Advanced Topics in Robotics 236610
Final Project - T*

Pinsky Doron <doron.pinsky@campus.technion.ac.il> 305347098
Shienman Moshe <smoshe@campus.technion.ac.il> 038057105

March 17, 2020

Introduction

In this project we implement the Time-Optimal Risk-Aware Motion Planning for Curvature-Constrained Vehicles (T*) algorithm as presented in Wettergren et al. [1]. In this paper, the authors address the problem of planning a collision-free path, to a goal state, in minimal time for an autonomous vehicle, which is naturally subject to kinematic constraints, such as bounded curvature and bounded turn rate. Dubins [2] used a geometric approach and showed that the shortest path between a pair of vehicle poses, for a vehicle with kinematic constraints that moves at a constant speed in an obstacle-free world, must be one of the following 6 types: RSR, RSL, LSR, LSL, RLR, and LRL, where L(R) refers to a left (right) turn with maximum curvature, and S indicates a straight line segment. Dubins-like path planning is popular because the resulting path consists of optimized parametric curves that can be expressed analytically and computed quickly.

As autonomous vehicles can travel at variable speeds, the time-optimal path can be different from the shortest path. Wolek et al. [3] derived a solution to find the time-optimal path for a curvatue-constrained variable-speed vehicle in an obstacle-free space. Under the assumption that the vehicle can travel in two different speeds, they identified a sufficient set of 34 candidate paths, where each candidate path contains circular arcs or straight line segments on which the vehicle can travel in either of the two speeds. However, Wolek et al. [3] do not consider the presence of obstacles. Lazard et al. [4] show that the problem of deciding whether a curvatue-constrained collision-free path exists between two given poses in the presence of polygonal obstacles is NP-hard, which means that no-exact algorithm exists for curvature-constrained time-optimal motion planning in arbitrary environment. The T* algorithm uses the approach presented by Wolek et al. [3] to approximate a solution to this problem in the discrete domain.

Wettergren et al. [1] also address the issue of vehicle safety. The authors propose a continuous risk function based on the concept of collision time, which is the time in which the vehicle can hit an obstacle along its heading direction, if it loses control. A vehicle is deemed safe if its collision time is greater than the time it takes to stop, maneuver around, or re-gain its control. The concept of collision time considers the complete information about the vehicle state, including its location with respect to the obstacles, heading angle and speed. As this is not the main focus of our work, we consider paths where safety is ensured by considering the vehicle as a polygon (a circle with radius r) and compute the C-space obstacles is a similar fashion to what we saw in the lectures.

The T* algorithm constructs a discrete configuration space and uses a grid-based A*-like search for motion planning where motion primitives are optimized over both the risk (vehicle safety) and time (time-optimal path).

Problem Formulation

We consider maps $\mathbb{M}$, containing obstacles, discretized into a set of unique grid cells, $\mathbb{C} = \{c_\alpha \subset \mathbb{R}^2, \alpha = 1, ..., |\mathbb{C}|\}$. Each grid cell $c_\alpha \in \mathbb{C}$ is of size 1×1 and is encoded with a symbolic state $s_\alpha \in \{O, F\}$ where $O \equiv obstacle$ and $F \equiv free$. Specifically, $s_\alpha = O$ if the cell is (partially) occupied by an obstacle; otherwise $s_\alpha = F$. The set of all free grid cells is defined as

$\mathbb{C}_{free} = \{c_\alpha \in \mathbb{C} : s_\alpha = F\}$. Let $\mathbb{O} = \{(x_\alpha, y_\alpha) \in c_\alpha : c_\alpha \in \mathbb{C}_{free}\}$ be the set of center positions of all obstacle-free cells. Let $\Theta = \{\frac{2\pi l}{L} : l = 0, ..., L-1\}$ be the set of $L \in \mathbb{N}^+$ heading angles and Let $\mathbb{V} = \{v_{min}, 1\}$ be the set of speeds. The configuration space is defined as:

$$\mathbb{Q} = \mathbb{O} \times \Theta \times \mathbb{V}$$

The size of $\mathbb{Q}$ relies on the size $\mathbb{C}_{free}$ and the value of L. Although a larger L could potentially produce better results, it will also lead to higher computational complexity. Fig. 1 shows the state expansion for different L. Our implementation supports defining L. To compare with the paper results, we used the 8-orientation state expansion with $L = 8$ in our experiments.

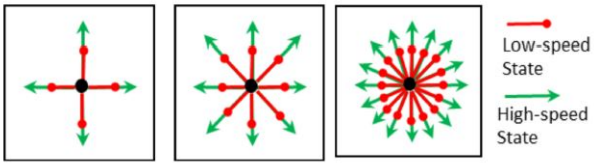


Fig. 1. State expansion in each cell when $L = 4, 8$ and 16.

The vehicle motion is described as:

$$\begin{cases} \dot{x}(t) = v(t)cos\theta(t) \\ \dot{y}(t) = v(t)sin\theta(t) \\ \dot{\theta}(t) = u(t) \end{cases}$$

where $(x, y, \theta) \in SE(2)$, u is the turn rate and v is the speed. It is assumed that the autonomous vehicle is capable of traveling at either $v_{min} \in \mathbb{R}^+$ or $v_{max} = 1$ which are normalized based on Wolek et al. [3]. Note that the Dubins path considered the special case where v is constant. The turn rate u is symmetric and bounded, s.t., $u \in [-u_{max}, u_{max}]$ where $u_{max} \in \mathbb{R}^+$ is the maximum turn rate and $\pm$ sign refers the left/right turn. The maximum and minimum turning radii of the vehicle are defined as $R = \frac{1}{u_{max}}$ and $r = \frac{v_{min}}{u_{max}}$. We denote the state of the vehicle inside a map m as $p^m = (x, y, \theta, v)$. The admissible control must be piece-wise continuous and drive the vehicle from p^m_{start} to p^m_{goal} while avoiding obstacles and satisfying the boundry conditions where $v \in [v_{min}, v_{max}]$ and the turn radius is $\in [r, R]$. Notice that this means we allow the vehicle to switch between v_{min} and v_{max} without considering acceleration.

T* Algorithm

Compared to Dubins curves, the vehicle can travel at variable speeds. Thus, the time-optimal path can contain arcs of different truning radii. For any pair of states p^m_i and p^m_{i+1} Wolek et al. [3] showed that the sufficient set, which guarantees to contain the time optimal path in the absence of obstacles, consists of 34 candidates paths, as shown in Table 1. Each path consists of up to five segments, where each segment could be one of the following:

1. Bang arcs (B), where the vehicle turns at maximum speed v_{max} and maximum turn rate u_{max} (i.e., with radius R).
2. Cornering arcs (C), where the vehicle turns at minimum speed v_{min} and maximum turn rate u_{max} (i.e., with radius r).
3. Straight line segments (S), where the vehicle moves at maximum speed v_{max}.

Any circular arc (B or C) can be either left or right. Each candidate path is read from left to right and consecutive turns within parentheses are of the same direction.

No.	Path Type[1]	Direction[2]	No.	Path Type	Direction
1	(B)S(B)	LSL	18	(B)S(BC)	LSR
2	(B)S(B)	LSR	19	(B)S(BC)	RSL
3	(B)S(B)	RSL	20	(B)S(BC)	RSR
4	(B)S(B)	RSR	21	(CB)(BCB)	LL
5	(BCB)(B)	LL	22	(CB)(BCB)	LR
6	(BCB)(B)	LR	23	(CB)(BCB)	RL
7	(BCB)(B)	RL	24	(CB)(BCB)	RR
8	(BCB)(B)	RR	25	(CB)S(B)	LSL
9	(B)(BCB)	LL	26	(CB)S(B)	LSR
10	(B)(BCB)	LR	27	(CB)S(B)	RSL
11	(B)(BCB)	RL	28	(CB)S(B)	RSR
12	(B)(BCB)	RR	29	(C)(C)(C)	LRL
13	(BCB)(BC)	LL	30	(C)(C)(C)	RLR
14	(BCB)(BC)	LR	31	(CB)S(BC)	LSL
15	(BCB)(BC)	RL	32	(CB)S(BC)	LSR
16	(BCB)(BC)	RR	33	(CB)S(BC)	RSL
17	(B)S(BC)	LSL	34	(CB)S(BC)	RSR

[1] B is a bang arc, C is a cornering arc and S is a straight line segment. The parentheses are used to indicate consecutive turns of the same direction.
[2] L is a left turn, R is a right turn and S means moving straight.

Table 1. The set of candidate paths between any pair of states.

Each candidate path has to be optimized for its parameters (i.e., the angles for arc segments and the lengths of straight line segments) to achieve time optimality, which is a nonlinear constrained optimization problem where the total time cost for each candidate path between a pair of states is the summation of the cost for each segment. Since each candidate path is requied to move exactly from p_i^m to p_{i+1}^m, there are five constraints which include: the total displacement along each axis, the total change in heading angle and the speeds specified by the first and last arc types. These conditions ensure the continuity in position, heading and speed from p_{start}^m to p_{goal}^m. The authors follow the solution of Wolek et al. [3] and use the nonlinear solver $IPOPT$ (Interior Point OPTimizer). The time optimal path between a given pair of states is the candidate path with the least time cost.

To avoid computational burden during motion planning, as the grid size, robot speeds and potential neighbours of each state are known a priori, the authors construct the Optimized Candidate Paths for State pairs (OCPS) table, that contains the optimized candidate paths for all possible pairs of states p_i^m and p_{i+1}^m. Specifically, consider a state p_i^m located in a center cell and all possible states p_{i+1}^m located in its 3×3 neighborhood. Since p_i^m has $8 \times 2 = 16$ choices corresponding to 8 directions and 2 speeds, and p_{i+1}^m has $8 \times 8 \times 2 = 128$ choices corresponding to 8 positions in the neighborhood, 8 directions and 2 speeds, the total number of 2048 pairs are considered. However, by exploring symmetry, only 272 pairs must be optimized for, while the rest can be derived accordingly.

The planning process is an A*-like search. Computing the time cost between a certain state pair p_i^m and p_{i+1}^m, is done by querying the OCPS table to obtain the set of optimized candidate paths that correspond to the speeds, orientations and relative locations of p_i^m and p_{i+1}^m. If a candidate path is not collision free, it is assigned with a cost of ∞. The candidate path with the lowest cost is returned to the A* algorithm and is considered as the cost from p_i^m and p_{i+1}^m (relative g value).

Our Implementation

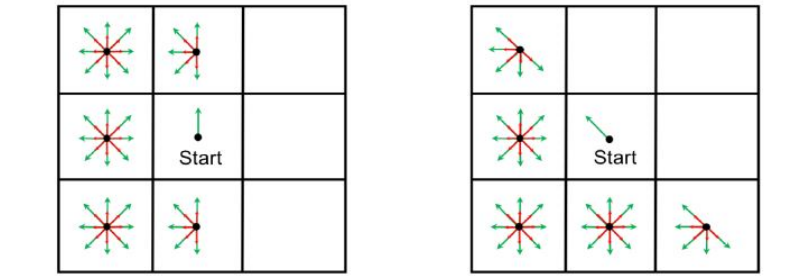
All code is available on *https://github.com/doron2/tstar*. It was implemented and tested under WSL (Windows Subsystem for Linux) using Ubuntu 18.04 LTS distribution. The github repository contains instructions and scripts to install all dependencies as well as sample maps and a run example. The T* algorithm, Dubins paths and the IPOPT solvers are implemented in c++

and compile to dll's (shared objects in linux - so's) with python wrappers. The map environment handler, config parser and display are all implemented in python3.

To compare with the original article, we implemented both the analytic closed form solution of dubins paths (*python/include/dubins.h*) and the T* algorithm inside an A* like framework.

User can provide different setups and configurations in *python/config.toml* . Specifically, one can define the map, the start and goal states, the cost and heuristic functions to use (euclidean, dubins or var speed dubins), the different speeds ($v_{min}, v_{max}, u_{max}$) and the radius of the robot for vehicle safety (zero counts as a point robot).

As in the original paper, to avoid computational burden during motion planning, we construct the OCPS table in a preprocessing stage. We modified the implementation of Wolek et al. [3] using the IPOPT open source library to construct the OCPS table for all 2048 state pairs. As suggested by Wettergren et al. [1], by exploring symmetry, we identified only 272 unique pairs that must be optimized for as can be seen in Fig 2. The rest are derived accordingly by mirroring over the appropriate axis and rotating.



(a) Case 1: Start state facing up, down, left, right (b) Case 2: Start state facing diagonally

Fig 2. Unique trajectories between 2 states.

The OCPS table is saved to a file, containing the specific parameters (speeds), and loaded uppon new session to avoid re-calculating in each new run. If one should change the parameters, the table obviously needs to be re-calculated.

Once the OCPS table is calculated, we use an A* like search algorithm where each state p_i^m is characterized by the following time-cost functions:

- cost function (g) : the cumulative cost from p_{start}^m to p_i^m as $g(p_{start=0}^m, p_i^m) = \sum_{i=0}^{i-1} J(p_i^m, p_{i+1}^m)$ where $J(p_i^m, p_{i+1}^m)$ is the time-optimal path between consecutive states found by querying the OCPS table.
- heuristic function (h): the heuristic from p_i^m to p_{goal}^m as $h(p_i^m, p_{goal}^m)$ which is determined by the length of the shortest Dubins path (with constant speed) using the small turning radius r divided by the maximum speed v_{max}.
- f the total cost for a state p_i^m (used as the priority in the A* open list) is defined as $f(p_i^m) = g(p_{start}^m, p_i^m) + h(p_i^m, p_{goal}^m)$.

Pruning

In a similar manner to Wettergren et al. [1], we implemented a three-step adaptive state pruning technique, as shown in Fig 3. Consider a state p_i^m and the baseline 8-orientation 2-speed in its neighbour cells, as shown in Fig. 3(a).

1. Obstacle-based Pruning: The states close to and facing obstacles or boundaries are considered as inevitable collision states, thus they are pruned, as shown in Fig. 3(b). Note that this is true only in the case where the lower turn radius r is larger than half of the size of a grid cell minus the radius of the robot.

2. Speed-based Pruning: In open regions away from obstacles, the vehicle is expected to travel at the highest speed to minimize the time cost, while the low-speed states are typically useful near obstacles to allow turning with a smaller turning radius for better controllability. Therefore, the low-speed states in the cells located far from obstacles can be pruned, as shown in Fig. 3(c).

3. Heading-based Pruning: Since the states with a diagonal heading and in an opposite direction to the goal will very likely produce higher costs, they are less likely to be part of the optimal state sequence. Thus, they can be dynamically identified and removed from $\mathbb{Q}$. Note that the states with non-diagonal heading angles of $0, \pi/2, \pi$ and $3\pi/2$ must be retained to ensure the completeness of the algorithm. In this regard, first connect the centers of each cell and using a straight line as shown in Fig. 3(d). Then, for each state with a diagonal heading, compute the angle $\zeta \in (-\pi, \pi]$ formed with the corresponding line. If $|\zeta| > \eta$, where $\eta \in (0, \pi]$ is a pre-defined threshold, then such state is pruned. Fig. 3(d) shows an example when $\eta = \pi/2$.

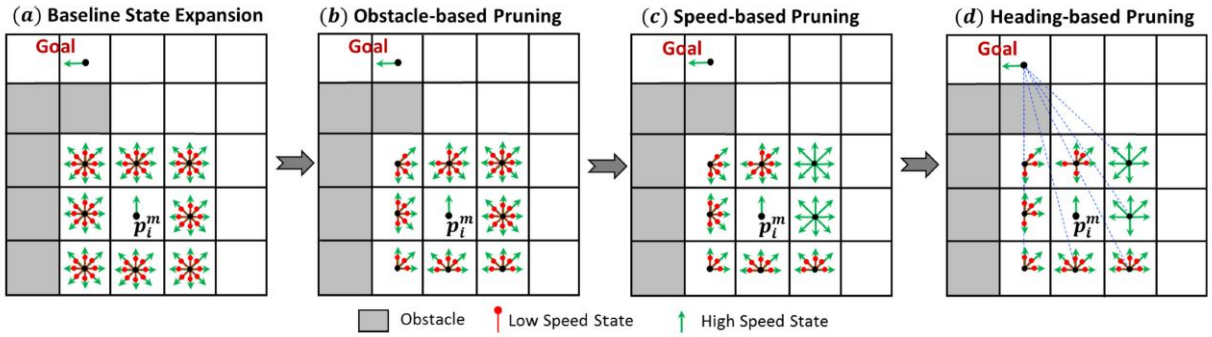


Fig 3. Illustration of the adaptive state pruning wihtin each cell

Experimental Results

To compare with the original paper of Wettergren et al. [1], we first constructed the maps presented in the paper. We used the same start and goal states and the same speed parameters.

Dubins path planner

We start by comparing the Dubins path planner where the robot is able to move between states only by using Dubins paths. Thus, the cost function of this planner is determined by the shortest dubins path between state pairs that is also collision free, while the heuristic function is determined by the length of the shortest Dubins path using the small turning radius r divided by the maximum speed v_{max}. For low speed Dubins:

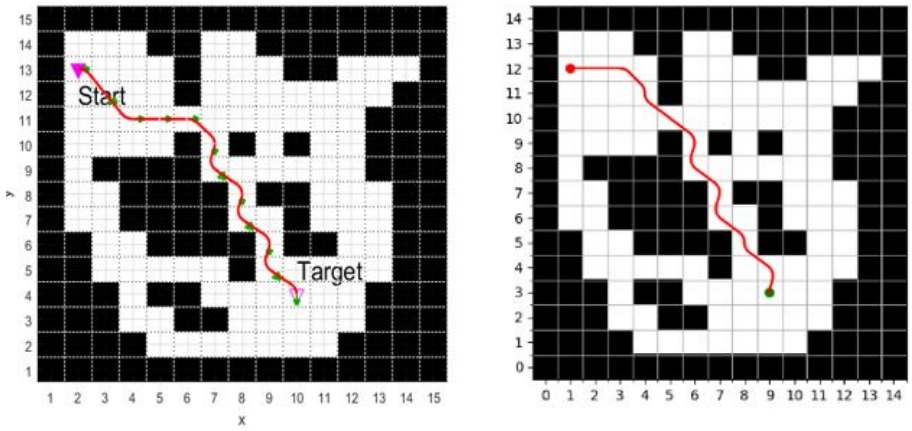


Fig 4. Wettergren et al. low speed dubins Vs. our low speed dubins

In both the original paper and our implementation, the final cost is 55.95 sec. As we do not know how the original paper implemented A* (in terms of order in the open list if two f values for different states are equal), it is expected that a different path with an identical cost is chosen.

For high speed Dubins:

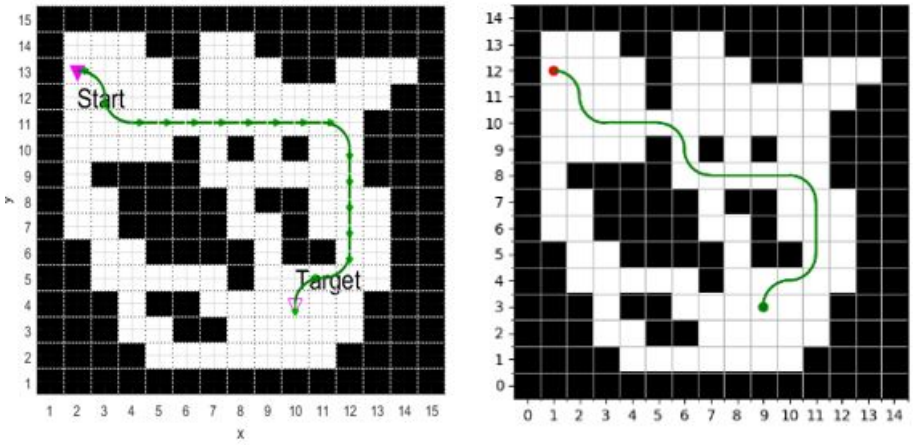


Fig 5. Wettergren et al. high speed dubins Vs. our high speed dubins

In both the original paper and our implementation, the final cost is 35.99 sec. As we do not know how the original paper implemented A* (in terms of order in the open list if two f values for different states are equal), it is expected that a different path with an identical cost is chosen.

T* - Variable speed Dubins path planner

As we do not know the exact safety margins used in Wettergren et al. [1] (unfortunately the paper does not accurately provides them) we compare the results obtained in the paper with both a point robot and a robot with a radius of 0.05 :

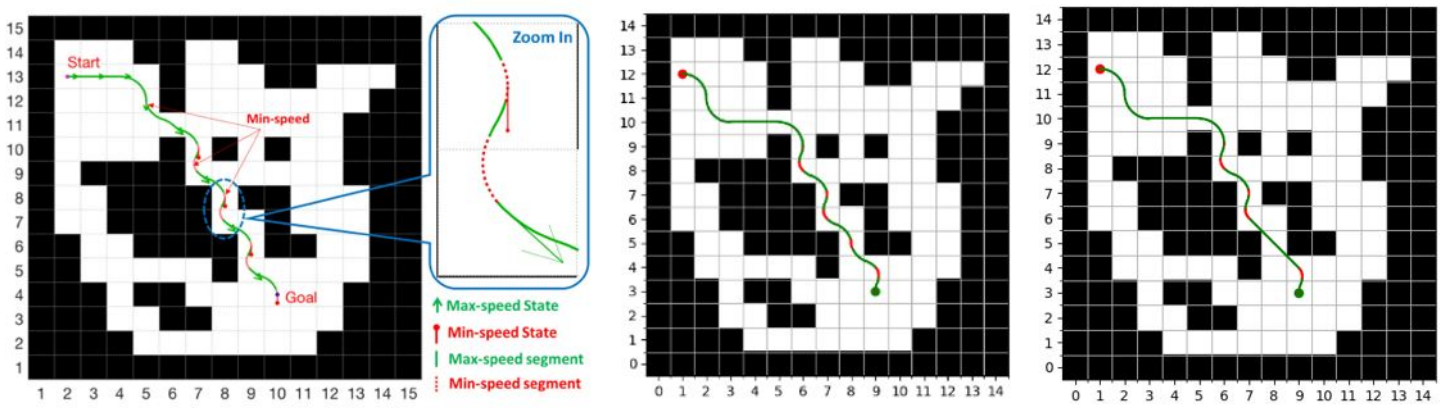


Fig 6. Wettergren et al. T* Vs. our T* with a robot with radius $r = 0.05$ Vs. our T* with a point robot

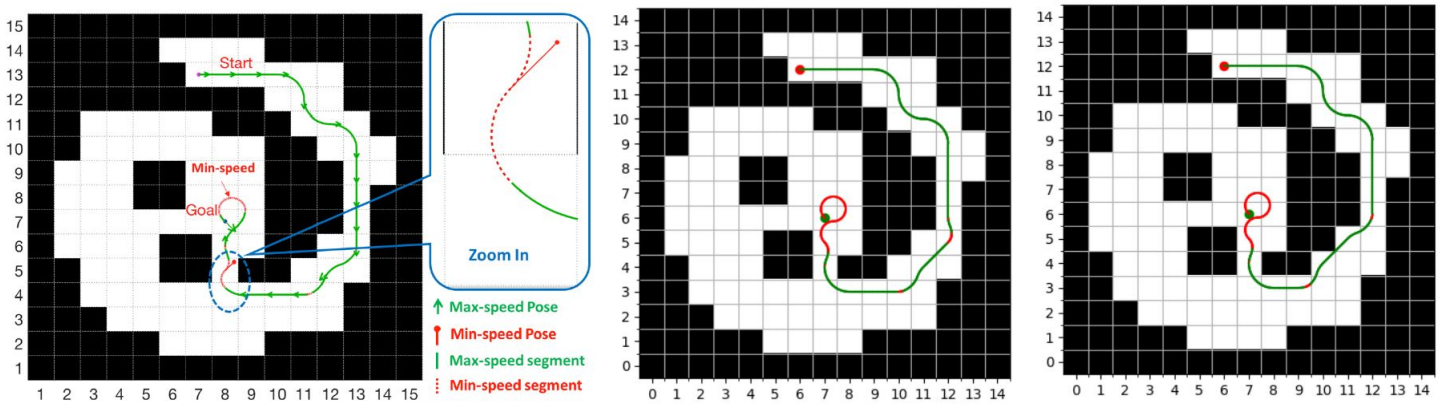


Fig 7. Wettergren et al. T* Vs. our T* with a robot with radius $r = 0.05$ Vs. our T* with a point robot

As we can see, in both maps (Fig 6. and Fig 7.) the paths chosen are very similar in both the grid cells chosen and the transitions between state pairs (low and high speed segments in red and greed). As before, as we do not know how the original paper implemented A* (in terms of order in the open list if two f values for different states are equal), it is expected that a different paths with similar costs are chosen. The time-cost reported by Wettergren et al. [1] for the map in Fig 6. is 34.51 while our time-cost using a radius $r = 0.05$ is 32.88. The time-cost reported by Wettergren et al. [1] for the map in Fig 7. is 54.24 while our time-cost using a radius $r = 0.05$ is 48.88. It is no surprise that using a point robot, produced better paths in terms of time-cost with 31.87 for the map in Fig 6. and 47.26 for the map in Fig 7. as the free c-space contains additional points, enabling the algorithm to find better solutions.

State Pruning

In order to evaluate the complexity reduction resulting from pruning, we compared the number of states expanded. A comparison was made for planning using Dubins paths, with minimum and maximum speeds, and for planning using variable speed Dubins. $\eta = \pi/2$ was chosen for heading based pruning in all three scenarios.

	states expanded without pruning	states expanded with pruning	improvement
min speed Dubins	298	114	61.8%
max speed Dubins	324	119	63.3%
Variable speed Dubins	720	258	64.2%

As we can see, using pruning can considerably reduce the number of states expanded (this is obviously scenario dependent; map, obstcales, speeds etc..).

Conclusions And Future Work

In this project, we implemented the T* algorithm based on the paper by Wettergren et al. [1]. T* approximates a solution to a time-optimal motion planning for curvature-constrained vehicle in a discrete space. To validate our implementation, we used the same experimental settings (maps and scenarios) as the original paper for comparison. Both Dubins path planner (using constant speed) and variable speed Dubins paths show similar results.

During our work we identified few issues that can lead to future research:

1. The authors assume that the vehicle can change velocities instantly and thus control is only piece-wise continuous. This assumption is not valid in the real world and future work can consider accelerations in order to have a more realistic model.

2. As the original paper uses an A* like algorithm for motion planning while discretizing the map, it forces the solution to go thru the center of each grid cell. Possible future research direction can include using a motion planning algorithm that operates in the continuous space, like PRM*, where each edge in the graph can be derived using a variable speed dubins path.

3. As the authors only consider a constant grid discretization, we could suggest to include a dynamic refinement of the map grid repeatedly, in areas close to obstacles or boundries, within the A* framework to simulate a continuous space. We could also suggest including a set of speeds between $[v_{min}, v_{max}]$ and additional turn rates u to produce better solutions. Obviously, this will induce a computational cost as the number of states will grow, but, it could be interesting to characterize this tradeoff.

References

[1] J. Song, S. Gupta and T. A. Wettergren, "T*: Time-optimal risk-aware motion planning for curvature-constrained vehicles". IEEE Robotics and Automation Letters, vol. 4, no. 1, pp. 33-40, Jan.2019.

[2] L. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents." American Journal of Mathematica, vol. 79, no. 3, pp. 497–516, 1957.

[3] A. Wolek, E. Cliff, and C. Woolsey, "Time-optimal path planning for a kinematic car with variable speed". J. Guid., Control, Dyn., vol. 39, no. 10, pp. 2374–2390, 2016.

[4] S.Lazard, J.Reif, and H.Wang. "The complexity of the two dimensional curvature-constrained shortest-path problem", in Proc. 3rd Int. Workshop Algorithmic Found. Robot., Houston, TX, 1998, pp. 49-57.