An Efficient Heuristic Estimate for Non-holonomic Motion Planning

Ji-Wung Choi

***Abstract*— A new efficient and admissible heuristic estimate function is proposed for non-holonomic motion planning. The heuristic calculation begins by partitioning a configuration space into visible and invisible spaces from the perspective of the goal configuration. The heuristic values of visible configurations are assigned by pre-computed heuristics through full state space assuming empty environment. The heuristics are extended, through the reduced Euclidean $2D$ space, into the invisible configurations by using dynamic programming. The numerical simulations demonstrate remarkable performance improvement in motion planning quries by applying the heuristic function, compared to other existing heuristics [1] and [2].**

I. INTRODUCTION

Motion planning is crucial to achieve autonomy of mobile robots. Heuristic search algorithm, such as A^*, is one of the most widely used algorithms to find the solution of motion planning problems for its performance and accuracy. The algorithm benefits from heuristic knowledge that guides search into promising directions to the goal configuration. Clearly, more accurate heuristic to estimate the true path costs leads to better performance. Conventional heuristic functions such as Euclidean distance may not be adequate for non-holonomic path search, because it underestimates path costs by violating kinematic constraints of robots. These underestimated heuristics can mislead search and thereby aggravate the bottleneck situation as the size of the search space grows. If we assume completely empty environment, feasible path costs can be pre-computed offline and stored in a Heuristic Look-Up Table (HLUT) [3]. However, the HLUT cannot consider obstacles since it is implausible to encode all possible worlds. So, a computationally efficient heuristic in large and complex environment has remained challenging.

One of the most effective heuristic functions in the literature may be the one applied for autonomous ground vehicles of CMU [1] and Stanford [2] teams in the DARPA Urban Challenge (DUC). The heuristic is given by the maximum of two component heuristics: 1) non-holonomic-without-obstacle (stored in HLUT) and 2) holonomic-with-obstacle. While the heuristic is well informed in local area around the goal configuration, it often underestimates the actual path costs in other area (as more detailed in Section II-B).

This motivated us to propose a novel, hybrid heuristic estimate function. The first step of computing the heuristic is to divide the state space into two: 1) visible and 2) invisible spaces from the goal state's perspective. Heuristic values of the visible space are simply copied from HLUT. For the invisible space, though, complexity is inherited from existence of obstacles. We cope with this complexity by projecting the visible heuristic values into the reduced Euclidean $2D$ space and applying dynamic programming to extend the values into the invisible space. As the result, the heuristic becomes admissible and consistent. The simulation results provided in Section IV demonstrate benefits from the new heuristic function in motion planning query.

This work was supported by the Academy of Finland under GIM project.

J.-W. Choi is with Department of Intelligent Hydraulics and Automation, Tampere University of Technology, 33101 Tampere, Finland ji.choi@tut.fi

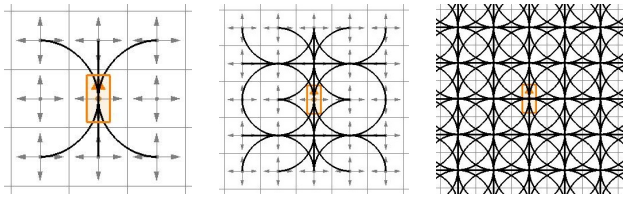
II. PRIOR WORK

A. State Lattice

Many work on search algorithms provide computationally efficient way in discrete state spaces [4], [5]. However, the resulting paths by such algorithms do not tend to be smooth (piecewise linear) and hence, do not satisfy kinematic feasibility of the robot. In order to satisfy motion constraints while achieving the computational advantages of discretization, Pivtoraiko and Kelly proposes the *state lattice* [6]. The state lattice is a graph representation of a discretized configuration space, where repeating pairs of nodes (configurations or states) are connected by a finite set of feasible path segments, referred to the *control set*. Each control must be subject to robotic systems' dynamics such as non-holonomic nature and the maximum curvature constraint so that the constrained motion planning is formulated into query as graph search.

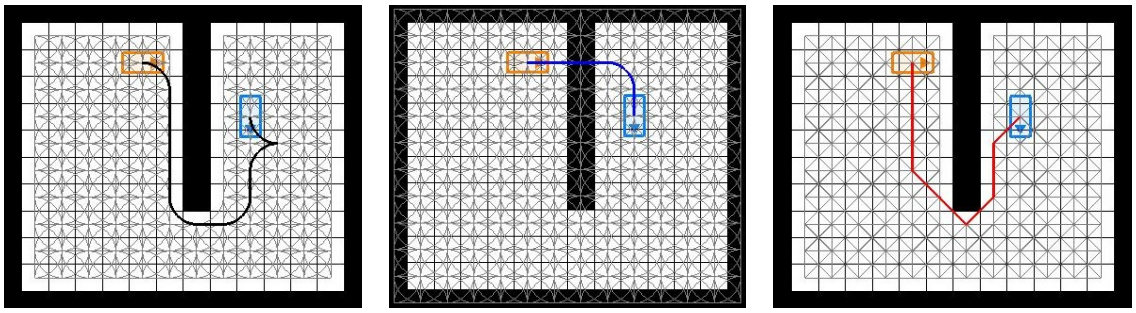
Fig. 1 illustrates an example of $3D$ state lattice (x, y, θ). A search graph is constructed by copying the control set at states in a configuration space, $\mathcal{C}_{space}$ (Fig. 1(c)). The advantage of the lattice planner comes from kinematic feasibility requirement of lattice connections. As such, a sequence of controls in the graph forms a path that guarantees feasibility.

Due to the compactness, we describe our new heuristic function in the state lattice frame depicted in Fig. 1.



(a) (b) (c)

Fig. 1. A $3D$ state lattice construction. (a) The lattice is composed of discretized position (x, y) and headings θ, illustrated as dots and arrows, respectively. The control set (thick solid curves) is a set of elementary feasible motions to connect each state and its reachable neighbors. (b) The reachability tree obtained by proceeding 2 steps with the control set. (c) The complete reachability tree is constructed by copying the control set at states in a $\mathcal{C}_{space}$.



(a) Nonholonomic-with-obstacle for h^+_{3D} (b) Nonholonomic-without-obstacle for h^-_{3D} (c) Holonomic-with-obstacle for h^+_{2D}

Fig. 2. Search graphs representation. (a) The nonholonomic-with-obstacle graph is constructed by copying the control set elements that do not hit $\mathcal{C}_{obstacle}$ to the $3D$ (full dimension) states in free space. Since each control is generated subject to kinematic constraints, a sequence of controls in the graph forms a safe path that guarantees feasibility. (b) The nonholonomic-without-obstacle graph is constructed by copying the control set at every state in a $\mathcal{C}_{space}$, without considering obstacles. While paths in the graph guarantee feasibility, they can collide with obstacles. (c) The holonomic-with-obstacl graph consists of 8-(or 4-)connected $2D$ states (positions) in free space. Paths in the graph guarantee obstacle avoidance but violate kinematic constraints; Given the initial and goal state (red and blue rectangle), costs of the optimal solutions (thick solid curves) through the graphs defined in (a), (b), and (c) are denoted h^+_{3D}, h^-_{3D}, and h^+_{2D}, respectively.

B. Maximum Heuristic

The heuristic search algorithms have been widely used to achieve autonomy of mobile robots operating in dynamically changing environment. A heuristic is intended to improve search performance by guiding the search in promising directions. The performance significantly depends on the accuracy of heuristic estimation for actual path cost. The accuracy comes at a price of computational complexity. Although, for example, naive $2D$ Euclidean distance heuristic is computationally cheap, it often underestimates path costs by violating non-holonomic nature of the robot. Conversely, the optimal solution in the $3D$ (full dimension) state lattice embedded in a free space, as in Fig. 2(a), may perfectly estimate the costs under the limit of the lattice resolution. (Let us denote the cost function h^+_{3D}.) However, it is computationally impractical as the search space grows.

So, it still has remained challenging to provide a heuristic function that balances estimate accuracy and computational efficiency. This subsection introduces one of the most effective approaches in the literature. The heuristic has been applied for unmanned ground vehicles of the CMU [1] and Stanford [2] teams in the DARPA Urban Challenge. We call the heuristic the *maximum heuristic* because it is given by the maximum of two component heuristics: 1) non-holonomic-without-obstacle and 2) holonomic-with-obstacle.

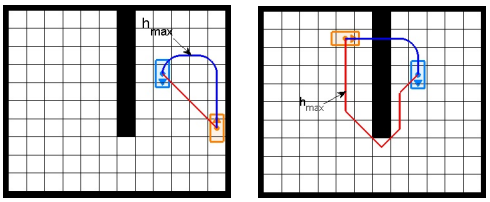
The non-holonomic-without-obstacle heuristic, denoted h^-_{3D}, is the cost of an optimal solution in the $3D$ (full dimension) state lattice copied at a $\mathcal{C}_{space}$ without considering obstacles. Fig. 2(b) shows the search graph in which h^-_{3D} is defined. The heuristic can be pre-computed and stored as a Heuristic Look-Up Table (HLUT). Then, it is translated and rotated with respect to the goal state. While the heuristic is well informed in sparse spaces, it gradually underestimates the true costs in dense spaces due to the lack of obstacle information. (Compare the solution in Fig. 2(b) with that in Fig. 2(a).) The reason why obstacles can not be taken into account on HLUT is because there are nearly infinite number of possibilities for obstacle-laden environments.

The holonomic-with-obstacle heuristic, denoted h^+_{2D}, copes with the complexity inherited from existence of obstacles. The cost is obtained by running Dijkstra's algorithm on the reduced $2D$ Euclidean state space with considering obstacles. Fig. 2(c) shows the search graph in which h^+_{2D} is defined. While the heuristic can be calculated online by virtue of state dimension reduction, it may underestimate the true path costs by ignoring non-holonomic nature. (Compare the solution in Fig. 2(c) with that in Fig. 2(a).)

The maximum heuristic, denoted h_{max}, takes the maximum of the two heuristics:

$$h_{max}(x,y,\theta) = \max(h^-_{3D}(x,y,\theta), h^+_{2D}(x,y)),$$

and thereby gains an order of magnitude performance improvement than either of the two component heuristics [1].

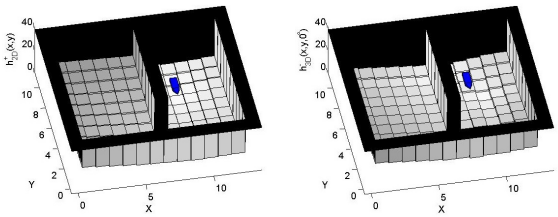


(a) $h^-_{3D} > h^+_{2D} \Rightarrow h_{max} = h^-_{3D}$ (b) $h^+_{2D} > h^-_{3D} \Rightarrow h_{max} = h^+_{2D}$

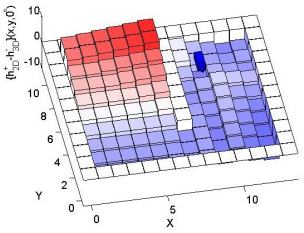
Fig. 3. The maximum heuristic value of a state (x,y,θ) (red rectangle) is determined by $h^-_{3D}(x,y,\theta)$ or $h^+_{2D}(x,y)$, depending on the location of (x,y) relative to the goal (blue rectangle). Blue and red curves represent solutions of h^-_{3D} and h^+_{2D}, respectively. (a) If (x,y) lies in open local area around the goal, h_{max} is likely to be determined by h^-_{3D}. (b) If an obstacle lies between (x,y) and the goal, h_{max} is likely to be determined by h^+_{2D}.

Fig. 3 shows that $h_{max}(x,y,\theta)$ is determined by $h^-_{3D}(x,y,\theta)$ or $h^+_{2D}(x,y)$ depending on the relative location of (x,y) to the goal state. Referring to Fig. 3(a), if a state (red rectangle) lies in open local area around the goal state (blue rectangle), then h_{max} is likely to be determined by h^-_{3D} (blue curve). Otherwise, as shown in Fig. 3(b), h_{max} is likely to be determined by h^+_{2D} (red linear path).

This position dependency of h_{max} is clearly visualized in Fig. 4(c). The figure shows the differences between the two



(a) $h^{+}_{2D}(x,y)$ (b) $h^{-}_{3D}(x,y,0^\circ)$



(c) $h^{+}_{2D}(x,y) - h^{-}_{3D}(x,y,0^\circ)$

Fig. 4. Given the environment in Fig. 3, (a) $h^{+}_{2D}(x,y)$ and (b) $h^{-}_{3D}(x,y)$ with $\theta = 0^\circ$ are displayed. (c) The differences $h^{+}_{2D}(x,y) - h^{-}_{3D}(x,y,0^\circ)$ represent position dependency of $h_{max}(x,y,0^\circ)$: h_{max} is determined by h^{+}_{2D} (or h^{-}_{3D}) at the positive (or negative) valued positions filled with red (or blue).

heuristics, $h^{+}_{2D}(x,y) - h^{-}_{3D}(x,y,0^\circ)$, for the $\mathcal{C}_{space}$ and the goal state shown in Fig. 3. While negative values (blue), in which h_{max} is determined by h^{-}_{3D}, are located in the relatively open space close to the goal, positive values (red), in which h_{max} is determined by h^{+}_{2D}, are located in the space concealed behind $\mathcal{C}_{obstacle}$ from the perspective of the goal.

The heuristic determined by h^{-}_{3D} in the blue area relatively well estimates true optimal path costs. Although, on the other hand, h^{+}_{2D} estimates better than h^{-}_{3D} in the red area, it still underestimates true path costs. For example, let us compare the red linear path in Fig. 3(b) with the optimal solution in Fig. 2(a). While the latter involves the robot making an U-turn to reach the goal state, the first simply consists of line segments, and hence informs shorter distance (smaller cost) than the latter. The underestimated heuristic can mislead the search into the exploration of wrong regions. This motivated us to propose a novel, hybrid heuristic estimate function.

III. The Proposed Hybrid Heuristic

This section proposes a new heuristic estimate function to overcome the drawback of the maximum heuristic, described above. The proposed heuristic has been created based on the following observations:

- $2D$ configuration space is relatively well divided into two separate regions, depending on the comparison between h^{-}_{3D} and h^{+}_{2D} (as in Fig. 4(c)).
- In the region closer to the goal (so that the goal is visible), h^{-}_{3D} well estimates optimal costs.
- In the other region (where the goal is invisible), h^{+}_{2D} estimates better than h^{-}_{3D}, but underestimates true optimal costs.

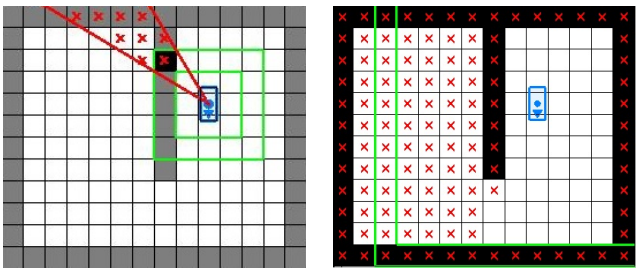
From the observations above, we introduce the concept of the *visibility* (from the perspective of the goal position) to divide $2D$ configuration space into two: visible and invisible spaces (detailed in Section III-A). Since h^{-}_{3D} performs well in the visible space, its heuristic is simply copied from HLUT into the region. Then, $2D$ version costs are extended from the visible to invisible space by using dynamic programming. As the result, the heuristic of the invisible space is a combination of non-holonomic and holonomic path costs and hence estimates better than pure h^{+}_{2D} (detailed in Section III-B).

A. Visible Space

The visible space $\mathcal{V} \subset \mathbb{R}^2$ is the set of the positions from which the goal (x_f, y_f) is visible. More specifically, if the line connecting (x,y) and (x_f, y_f) intersects no obstacle, then (x,y) is defined to be visible, otherwise invisible.

Algorithm 1 Visible Space Construction

```
1:  procedure BUILDVSPACE(x_f, y_f, C_space)
2:      Mark all (x,y) ∈ C_space as visible.
3:      [x_min, x_max, y_min, y_max] ⇐ size(C_space)
4:      L_max ⇐ max(x_max − x_f, x_f − x_min, y_max − y_f, y_f − y_min)
5:      for i = 1 → L_max do
6:          for all (x,y) s.t. L_∞([x,y] − [x_f, y_f]) = i do
7:              if (x,y) ∈ C_obstacle then
8:                  [x', y'] ⇐ InvisibleStates(x, y, x_f, y_f)
9:                  Mark all (x', y') invisible.
10:             end if
11:         end for
12:         if all (x,y) are marked as invisible then
13:             break
14:         end if
15:     end for
16:     return V ⇐ {(x,y) | (x,y) is marked as visible}
17: end procedure
```



(a) Invisible states by a cell (b) Overall invisible states

Fig. 5. (a) Given the goal state (blue rectangle) and a state in $\mathcal{C}_{obstacle}$ (black cell), invisible states (red crosses) concealed behind the cell lies inside of two tangential rays (red lines) from the goal to the cell. (b) The invisible space (filled with red crosses) is the set of all invisible states by $\mathcal{C}_{obstacle}$. The visible space (white cells) is the difference set of the invisible space.

Algorithm 1 is the pseudo code for constructing the visible space $\mathcal{V}$. Initially, all $2D$ states in a $\mathcal{C}_{space}$ are marked as *visible* (Line 2). Then, it iterates exploring (x,y) on the square boundary around (x_f, y_f) as its maximum norm[1] L_∞ grows from 1 until it reaches the boundary of $\mathcal{C}_{space}$ (Line 6 -11). If (x,y) is in $\mathcal{C}_{obstacle}$ (Line 7), then all the states

[1] $L_\infty([x,y]) = \max(|x|, |y|)$

concealed behind (x,y) from the perspective of (x_f,y_f) are calculated (Line 8) and marked as invisible (Line 9). Fig. 5(a) visualizes the invisible states (red crosses) by an obscured cell (black). The invisible states lie inside of two tangential rays from (x_f,y_f) to the cell. They can be pre-computed and stored in a look-up table to speed up the construction. The procedure terminates as all the (x,y) on the square boundary (green lines) is invisible as shown in Fig. 5(b) (Line 12).

B. The Hybrid Heuristic Construction

The hybrid heuristic is calculated based on the visible space, as presented in Algorithm 2. In the pseudo-code, Θ is the canonical set of discrete headings, and ρ is the threshold distance to compensate underestimate effect by holonomic-with-obstacle path costs (more detailed in next subsection). The algorithm is mainly divided into two loops: one cycles through the visible space (Line 5-16) and the other through the invisible space (Line 17-20). The first loop copies $h^-_{3D}(x,y,\theta)$ to all visible states (Line 7). As stated in Section II-B, the heuristic values are pre-computed offline and stored in HLUT so that the copy performs in constant time complexity. The copied heuristics are extended to heuristics of invisible states. Since invisible heuristics are defined in $2D$, we project the visible heuristic values $h^-_{3D}(x,y,\theta)$ onto $2D$ version, $H_{2D}(x,y)$ with the minimum for all $\theta \in \Theta$ (Line 9). If a visible $2D$ state (x,y) has an invisible successor (x',y'), then the heuristic of the successor is updated with the minimum of sum of $H_{2D}(x,y)$, $\|(x,y)-(x',y')\|$, and ρ (Line 12-13). Once all visible states has been copied, H_{2D} will have contained heuristics for all the visible states and the invisible states adjacent to the visible space. Finally, we run dynamic programming in $2D$ space to solve the shortest distance from each invisible state to the goal with incorporating the existing H_{2D} (Line 18).

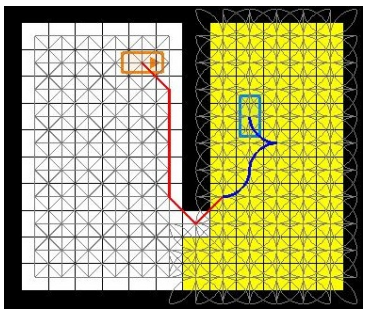


Fig. 6. The hybrid heuristic graph is constructed by copying h^-_{3D} graph at visible space (yellow cells) and h^+_{2D} graph at invisible space (white cells). Thus, the optimal solutions through the hybrid graph combines h^-_{3D} (blue) and h^+_{2D} paths (red).

Fig. 6 shows the hybrid heuristic graph and a solution through the graph. In the visible space filled with yellow, h_{hyb} is determined by non-holonmic-without-obstacle path cost as h_{max} is. The difference between h_{hyb} and h_{max} is that h_{hyb} combines it with holonomic-with-obstacle path cost in the invisible space. In other words, the reference path for h_{hyb} is the combination of a holonomic path (the red linear path)

Algorithm 2 Hybrid Heuristic Estimate

```
1:  procedure HybridHeuristic(V, C_space, Θ, ρ)
2:      for all (x,y) ∈ C_space do
3:          H_2D(x,y) ⇐ ∞
4:      end for
5:      for all (x,y) ∈ V do
6:          for all θ ∈ Θ do
7:              h_hyb(x,y,θ) ⇐ h^-_3D(x,y,θ)
8:          end for
9:          H_2D(x,y) ⇐ min_{θ∈Θ} h^-_3D(x,y,θ)
10:         for all (x',y') ∈ Succ(x,y) do
11:             if (x',y') ∉ V then
12:                 H_t ⇐ H_2D(x,y) + ||(x,y) − (x',y')|| + ρ
13:                 H_2D(x',y') ⇐ min(H_2D(x',y'), H_t)
14:             end if
15:         end for
16:     end for
17:     for all (x,y) ∉ V do
18:         H_2D(x,y) ⇐ DynamicProgramming(x,y,H_2D)
19:         h_hyb(x,y) ⇐ H_2D(x,y)
20:     end for
21: end procedure
```

in the invisible space and a non-holonomic path (the blue curve) in the visible space as shown in Fig. 6. This hybrid path cost alleviates the underestimate of a pure holonomic path of h_{max} for invisible states. Note that while the reference path in Fig. 2(c) simply consists of line segments, the path in Fig. 6 involves making the robot U-turn to reach the goal as that in Fig. 2(a) does, As the result, h_{hyb} matches closer to the optimal cost than h_{max} does.

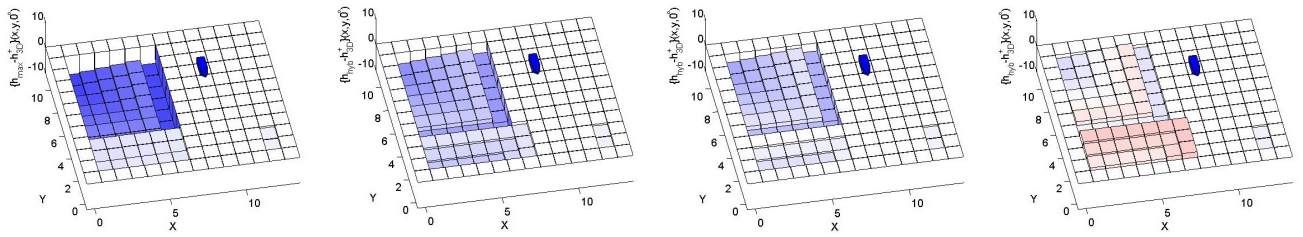
Properties of the hybrid heuristic can be summarized as follows:

- Since both h^-_{3D} and h^+_{2D} are admissible and consistent, the combined heuristic h_{hyb} is also admissible and consistent.
- h_{hyb} is constructed in same time complexity as h_{max} is. Note that h_{max} runs dynamic programming from the goal to all $2D$ states. Similarly, h_{hyb} runs dynamic programming for $2D$ states, among of which costs of visible states are copied from HLUT in constant time.
- h_{hyb} can be beneficial by saving memory for HLUT. Unlike h_{max} using HLUT for whole configuration space, h_{hyb} uses HLUT only for the visible space.

C. Threshold Distance

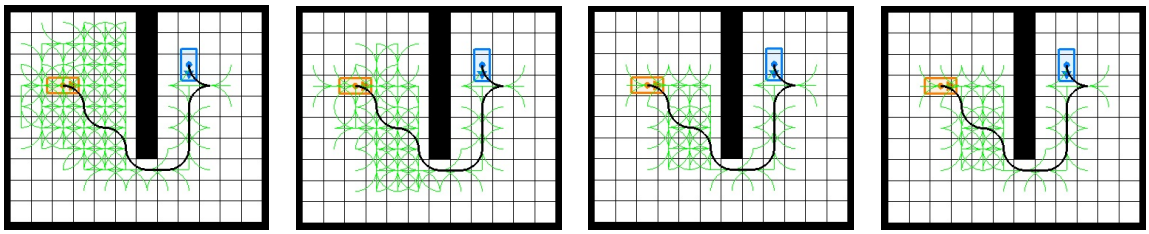
The threshold distance ρ controls the gap of heuristic values between visible and invisible spaces (Line 12 of Algorithm 2), and thereby compensates underestimating effect by holonomic paths in invisible space.

Fig. 7 shows the accuracies of h_{max} and h_{hyb} (with different ρ) to estimate h^+_{3D}, for the $\mathcal{C}_{space}$ and the goal depicted in Fig. 6. Recall that h^+_{3D} are the costs of the shortest paths through the non-holonomic-with-obstacle graph as in Fig. 2(a), and thereby perfectly estimate true optimal costs under the limit of the lattice resolution. While the graph in



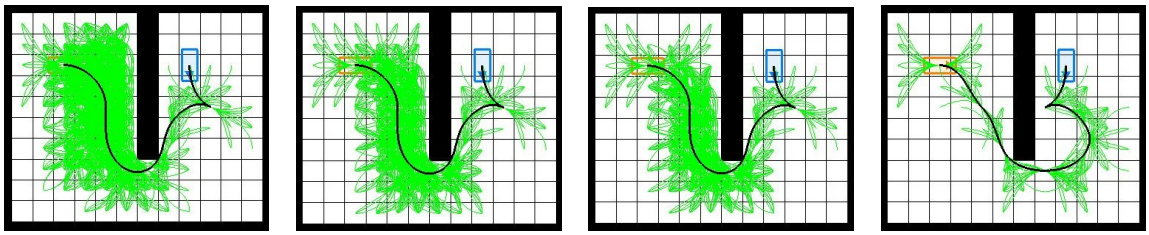
(a) $\{h_{max} - h^+_{3D}\}(x,y,0^\circ)$ (b) $\{h_{hyb} - h^+_{3D}\}(x,y,0^\circ)$ with $\rho = 0$ (c) $\{h_{hyb} - h^+_{3D}\}(x,y,0^\circ)$ with $\rho = 1$ (d) $\{h_{hyb} - h^+_{3D}\}(x,y,0^\circ)$ with $\rho = 3$

Fig. 7. Accuracies of h_{max} and h_{hyb}, to estimate optimal costs h^+_{3D}. Blueness (or redness) indicates underestimation (or overestimation).



(a) h_{max} (b) h_{hyb}, $\rho = 0$ (c) h_{hyb}, $\rho = 1$ (d) h_{hyb}, $\rho = 3$

Fig. 8. The A^* resulting paths (black curves) depending on applied heuristics. The green curves represent expanded controls.



(a) h_{max} (b) h_{hyb}, $\rho = 0$ (c) h_{hyb}, $\rho = 1$ (d) h_{hyb}, $\rho = 3$

Fig. 9. The resulting paths with the use of a higher resolution control set.

Fig. 7(a) represents $h_{max}(x,y,0^\circ) - h^+_{3D}(x,y,0^\circ)$, the graphs in Fig. 7(b)-7(d) represent $h_{hyb}(x,y,0^\circ) - h^+_{3D}(x,y,0^\circ)$ with ρ varying from 0 to 3. So, while negative values (blue) indicate underestimation, positive values (red) indicate overestimation. Zero values (white) indicate perfect estimation.

Referring to Fig. 7(a), while h_{max} nearly matches h^+_{3D} around the goal (blue cone), it underestimates in the area concealed behind the wall obstacle. This underestimated heuristics can lead to wasteful exploration of dead-ends. Applying h_{hyb} alleviates the underestimate as shown in Fig. 7(b). In addition, we can see that the underestimated heuristic grows to match h^+_{3D} as ρ grows, shown in Fig. 7(c)-7(d). At the same time, it causes overestimate in the lower left area. This may keep search from rolling backward at the area so as to speed up the search but to produce less optimal paths.

Fig. 8 shows the resulting paths by applying A^* with h_{max} and h_{hyb} with $\rho = 0, 1$, and 3. As expected, the h_{max} path undergoes wasteful exploration at the upper left area. On the other hand, h_{hyb} generates paths with less nodes expanded by virtue of better accuracies shown in Fig. 7. Fig. 9 shows the results with a higher resolution control set. We can see that the resulting path is generated with less expanded nodes but provides less optimality as ρ grows (Fig. 9(d)).

IV. Simulation

In this section we demonstrate search performance improvement by applying our new heuristic function compared to the maximum heuristic function. Simulations provided in this section were implemented in MATLAB on Intel Core i5 CPU 2.5 GHz and 4 GB RAM. The motion planner uses the control set with 16 discrete headings (as shown in Fig. 10) to generate a path.

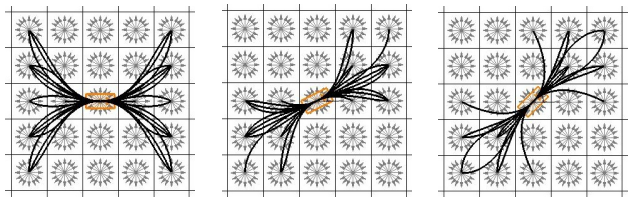


Fig. 10. The control sets allowed for the states with 0, $26.6^\circ = \arctan(1/2)$, and 45°. Reflecting the sets around $X-$ and $Y-$axes comprises overall control set with 16 total discrete headings.

Fig . 11 depicts several paths planned by applying A^* with h_{max} in the top row and with h_{hyb} in bottom for identical

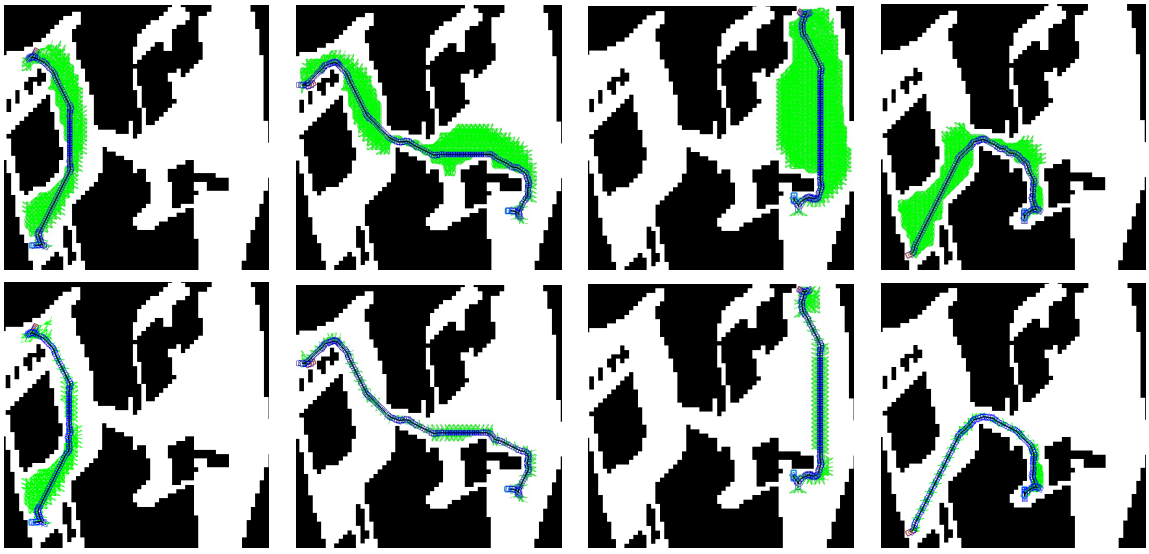
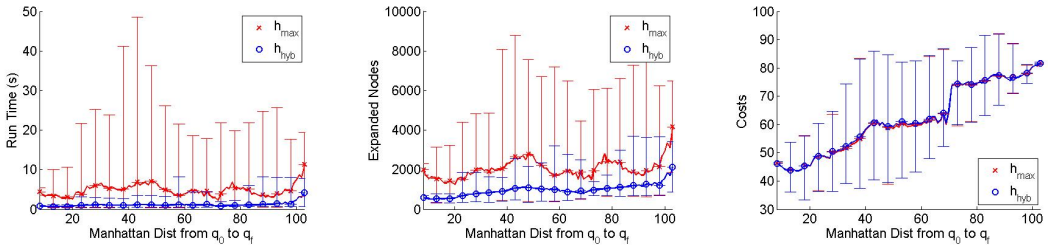


Fig. 11. Resulting paths by applying A^* with h_{max} (top) and h_{hyb} (bottom).



(a) (b) (c)

Fig. 12. Simulation performance depending on manhattan distance between the initial and goal states.

situations at each column. The map encodes Tampere University's Mobile Machine Lab area into a $50m \times 50m$ binary matrix with $0.5m$ resolution. The initial and goal states are randomly selected such that the initial state is invisible from the goal, since h_{hyb} is equivalent to h_{max} when the initial state is visible. Here, the threshold cost ρ was set to 5.

Fig. 12 shows the comparison of planning performance with h_{hyb} versus that with h_{max} for 15000 test cases. It highlights the benefits of applying the hybrid heuristic function. Runtime presented in Fig. 12(a) is slower than that of [2]. This is because the simulation in this paper has been implemented in Matlab which is higher-level than C++ used in [2]. However, h_{hyb} (blue circles) clearly outperforms h_{max} (red crosses) in terms of runtime (Fig. 12(a)) and expanded nodes (Fig. 12(b)). The data shows that planning with h_{hyb} is about five times faster than planning with h_{max}. The improvement in states expanded is greater than a factor of 1.5. On the other hand, planning with h_{hyb} provides slightly worse path costs than planning with h_{max}, but with less than a factor of 1.02, as shown in Fig. 12(c).

V. Conclusion

This paper proposes a new efficient heuristic estimate function to speed up non-holonomic motion planning. The heuristic is computed in three steps: 1) dividing cartesian space into visible and invisible spaces, 2) copying pre-computed non-holonomic-without-obstacle heuristics to the visible states, and 3) running dynamic programming on the invisible states by accumulating the visible heuristics with holonomi-with-obstacle costs. The numerical simulations demonstrate that the hybrid heuristic leads to an order of magnitude performance improvement, compared to existing heuristics [1] and [2].

References

[1] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. *International Journal of Robotics Research (IJRR)*, 28:933–945, 2009.

[2] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, Chicago, USA, June 2008.

[3] R. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3375–3380, October 2006.

[4] S. Koenig and M. Likhachev. D* lite. In *the AAAI Conference of Artificial Intelligence (AAAI)*, pages 476–483, 2002.

[5] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. pages 1177–1183, 2007.

[6] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, pages 3231–3237, August 2005.