Numerical Approach to Reachability-Guided Sampling-Based Motion Planning Under Differential Constraints

Scott Drew Pendleton, Wei Liu, Hans Andersen, You Hong Eng, Emilio Frazzoli, Daniela Rus, and Marcelo H. Ang, Jr.

***Abstract*—This paper presents a new method for motion planning under differential constraints by incorporating a numerically solved discretized representation of reachable state space for faster state sampling and nearest neighbor searching. The reachable state space is solved for offline and stored into a "reachable map" which can be efficiently applied in online planning. State sampling is performed only over states encompassed by the reachable map to reduce the number of unsuccessful motion validity checking queries. The nearest neighbor distance function is revised such that only reachable states are considered, with states which are unreachable or only reachable beyond a designated time horizon disregarded. This method is generalized for application to any control system, and thus can be used for vehicle models where analytical solutions cannot be found. Greater improvement is expected for more constrained systems where motion checking cost is relatively high. Simulation results are discussed for case studies on a holonomic model and a Dubins car model, both with maximum speed limitation and time included as a dimension in the configuration space, where planning speed (measured by tree growth rate) can be improved through reachability guidance in each system by at least a factor of 3 and 9, respectively.**

***Index Terms*—Motion and path planning, nonholonomic motion planning.**

I. Introduction

ONE of the fundamental challenges of robotics is motion planning - solving for a set of control actions which can bring a system from an initial condition to a goal condition [1]. While motion planning can by applied to a wide range of robotic platforms including quadrotors, articulating arms, autonomous cars, etc., the concepts may also be applied to various areas outside of traditional robotics including graphics animation and medical surgery [2]. Despite the high complexity of even relatively basic motion planning problem formulations [3], there has been great progress in this field of research, where recent formulations of asymptotically optimal sampling-based planners such as RRT* (optimal Rapidly-exploring Random Tree) [4] have become especially popular and are applied to many real-time applications, including public trials with autonomous buggies [5].

Besides bringing a robot from its initial condition to goal condition, a motion planning solution must also obey kinodynamic motion constraints and avoid obstacles while possibly optimizing an objective function. Sampling-based approaches find such a solution by randomly sampling a set of states from a configuration space and checking their connectivity, while rejecting samples and connections which intersect with obstacle occupied space. Popular sampling-based methods are probabilistically complete in the sense that the probability of finding a feasible solution, if one exists, approaches one as the number of samples increases.

State sampling can be made more efficient by limiting the states that are sampled to only those from within a set of states known to be reachable from the initial condition given the robot's kinodynamic constraints applied to an obstacle free environment. Likewise it is only beneficial to check for connectivity between neighboring states when they fall within each other's reachable sets; checking any states that are nearby by Euclidean distance metric but not reachable within a short period of time given kinodynamic constraints is a waste of computational effort. Adding Reachability Guidance (RG) to state sampling and Nearest Neighbor (NN) searching can provide significant efficiency boosts to planning speed, especially for systems where motion is highly constrained or the motion checking cost is high, and the standard naive approaches of uniform sampling over a hyperrectangle and NN searching by Euclidean distance metric would otherwise result in slow planner performance.

One of the most fundamental differential constraints in a system is time, where time t must increase at a constant rate $\dot{t} = 1$. Whether or not time is explicitly included as a state parameter, other state parameters will typically have differential constraints with respect to time, such as velocity and/or acceleration limits. Robot differential constraints are applied to generate velocity profiles, which may be solved for in a decoupled manner only over the chosen geometric path [5], [6], or in a direct integrated

Manuscript received September 9, 2016; revised November 29, 2016; accepted December 26, 2016. Date of publication January 11, 2017; date of current version March 10, 2017. This paper was recommended for publication by Associate Editor L. Tapia and Editor N. Amato upon evaluation of the reviewers' comments. This work was supported by the National Research Foundation, Prime Ministers Office, Singapore, under its CREATE programme, Singapore-MIT Alliance for Research and Technology, Future Urban Mobility (FM) IRG.

S. D. Pendleton, W. Liu, H. Andersen, and M. H. Ang Jr. are with the National University of Singapore, Singapore 138602 (e-mail: scott.pendleton01@u.nus.edu; liu_wei@u.nus.edu; hans.andersen@u.nus.edu; mpeangh@gmail.com).

Y. H. Eng is with the Singapore-MIT Alliance for Research and Technology, Singapore 119223 (e-mail: youhong@smart.mit.edu).

E. Frazzoli and D. Rus are with the Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: frazzoli@mit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LRA.2017.2651940

manner simultaneously with geometric path solving over every connection in the tree as it is built [7]–[11]. In order to perform predictive planning, that is planning around predictions of environmental changes such as obstacle movements, each state must have an appended time dimension for collision checking and/or cost heuristic evaluation. This is contrasted against the popular reactive planning approach, where a geometric path is found either assuming static environment or enlarged obstacle spaces to account for potential movements and replanning is performed often to get real-time updates on the environment state. While the RG proposed in this work can still be applied to reactive planners, the experiments discussed in later sections use the more challenging predictive planning. Reactive planning and decoupled differential constraint handling can result in very inefficient trajectories or failure to find a trajectory due to the decoupling or expanded approximation of obstacle occupied space. Conversely, predictive planning with direct integrated differential constraint handling can overcome these shortcomings but is more computationally complex. These issues are the primary motivation to improve the efficiency of differential constraint handling through RG.

The primary contribution of this work is (i) a novel method of deriving a numerically solved discretized representation of reachable maps offline, where (ii) the reachable map is applied as a prior to guide state sampling and NN searching in online sampling-based motion planning with replanning. RG sampling and NN searching is shown to improve planning speed when applied to two example cases of (i) predictive planning for a holonomic robot given a maximum speed constraint where the reachable set is solved for analytically and (ii) predictive planning for a car-like non-holonomic robot subject to a maximum speed constraint and maximum turning radius where the reachable set is solved numerically.

II. Related Works

Several other works have presented analytical approaches to incorporate RG into motion planning. The asymptotic optimality of RRT* was shown to extend to a reachability guided variant where the reachable subspace was represented as a hyperrectangle derived from the linearized dynamics approximation of a system through the Ball Box Theorem presented in [9], where the "box" was an under-approximate of the true reachable subspace. A similar approach was taken in [11] to prove asymptotic optimality in differential constraint handling variants of PRM (Probabilistic RoadMaps) and FMT (Fast Marching Tree). Another asymptotically optimal sampling based algorithm to handle differential constraints, Goal-Rooted Feedback Motion Tree (GR-FMT) was presented in [12], limited in application to controllable linear systems with linear constraints. An analytical method for solving a two point boundary value problem subject to kinodynamic constraints was presented in [10], which could be used for finding optimal state-to-state connections and NN searching but was limited to systems with linear dynamics.

A machine learning approach was taken in [13] to query for whether a state was reachable from a given base state, though this method required applying a Support Vector Machine (SVM) classifier over a feature set of 36 features for the Dubins car model, where online solving for 36 features could be relatively computationally expensive.

The Reachability Guided RRT planner presented in [14] relied on construction of Voronoi diagrams to build approximations of reachable sets rooted from each graph node for sampling biasing, where Euclidean distance metric was still used for NN searching. This method may not easily be extended to higher dimensional spaces, as Voronoi diagrams are then no longer easily constructed.

There are also relatively few planning methods that have been demonstrated to be effective for solving over a configuration space with an appended time dimension. Early works explored control sampling approaches [8], and recent state sampling works have made model simplifications to handle the differential constraints in an online manner, such as keeping to a constant ego robot speed [15].

This work presents a numerical method to achieve a better approximation of the reachable subspace than prior analytical methods can achieve for non-linear systems. This numerical method can furthermore be applied when an analytical solution cannot be found, and because the reachable subspace map is solved in an offline manner, online application of the numerical RG can be faster than analytical approaches (or the machine learning approach mentioned). It will also be shown that numerical RG can drastically improve predictive planning speed for planners solving over a configuration space with an appended time dimension.

III. Analytical Reachability

A. General Method

We consider a robot's reachable space to be the subset of the state space which is reachable given a robot's control restrictions, irregardless of obstacle presence (collision checking imposed at a later stage). Reachability Guidance (RG) is then a method to restrict planning to consider only states which fall within the robot's reachable space while building a path search tree, so as to speed up the planning process by ignoring known impossibilities. In this work, RG is incorporated in the context of state sampling and nearest neighbor (NN) searching. The planner is "guided" to sample only within the reachable space originating from the robot's initial state. Likewise, the planner is "guided" to consider two states as being potentially "nearby" each other only if one state from the pair is located within the reachable space originating from its paired state.

For some systems, the reachable space can be represented by simple analytical formulations thus there is no need for a discretized prior reachability map to be stored. However, methods for deriving analytical formulation can be case specific and not easily generalized for a wide variety of systems. Nevertheless, this work will first examine a simple system for which the reachability guided concept is more easily conceptualized. The reachable space for a 2D holonomic robot with a maximum speed constraint can be represented by common geometric shapes, as shown in Fig 1.

B. 2D Holonomic Robot With Maximum Speed Constraint

Consider a holonomic robot operating over a 2D plane with a maximum speed constraint. The configuration space $\mathcal{X}$ is then defined as $\mathcal{X} = \Re^2 \times \mathcal{T}$, where each state, $s \in \mathcal{X}$, is represented by a vector of Cartesian coordinates, x and y, and time t: $s =$

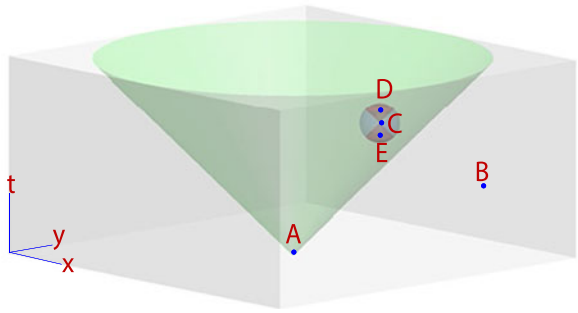

Fig. 1. Reachable space for 2D holonomic robot with speed constraint. The green cone is the valid state sampling volume, given a root state at A. State B is outside of the reachable state sampling cone, where states C, D, and E are inside. Given the small sphere as the Euclidean NN search region for state C, the red spherical sectors depict the reachable NN regions. State D is in state C's forward reachable set, and state E is in state C's backward reachable set.

Algorithm 1: Uniform Sampling of a Cone.

$r_{tmp} = \text{rand}[0, \nu_{max} * t_{max}]$;
$\alpha = \text{rand}[-\pi, \pi)$;
$t_{tmp} = \text{rand}[0, t_{max}]$;
if $r_{tmp}/t_{tmp} > \nu_{max}$ **then**
  $r = \nu_{max} * t_{max} - r_{tmp}$;
  $t = t_0 + t_{max} - t_{tmp}$;
else
  $r = r_{tmp}$;
  $t = t_0 + t_{tmp}$;
$x = x_0 + r\cos\alpha$;
$y = y_0 + r\sin\alpha$;
return $s = [x, y, t]$;

$[x, y, t]$. The control space $\mathcal{U}$ is composed of controls in speed, ν, and orientation, θ, such that each control $u \in \mathcal{U}$ is given as $u = [\nu, \theta]$. Given that a holonomic robot can move equally well in any direction regardless of heading, assume that the motion in any direction is constrained to a maximum speed $\nu_{\max}$, $\nu \in [0, \nu_{\max}]$. The dynamics of the system are governed by the following equations:

$$\dot{x} = \nu\cos\theta; \quad \dot{y} = \nu\sin\theta; \quad \dot{t} = 1 \tag{1}$$

The reachable area is then described by a cone centered at initial condition $[x_0, y_0, t_0]$ given by $\sqrt{(x-x_0)^2 + (y-y_0)^2}/(t - t_0) < \nu_{\max} \quad | \quad t > t_0$. This area can be sampled uniformly within a time horizon $t_{\max}$ by drawing from uniform distributions for polar coordinates r and α, as well as a uniform distribution for t, then making a conditional adjustment to r and α and solving for $s = [x, y, t]$ according to Algorithm 1.

Although more computation is required for generating each sample, the alternative of sampling evenly over each dimension for its respective bounded range would result in many samples being rejected after a motion check, i.e. a check that a direct path from the initial state to the sampled state obeys the maximum speed condition. In this system the volume of the reachable space is only 26.2% of the bounded state space volume (cone vs rectangular prism volume).

Similarly, the NN distance function, $dist_{\text{NN}}$, can be modified to only consider samples which fall within the maximum velocity cone as extended from the queried state, rejecting all other samples as being infinitely far away:

$$dist_{\text{NN}} = \begin{cases} \sqrt{\Delta x^2 + \Delta y^2} + w\Delta t, & \text{if } \frac{\sqrt{\Delta x^2 + \Delta y^2}}{|\Delta t|} \leq \nu_{\max} \\ \infty, & \text{otherwise} \end{cases} \tag{2}$$

where Δx, Δy, and Δt are the difference of values in x, y, and t dimensions respectively for the two queried states, and w is a weighting factor to optionally consider time duration of a path as more or less influential to the path cost than path length (here $w = 1$). Note that in the RRT* algorithm and in other asymptotically optimal sampling based planners, it is common to either consider neighbors within a set radius hypersphere, or a fixed number of neighbors, where either the hypersphere radius or number of neighbors considered decreases with increasing number of tree nodes. Depending on the maximum speed allowed, Eq. 2 may be more or less restrictive. If $\nu_{\max} = 1$, then the ratio of volume considered in Eq. 2 to the volume otherwise considered by a hypersphere is only 29.2% (spherical sector vs sphere volume).

IV. Numerical Reachability

A. General Method

In order to incorporate RG into motion planning, the robot's reachable space (as defined earlier in Section III) must be clearly discernible from non-reachable space. Besides maximum speed constraints, which are relevant to a majority of robotic systems, there can still be many other constraints imposed on the robot motion. When additional constraints are incorporated, it may no longer be easy to solve for reachability bounds analytically. Systems with non-linear dynamics and higher dimensional systems in particular pose difficulty for analytical methods. For example, analytical representations of reachable space for the non-linear Dubins car model, later explored in Section IV-B, were attempted in many of the works mentioned in Section II: the reachable space was under-approximated [9], linearized [10], or solved for in only x and y dimensions without indication for valid θ values [13]. However, so long as the system model is known, a discretized representation of the reachable space may be found by numerical methods. We call this discretized representation of the reachable space as a "reachable map," where all positions on the map (i.e. state locations) are labeled as "reachable" or "not reachable."

The primary steps to solving for a numerically generated reachability map are (i) approximate the reachable space through state propagation, building a "reachable graph" (V) of representative reachable states, then (ii) parse the reachable graph into a compressed data storage format for easier online referencing, where the output is called the reachable map ($\{M, \overrightarrow{min}, \overrightarrow{res}, R\}$). This two step process is presented via pseudo code in Algorithms 2 and 3. First, a "reachable graph" is generated (Algorithm 2) by taking the initial state as a zero vector $\overrightarrow{0}$, and iteratively propagating states forward in time by time step Δt up to time horizon $t_{\max}$ by applying a representative set of discrete controls $U \subset \mathcal{U}$ which span the extremal bounds of the control space, such that a graph of nodes V can provide good coverage of the reachable space. Generally more nodes can provide more coverage and therefore a better estimate of

Algorithm 2: Reachable Graph Propagation.

Input: $U, t_{max}, \Delta t$
Output: V
1: $V \leftarrow \{\vec{0}\}$;
2: **for all** $v \in V$ **do**
3: | **if** $(v[t] < t_{max})$ **then**
4: | | **for all** $u \in U$ **do**
5: | | | $s_{tmp} = u$ applied to v for duration Δt;
6: | | | **if** $(s_{tmp} \notin V)$ **then**
7: | | | | $V \leftarrow V \cup \{s_{tmp}\}$;
return V

Algorithm 3: Reachable Map Parsing.

Input: $V, \overrightarrow{res}$
Output: $M, \overrightarrow{min}, \overrightarrow{res}, R$
1: $\overrightarrow{min} \leftarrow \vec{0}; \overrightarrow{max} \leftarrow \vec{0}; R \leftarrow \emptyset$
2: $M[\vec{0}] =$ true;
3: **for all** $v \in V$ **do**
4: | **for all** $n \in [1, N]$ **do** //grow map bounds if needed
5: | | **if** $(v[n] < \overrightarrow{min}[n])$ **then**
6: | | | $\overrightarrow{min}[n] = \text{floor}(v[n]/\overrightarrow{res}[n]) * \overrightarrow{res}[n]$;
7: | | **else if** $(v[n] > \overrightarrow{max}[n])$ **then**
8: | | | $\overrightarrow{max}[n] = \text{floor}(v[n]/\overrightarrow{res}[n]) * \overrightarrow{res}[n]$;
9: | | resize M by $\overrightarrow{bounds}[n] \leftarrow [\overrightarrow{min}[n], \overrightarrow{max}[n]]$;
10: | | init $M[\overrightarrow{new}] =$ false, $\forall \overrightarrow{new}$ after resize;
11: | | $\overrightarrow{index}[n] = \text{floor}((v[n] - \overrightarrow{min}[n])/\overrightarrow{res}[n])$;
12: | $M[\overrightarrow{index}] =$ true; //register true at v's index
13: **for all** $m \in M$ **do**
14: | **if** $m ==$ true **then**
15: | | $R \leftarrow R \cup \{\overrightarrow{index}$ of $m\}$;
return $M, \overrightarrow{min}, \overrightarrow{res}, R$

the reachable space extents, but storage of large graphs for real time referencing may be unreasonable without a data compression scheme (hence subsequent Algorithm 3). Since reachable graph state propagation can be very time and memory consuming (runtime of several hours, file size exceeding 15 GB for later experiments), it is recommended to offload the random-access memory (RAM) burden by incrementally storing the graph to a file in read-only memory (ROM) as it is built. If the graph size is too large to store in short-term memory (RAM), the program can always iteratively read and write only the few relevant lines from a file needed to propagate states. This can also function as a safety measure to avoid data loss due to power disruptions or other potential computer faults which could result in RAM loss partway through the reachable graph propagation.

Note that in the graph building stage, an upper limit to the number of nodes, G, can by calculated by

$$G = \sum_{\tau=0}^{t_{\text{step}}} K^{\tau} \tag{3}$$

where K is the number of controls applied to each base node (size of U), and t_{step} is the number of time steps over which the graph is built. The number of nodes can be reduced by filtering for duplicates especially in early stages of graph propagation (line 2), which can significantly reduce code runtime and the RAM and/or ROM required for code execution. Duplicates could be found by simple linear searching or spatial partitioning, as with KD-trees, though duplicate searching may be time consuming for very large graph sizes. Parallel processing could also reduce runtime.

So as not to reduce the sampling-based planners' completeness to resolution completeness over the reachable graph, continuous sampling ranges should be used rather than drawing samples from a reachability graph directly. Toward this aim, the reachable map is built taking the reachable graph as an input (Algorithm 3), where the graph nodes are indexed into discretized hyperrectangle cells spanning the configuration space. As each vertex in the reachable graph is considered, the bounds, $\overrightarrow{bounds}$, of the reachable map are expanded in each dimension $n \in [1, N]$ (of the N dimensional state space) as necessary to span the vertex location (lines 3–3) and the map value is updated to "true" at $\overrightarrow{index}$, the corresponding index for the location of the vertex pose (line 3). All newly created map indices, $\overrightarrow{new}$, resulting from updated map bounds (line 3) are initialized with value "false" (line 3). Cells which do not contain any nodes are considered not reachable. The reachable map is then stored into a file for later access by the planner.

The final output is a map M which indexes all discretized hyperrectangles spanning an ego-centric configuration space (centered around $\vec{0}$ as the initial state) and gives a boolean for whether there are any states within each cell that are reachable. The state indicative of the minimum bound in the map, $\overrightarrow{min}$, and the resolution used to discretize each dimension, $\overrightarrow{res}$, also need to be stored as corresponding to the boolean map M. It is furthermore recommended to store a vector R specifying which indices in the map are reachable. Thus to perform uniform sampling over the reachable map, for each sample an indexing vector $\overrightarrow{r}$ would be drawn from the reachable vector R at random, then a state s would be taken uniformly at random from within the corresponding hyperrectangle:

$$s = \overrightarrow{min} + (\overrightarrow{r} + \overrightarrow{rng}) \odot \overrightarrow{res}; \tag{4}$$

where $\overrightarrow{rng}$ is an n-dimensional vector of independently generated random numbers in the range [0,1], and symbol $\odot$ indicates element-wise multiplication.

The reachable map can also be queried during NN searching such that when calculating distance between potentially nearby states, if neither state lies in each others' reachable map, then the distance is considered to be infinity, otherwise the standard distance calculation for the space is used:

$$dist_{\text{NN}}(s_1, s_2) = \begin{cases} dist(s_1, s_2), & \text{if } M\left[\text{floor}\left(\Delta s \odot \frac{1}{\overrightarrow{res}}\right)\right] \\ \infty, & \text{otherwise} \end{cases} \tag{5}$$

where $dist(s_1, s_2)$ is the standard distance function for the state space, e.g. the Euclidean norm $|\Delta s|$ (others possible), and $\Delta s = s_2 - s_1$ for parent state s_1 and queried neighbor state s_2. The output of $\text{floor}(\Delta s \odot \frac{1}{\overrightarrow{res}})$ is a vector to index the particular boolean value in M which represents the reachability of the hyperrectangle encapsulating state s_2 as measured from a frame of reference with s_1 as its origin.

The main risk of this method lies in the potential to compromise the completeness of the motion planner. Since state propagation is restricted to application of a limited discrete control set U over finite time steps, it is possible that the subsequently generated reachable map may report some false negatives such that a subset of the reachable space is incorrectly indicated to be unreachable. To reduce this risk, the resolution of the map should be carefully chosen in relation to the maximum incremental step change possible in each state dimension that results from the applied control set over chosen time increment Δt:

$$\overrightarrow{res}[n] > \max_{u \in U} \Delta s[n], \forall n \in [1, N] | \Delta t \tag{6}$$

By this choice of map resolution, the false negatives should be limited to the boundaries of the reachable space. The possibility of false negative occurrence will further be reduced by choosing smaller time step sizes and a greater number of discrete controls to propagate the reachable graph (where limitations are based on computer memory and graph size from Eq. 3). There will inevitably be several false positives reported by querying the reachable map as well due to cell decomposition in the reachable map parsing stage, but these are of less concern since they can easily be distinguished and filtered out during motion checking. Nevertheless, despite the limited completeness due to false negatives, the method can be useful in practice as it is well generalized and hence applicable to a wide range of varied control systems, and as will be shown in the later Section VI, it can drastically improve planning speed and success rates for the tested car-like robot system.

The complexity of building the reachable graph and subsequent reachable map also increases exponentially with respect to space dimension. The graph size may be reduced by reducing the number of time steps to counter this problem (Eq. (3)), though this would unfavorably either shorten the planning time horizon and/or reduce the reachable map resolution (Eq. (6)). However, since the reachable map is generated offline and only queried online as a fixed prior, the offline computational burden does not entirely rule out the viability of the method in higher dimensional spaces. The reachable map can be calculated offboard the robot on a computer with greater processing capabilities, where the onboard computer needs only to have enough memory to store the resultant reachable map (which is significantly smaller in size than the reachable graph). The most important factor for application of the method is the computational cost of online processes, where both RG sample generation (Eq. (4)) and RG NN distance calculation (Eq. (5)) have time complexity $O(n)$, linear growth with respect to state dimension; this is the same order of complexity as sampling and NN distance calculation by the standard sampling-based method.

B. Dubins Car Robot With Maximum Speed Constraint

This section will examine the Dubins car model, which enforces a minimum turning radius constraint by finding paths in $SE(2)$ space composed of piecewise circular arc segments and straight lines while still incorporating a maximum speed constraint. While it is commonly assumed in the Dubins car model that the car travels at a known constant speed and thus fixed minimum turning radius, in this work it will be assumed that the minimum turning radius, $\rho_{\min}$ is still fixed regardless of changes in speed, and any speed ranging from zero to $\nu_{\max}$ may be chosen.

The configuration space can then be $\mathcal{X} = SE(2) \times \mathcal{T}$, where each state, $s \in \mathcal{X}$, is represented by a vector of Cartesian coordinates x and y, orientation θ, and time t: $s = [x, y, \theta, t]$. The control space $\mathcal{U}$ is composed of controls in speed, ν, and steering angle, ϕ, such that $u \in \mathcal{U} = [\nu, \phi]$. The system dynamics are governed the common bicycle model, given by the following equations:

$$\dot{x} = \nu \cos\theta; \quad \dot{y} = \nu \sin\theta; \quad \dot{\theta} = \frac{\nu}{L} \tan\phi; \quad \dot{t} = 1 \tag{7}$$

where L is the vehicle wheelbase (distance between front and rear axle). Note that the turning radius ρ is related to the steering angle by $\tan(\phi) = L/\rho$, and the control set is then bounded by $\nu \in [0, \nu_{\max}]$ and $\phi \in [-\tan^{-1}(L/\rho_{\min}), \tan^{-1}(L/\rho_{\min})]$. The Dubins car model however adds the simplification that the steering rate is unbounded and the steering angle is to be one of only three possibilities corresponding to straight linear motion, full left turn, or full right turn ($\rho \in \{\infty, \rho_{\min}, -\rho_{\min}\} \therefore \dot{\theta} \in \{0, \nu/\rho_{\min}, -\nu/\rho_{\min}\}$), since the shortest possible distance path is proven to be composed of piecewise straight and circular arcs of this nature [16]. Also note that application of the Dubins path solver in the local steering of the RRT* algorithm for a car-like robot has been shown not to compromise asymptotic optimality [17].

While the boundary of the reachable space in x and y dimensions can be solved for analytically at any given time step t [13], the permissible range of orientations at a given x, y, and t is not easily characterized. Hence, a numerical approach will be explored.

For state propagation, the following four controls are applied in this work: $u \in U = \{[0, 0], [\nu_{\max}, \tan^{-1}(\frac{L}{\rho_{\min}})], [\nu_{\max}, 0], [\nu_{\max}, -\tan^{-1}(\frac{L}{\rho_{\min}})]\}$. These controls correspond with zero velocity, maximum speed left turn, maximum speed straight path, and maximum speed right turn, chosen as representative extremal values of the control set, $U \in \mathcal{U}$.

Starting iteratively from an initial state, all controls in set U are applied to each unexpanded state as a constant control over a fixed time step to propagate more subsequent states until a designated time horizon as a stopping point. In this case a constant speed and steering angle are applied, where subsequent state values can be found by leveraging the property that Dubins curves are exactly circular. Thus, knowing a fixed increment path length of $\nu_{t-1}\Delta t$ over time step Δt and by applying chord length and arc length formulas, we arrive at the following state propagation equations to follow radius ρ_{t-1} with speed ν_{t-1}:

$$\begin{aligned}
x_t &= x_{t-1} + 2\rho_{t-1} \sin\left(\frac{\nu_{t-1}\Delta t}{2\rho_{t-1}}\right) \cos\left(\theta_{t-1} + \frac{\nu_{t-1}\Delta t}{2\rho_{t-1}}\right) \\
y_t &= y_{t-1} + 2\rho_{t-1} \sin\left(\frac{\nu_{t-1}\Delta t}{2\rho_{t-1}}\right) \sin\left(\theta_{t-1} + \frac{\nu_{t-1}\Delta t}{\rho_{t-1}}\right) \\
\theta_t &= \theta_{t-1} + \frac{\nu_{t-1}\Delta t}{\rho_{t-1}} \\
t_t &= t_{t-1} + \Delta t
\end{aligned} \tag{8}$$

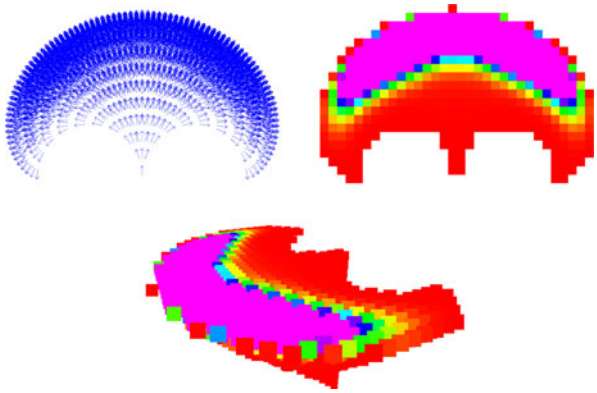

Fig. 2. The reachable space for a Dubins car robot with maximum speed constraint, represented as a reachable graph (top left), and a reachable map (top view in top right, orthographic view in bottom). The reachable graph is expanded first and will contain a relatively large number of states (here shown for 10 time steps). The reachable map is then processed to classify each hyperrectangle spanned by the graph states as reachable or not. The reachable map is plotted in 3-dimensions, x, y, and t (time), where discretization over orientation dimension θ is not explicitly shown due to visualization restrictions. The color spectrum correlates to the number of graph states spanning each hyperrectangle, where red indicates few graph states, and magenta indicates many states. Regions spanned by many graph states generally have wider ranges of valid orientations. Unreachable cells are not filled (transparent).

The ratio of reachable set volume to total bounded space volume for this system is dependent on the minimum turning radius, maximum speed and time horizon used, as well as the cell discretization resolution. For the finest level of discretization tested, with 20 time steps over a time horizon of 10 seconds, resolution in x and y of 0.5 m, θ resolution of 10 degrees, and robot with maximum speed of 1 m/s and minimum turning radius of 3 m, the reachable set volume is estimated to be only 4.03% of the bounded state volume. A 3D visualization of this reachable set is shown in Fig. 2, where the θ dimension is shown by the vector orientation in the reachable graph (top left), but is not shown explicitly in the reachable map since we're graphically limited to three dimensions. The color spectrum correlates to the number of states from the reachable graph which are associated with each reachable map location (magenta for higher value, red for lower), where a higher number of graph states typically indicates a greater range of valid orientations. Note that the original graph size was greater than 10^9 states (approx. 15 GB), but the reachable map was stored as a set of 483840 boolean values (approx. 30 MB).

The NN distance metric from Eq. (5) is used, where $dist(s_1, s_2) = dubDist(s_1, s_2) + w\Delta t$ with the $dubDist$ function solving for Dubins path length in the x-y plane (according to method in [16]), and w used as a weighting factor to optionally consider time duration as more or less influential to the path cost than path length (here $w = 1$).

V. Experiment Method

Both the analytical and numerical approach to RG (as described in Sections III and IV respectively) were applied in simulation over data collected from an autonomous golf cart platform operating in the National University of Singapore's Utown Plaza (Fig. 3). The primary sensors used were SICK LMS151 2D LIDARs, with moving obstacles' instantaneous pose and velocity detected through a supervised learning method of Support Vector Machine (SVM) applied over spatio-temporal features extracted from the scan data as detailed in [18]. These obstacles are then assumed to follow constant velocity trajectories and incorporated into an obstacle occupancy grid map, as in [5] but now layered by the additional time dimension. A prior feature map was built by applying the Simultaneous Localization and Mapping (SLAM) algorithm, where a synthetic LIDAR model is applied to reference vertical normal surfaces from a 3D rolling window of accumulated 2D laser scans to perform Adaptive Monte-Carlo Localizations. Details about the mapping and localization methods can be found in [19] and [20], while more specifics on the vehicle platform and various other subsystems are provided in [21].

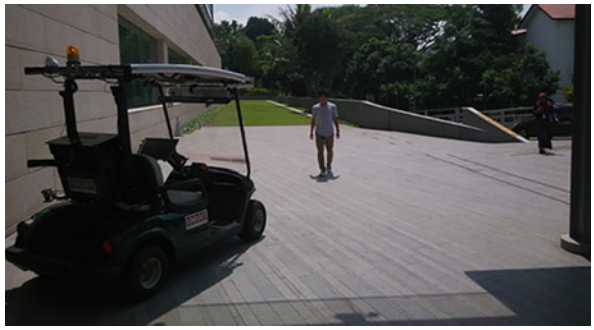

Fig. 3. The autonomous golfcart shown in the testing area on the National University of Singapore's Utown Plaza.

An RRT* [4] planner was utilized as implemented in the Open Motion Planning Library (OMPL)[22], where the planner's distance function was adjusted and new sampler classes were created according to the methods described in the previous sections. Robot Operating System (ROS) [23] was used as the underlying framework for interfacing with the autonomous vehicle. The code was executed on a computer with a 3.4 GHz processor and 16 GB RAM.

The golf cart was tasked to move forward and to the right while avoiding an oncoming pedestrian as per the scenario shown in Fig. 4. The pedestrian measured speed was 1.2 m/s vs the robot's 1 m/s limitation, and given the starting positions of the robot and pedestrian, the robot has only an approximately 2 second window of safe trajectories to pass in front of the pedestrian with at best less than 1 m clearance. The possible solution homotopy of passing behind the pedestrian is not seen at this time instance due to the planning time horizon.

The goal was specified as a position and orientation, where a state was deemed to satisfy the goal condition if it lies within a 0.2 m threshold Euclidean $\Re^2$ distance from the goal pose. The cost is taken simply as the sum of path length and time duration, with state rejection for clearance of less than 0.1 m, though in many situations obstacle clearance costs are advisable as well and can be implemented into a time varying cost map as in [24]. Four planner variations were tested for comparison over both the $\Re^2 \times \mathcal{T}$ and the $SE(2) \times \mathcal{T}$ models: (i) standard RRT* with no RG, (ii) RRT* with RG state sampling, (iii) RRT* with RG NN searching, (iv) RRT* with both RG state sampling and

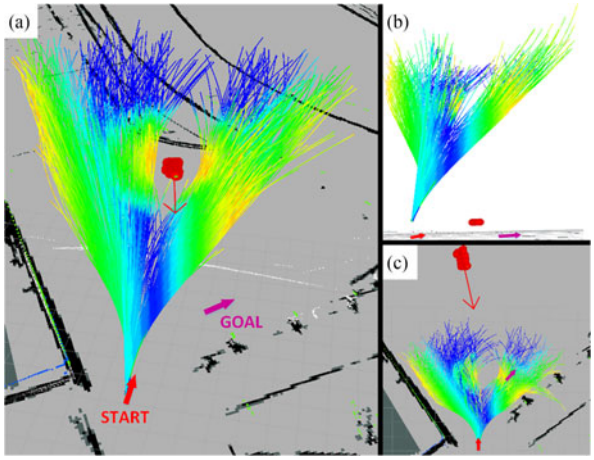

Fig. 4. The golfcart's mission: move from the red arrow pose labeled "START" to the purple arrow pose labeled "GOAL", while avoiding collision with incoming moving obstacle shown in red with velocity vector. Views a) orthographic, b) side, c) top. Black areas are walls, grey is drivable. Example path search tree shown for the Dubins model after 0.4 second planning time. Color of the path indicates obstacle clearance (continuous color spectrum with cool colors for greater clearance, warm colors for less clearance). Time dimension is normal to the map plane.

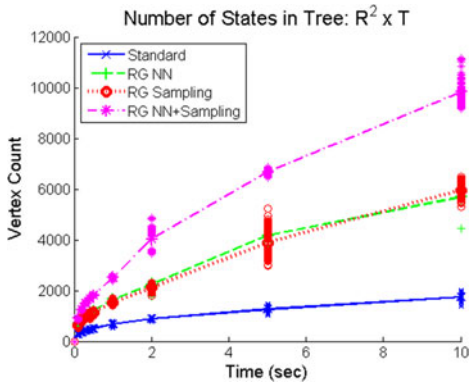

Fig. 5. Planner vertex counts for various levels of reachability guidance in the 2D holonomic robot model. Full distributions are plotted with lines extended between mean values.

RG NN searching. When RG is not applied, standard uniform sampling is performed and NN distance is calculated according to the model's standard distance function. For each of these planner variants, 100 simulated planning trials were performed with the number of tree vertices and success rate recorded at various planning durations (time to solve) ranging from 0.1 to 10 seconds.

Replanning was also executed at a frequency of 2 Hz, with a committed trajectory of 0.5 second duration executed between each replanning iteration.

VI. Results

A comparison of tree size given by vertex count for various levels of RG is shown for the $\Re^2 \times \mathcal{T}$ model (Fig. 5) and the $SE(2) \times \mathcal{T}$ model (Fig. 6). Incorporating either just RG NN searching or just RG sampling improves the tree expansion rate significantly, but still the fastest tree expansion rate is achieved

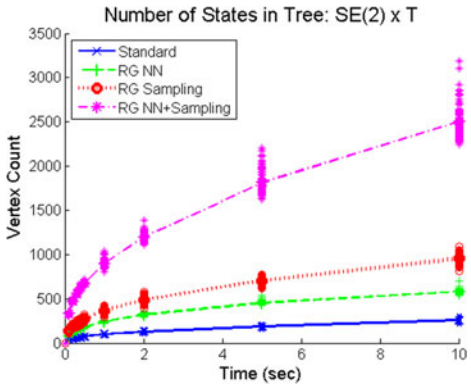

Fig. 6. Planner vertex counts for various levels of reachability guidance in the Dubins car robot model. Full distributions are plotted with lines extended between mean values.

TABLE I
Planning Success Rates (%): SE(2) × $\mathcal{T}$

Planning Time (s)	0.1	0.2	0.3	0.4	0.5	1.0	2.0	5.0	10
Standard	5	7	9	16	21	13	26	35	55
RG NN	30	39	44	55	53	55	73	65	81
RG Sampling	30	29	41	38	37	46	65	77	86
RG NN + Sampling	61	71	80	80	85	91	91	96	99

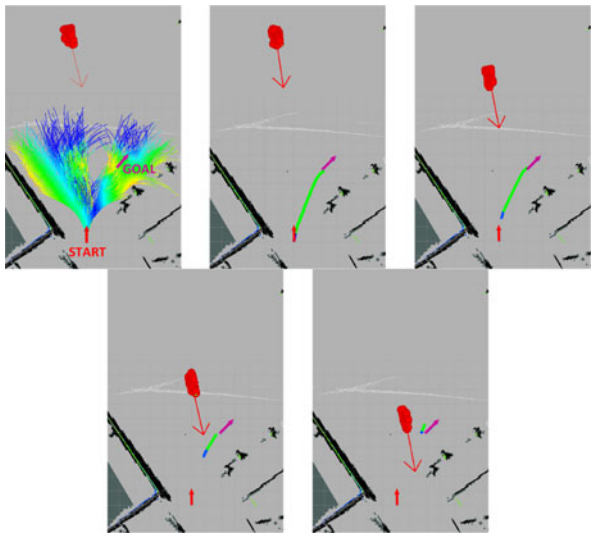

Fig. 7. Replanning snapshots with 2 Hz replanning frequency. Top left shows the initial path search tree. Top center is the initial best plan, with commit path shown in blue, and full solution path shown in green. The next snapshot is shown in the top right, followed by the bottom left and then the bottom right.

by utilizing both RG NN searching and RG sampling together. For the $\Re^2 \times \mathcal{T}$ model, the tree size is increased by about 3-4 times through RG in planning times under 1 second, and 5.6 times for a planning time of 10 seconds. Greater improvement rates are seen in the $SE(2) \times \mathcal{T}$ model, with the tree size increased by a factor ranging from 9.1–10.1 times for each of the various planning times by incorporating both RG sampling and NN searching.

Improvements are primarily due to not having considered unreachable states during the various planning stages (thereby avoiding wasted time). Note that the ratios of speed improvement are also loosely correlated with the inverse ratios of reachable space to total state space as calculated in Sections III-B and IV-B. Where the holonomic robot's reachable space was roughly one quarter that of the entire state space, speed is improved by approximately 4 times. While similarly we might expect a 25 times speed up for Dubins car robot, with reachable space at only 4% of total state space, this is unrealized likely due to the imperfect (over-approximated) representation of the reachable space, which results in still some wasted time considering a few unreachable states (albeit much less than without RG). The planner for the $\Re^2 \times \mathcal{T}$ model was already able to find a solution 99 to 100% of the time with or without RG in the test scenario. However, in the more complex $SE(2) \times \mathcal{T}$ model this was not the case, and success rates (Table I) were greatly improved with RG NN searching and RG sampling, where there appears to be strong correlation between success rate and planner tree size (Fig. 6).

With relatively high success rates possible even for planning times under 1 second, it's then feasible for a RG planner to be applied in online replanning applications. Fig. 7 shows several snapshots of the robot's motion and replanned path over the same test scenario as was used in the previous analysis. A new path search tree was generated every 0.5 seconds, with the first 0.5 seconds of each plan executed as a committed trajectory.

VII. Conclusion

The numerical approach to RG sampling-based motion planning detailed in this work was shown to be effective in improving planning speed (taken as rate of tree growth) by approximately an order of magnitude in the Dubins car robot model scenario, and likewise improved planning success rates dramatically. The RG planning concept was also demonstrated to improve planning speed in a simple 2D holonomic robot case under maximum speed constraint, where the reachable subspaces were shown as common geometric volumes and incorporated into state sampling and NN searching through analytical formulation. A generalized algorithm was presented for deriving a numerical representation of the reachable subspace, first by building a large offline state graph, then compressing the data representation into a smaller reachable map. The numerical approach can be used in situations when analytical formulations are difficult to derive or are otherwise inaccurate due to approximations.

Future works can extend numeric RG planning to dynamic models with acceleration constraints, and implement this into planner for online applications in various autonomous robot platforms. Obstacle clearance cost representations for predictive planning can also be further investigated.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[2] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *Int. J. Robot. Res.*, vol. 18, no. 11, pp. 1119–1128, 1999.

[3] J. H. Reif, *Complexity of the Generalized Mover's Problem*. Center Res. Comput. Technol., Aiken Comput. Lab., Univ, Harvard Univ., Cambridge, MA, USA, 1985.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[5] W. Liu *et al.*, "Autonomous vehicle planning system design under perception limitation in pedestrian environment," in *Proc. 7th Int. Conf. Cybern. Intell. Syst./IEEE Conf. Robot. Autom. Mechatronics*, 2015, pp. 159–166.

[6] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, 1986.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[8] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, 2002.

[9] S. Karaman and E. Frazzoli, "Sampling-based optimal motion planning for non-holonomic dynamical systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5041–5047.

[10] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5054–5061.

[11] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: The driftless case," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2368–2375.

[12] J. H. Jeon, S. Karaman, and E. Frazzoli, "Optimal sampling-based feedback motion trees among obstacles for controllable linear systems with linear constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 4195–4201.

[13] R. E. Allen, A. A. Clark, J. A. Starek, and M. Pavone, "A machine learning approach for real-time reachability analysis," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 2202–2208.

[14] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 2859–2865.

[15] C. Chen, M. Rickert, and A. Knoll, "Kinodynamic motion planning with space-time exploration guided heuristic search for car-like robots in dynamic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 2666–2671.

[16] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *Am. J. Math.*, vol. 79, no. 3, pp. 497–516, 1957.

[17] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proc. 49th IEEE Conf. Decis. Control*, Dec. 2010, pp. 7681–7687.

[18] B. Qin *et al.*, "A spatial-temporal approach for moving object recognition with 2D lidar," in *Proc. Int. Symp. Exp. Robot.*, 2014, pp. 807–820.

[19] Z. Chong, B. Qin, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, and D. Rus, "Mapping with synthetic 2D lidar in 3D urban environment," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 4715–4720.

[20] Z. Chong, B. Qin, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, and D. Rus, "Synthetic 2D lidar for precise vehicle localization in 3D urban environment," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 1554–1559.

[21] S. Pendleton *et al.*, "Autonomous golf cars for public trial of mobility-on-demand service," in *Proc. 2015 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 1164–1171.

[22] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[23] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, vol. 3, p. 5.

[24] S. Pendleton, X. Shen, and M. Ang Jr., "Moving obstacle avoidance via time-varying cost map," in *Proc. 2013 IFTOMM Int. Symp. Robot. Mechatronics.*, 2013, pp. 978–981.