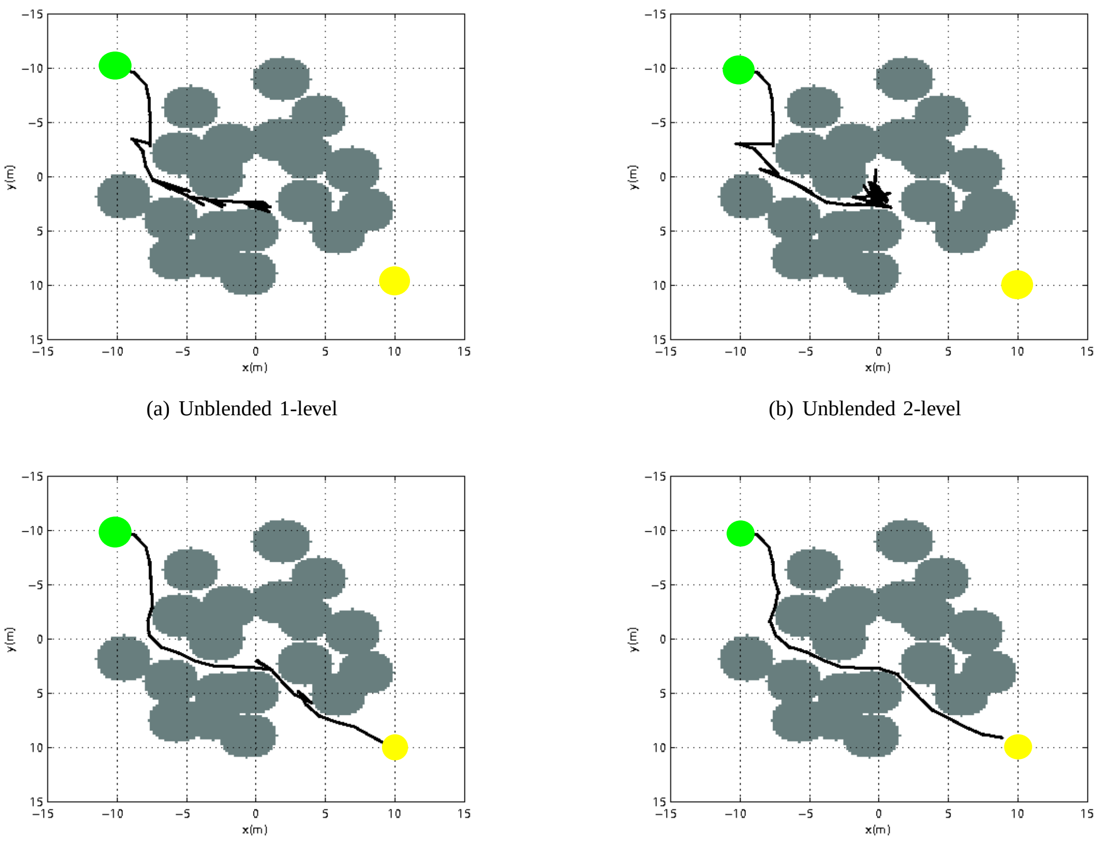


Fig. 17. Path length for the random tests

Additionally, blending aids in maneuvering the robot in spaces that require moving backwards and forwards. The cul-de-sac test forced the planner to produce n-point turns to align the vehicle to the tunnel. Increasing the planning horizon by one level allowed the planner to look further than one control command. In this situation, the unblended planner using a set of 8 single commands and 64 coupled commands drove back and forth multiple times to obtain an appropriate vehicle heading. In contrast, blending enabled the planner to find a 4-point turn that properly positioned the vehicle while a similar path required the unblended planner to investigate three times more paths with a result that is twice as long.

In Figures 11 and 15, the blended paths oscillate more than their unblended counterparts. This phenomenon is an artifact of the static lookahead distance used when determining the global heading. Due to the small distance used, $1.5m$, the calculated angle may not reflect the true angle of the path. Moreover, the path heading can change drastically as the position changes which leads to oscillations. We suggest dynamically changing the lookahead distance to produce smoother paths.

When navigating in randomly-generated environments,



(a) Unblended 1-level

(b) Unblended 2-level

(c) Blended 1-level

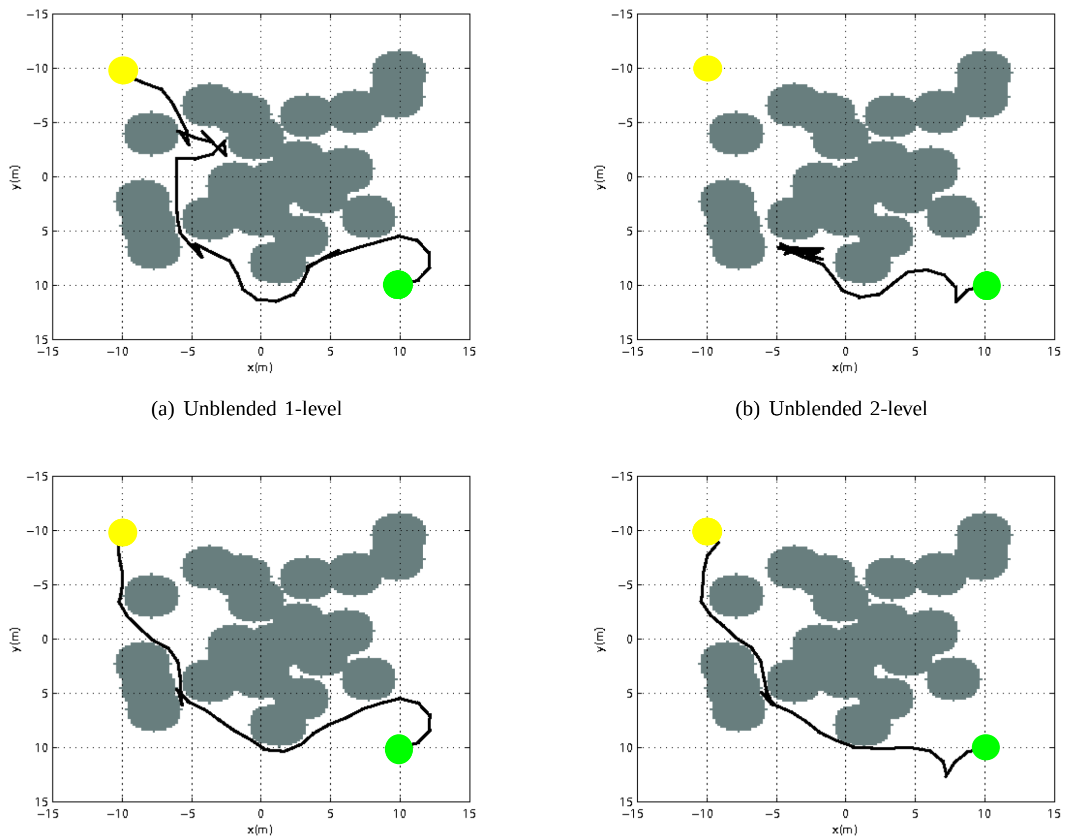
(d) Blended 2-level

Fig. 18. This figure shows the plan results from the 4 different planners on test 6 of the random tests. The start position is the top left green circle and the end position is the bottom right yellow circle. The unblended planners fail to navigate the vehicle through the environment. The one-level blended planner achieves the goal, but the resulting path involves many 3-point turns while the two-level blended planner navigates the environment smoothly and efficiently.

the blended planners performed better than the equivalent unblended planners both in planning time and path length. Figure 18 shows test 6 which contains tight spaces the robot must navigate through. The unblended planners spend time aligning the vehicle to certain angles by moving forwards and backwards multiple times but failed to find a plan due to their static paths. The one-level blended planner found a path to the goal, but causes the vehicle to back up many times. On the other hand, the two-level blended planner generates smooth paths due to its two step lookahead. This example illustrates that blending generates more complete plans in highly constrained scenarios. Figure 19 depicts another environment containing tight tunnels. The blended planners navigated this environment with a few backing up maneuvers. The single-level unblended planner succeeded in reaching the goal, but traveled 86.94m while the two-level unblended planner failed to navigate through the narrow openings. The two-level unblended planner performed worse because it favored paths that realigned the vehicle heading using a sequence of backward and forward motions rather than persisting in the forward direction. Overall, the two-level blended planner resulted in the lowest path length while the one-level blended planner is the most efficient computationally.

V. Conclusion

We present blending as a technique for using global path information to plan local paths that preserve vehicle constraints but adhere to global feasibility. Blending enables the local planner to plan paths with a smaller command set than an equivalent unblended planner. Rather than planning to achieve one optimal global path, blending creates more diverse local paths by merging the candidate control set with the set of global paths defined by the global path field. Additionally, blending enables the action commands to find the feasible regions in the world without constraining the paths generated to meet at a given set of sample points. Finally, blending finds feasbile paths in situations where an equivalent unblended planner may not. Our test results show that blending performs more efficiently both in time and distance than without it (all other factors equal) while maneuvering in tight spaces and in randomly generated environments.



(a) Unblended 1-level (b) Unblended 2-level

(c) Blended 1-level (d) Blended 2-level

Fig. 19. This figure shows the plan results from the 4 different planners on test 7 of the random tests. The start position is the bottom right green circle and the end position is the top left yellow circle. The blended planners navigates the vehicle to the goal efficiently while the unblended single-command planner requires twice the distance. The unblended two-level planner fails to reach the goal at all.

For future work, we would like to test this method with different, more complex vehicle models, such as the autonomous helicopter. We will also conduct comparisons of the planner to other existing local planners. Additionally, field testing with the blended planner would give practical results of the approach.

VI. Acknowledgments

This work was partially sponsored by the U.S. Army Research Laboratory, under contract Robotics Collaborative Technology Alliance (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. Ling Xu is partially supported by a U.S. National Science Foundation Graduate Research Fellowship. We also thank Omead Amidi, Nicholas Vallidas, and James Kuffner for their development of the initial code base.

References

[1] S. LaValle and J. Ku. Randomized kinodynamic planning. In *Proceedings of the IEEE Int'l Conf. on Robotics and Automation*, 1999.

[2] E. Feron, E. Frazzoli, and M. Dahleh. Real-time motion planning for agile autonomous vehicles. In *Proceedings of the AIAA Conf. on Guidance, Navigation and Control*, August 2000.

[3] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

[4] O. Brock and O. Khatib. Real time replanning in high-dimensional configuration spaces using sets of homotopic paths. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2000.

[5] A. Lacaze, Y. Moscovitz, N. DeClaris, and K. Murphy. Path planning for autonomous vehicles driving over rough terrain. In *Proceedings of the IEE ISIC/CIRA/ISAS Joint Conf.*, 1998.

[6] Mihail Pivtoraiko, Ross Alan Knepper, and Alonzo Kelly. Optimal, smooth, nonholonomic mobile robot motion planning in state lattices. Technical Report CMU-RI-TR-07-15, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2007.

[7] Thomas Howard, Colin Green, and Alonzo Kelly. State space sampling of feasible motions for high performance mobile robot navigation in highly constrained environments. In *Proceedings of the 6th International Conferences on Field and Service Robotics*, July 2007.

[8] David Ferguson and Anthony (Tony) Stentz. Field D*: An interpolation-based path planner and replanner. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, October 2005.