OverR3T: Rapidly-exploring Random Reachable Set Tree for Safe Kinodynamic Planning of Nonlinear Systems by Over-approximating Reachable Sets

Yingxue Wang, Yifei Shao, and Hardik Parwana
Supervisor: Prof. Dmitry Berenson
University of Michigan

Abstract— **Sampling-based methods such as rapidly-exploring random tree (RRT), probabilistic roadmap method (PRM), and their variants have proven to be effective in planning in high-dimensional complex spaces but does not account for dynamic feasibility of the robot and safety. We proposed a reachability-based variant of RRT, OverR3T, to efficiently plan with the aid of close overapproximations of forward reachable sets for non-linear systems. Using overapproximating reachable sets, OverR3T can use reachable sets over longer time horizons for planning thus improving the planning speed. Using a combination of offline precomputation and online 'slicing', in addition to zonotope-based collision check, we further reduce the online planning time. The performance of the proposed method along with other state-of-the-art reachability-guided sampling algorithms is tested in a simulated pendulum swing-up problem with kinodynamic constraints and obstacles, and is shown to outperform R3T in obtacle-free environment.**

I. Introduction

Sampling based methods have been useful in complex and higher dimensional spaces as it does not require explicit discretization of planning region into cells. Among these, the most popular ones include RRT, PRMs and their variants. However, most of these variants do not account for dynamics of the robot and fail to provide safety guarantees for a trajectory to goal in practice. New work in kinodynamic motion planning solves this problem of dynamic feasibility. Some use an existing library of connectable trajectories [1] for planning, some of the work lift the sampling based planning from configuration space into state space [2], and others try to sample in a way that ensures that the new nodes added to the tree are easily reachable from the parent node [3].

The above approaches, however, usually come with some tradeoffs: They use discretization (either in state space or control input space) to solve the problem, which can prohibit finding a solution when very fine actuation is required, or become computationally intractable for high dimensional spaces. A newly proposed method, R3T[4], uses reachability to aid planning, and does not suffer from these side effects. Furthermore, with some smart uses of reachable set representations, the computational burden of finding nearest neighbor to the reachable set can be greatly alleviated.

The idea of using reachability for planning is powerful. An ideal reachability based RRT would have, for some finite time horizon, an exact reachable set of each node in the tree and when a new point is sampled in state/configuration space, it would project the sample to the nearest reachable set. This projected point would then be added to the tree as new node ensuring that all paths between nodes are dynamically feasible. However, getting exact reachable sets is very expensive in general, even for offline analysis, and approximations based on finite number of points[3], which they call keypoints, or based on linearizations[4] are used instead. Moreover, since the projected point is no longer guarantanteed to lie in *exact* reachable set due to approximations, an additional step to use nonlinear dynamics to reach near the projected point is required and it is the result of this nonlinear state propagation that is added as a new node to tree.

However, critical shortcomings of these are that keypoints themselves are very sparse and that both keypoints and linearized under-approximating reachable sets are known to be inaccurate representations over large time horizons. This can cause slow performance in large state space problem and the new node to lie far away from any reachable sets in the tree. In addition, all the above methods use a constant RRT time step for constructing reachable set boundary, and hence constant time step for paths between any two nodes, which would not be ideal if we want to take intricate paths, for example small turns, around obstacles. Finally, the attempt to reach near the projected point would require control design which is also not trivial.

In this project, we propose to use overapproximating zonotope reachable sets instead of linearized reachable sets. Firstly, these reachable sets are shown to be more accurate over longer time horizon, which improves the planning efficiency. Secondly, these over-approximating reachable sets also enable efficient collision check, which is where most planning algorithms spend the majority of the computation time. Thirdly, the zonotope reachable set, being computed for time intervals, can easily accommodate adaptive stepsize (upto a maximum user provided RRT time horizon), which improves the result like the RRT-Connect variant of RRT. Finally, the overapproximating reachable sets are computed with control parameterization of nonlinear system which trivializes control design during nonlinear propagation step after projection. Using these four advantages and a combined offline precomputation and online planning architecture, we improve upon the state of the art method on speed. The novel contributions can be summarized as follows:

- Using zonotope represented overapproximating reachable sets to enable large RRT step sizes.
- Efficient collision check for obstacles in form of AH-Polytopes.
- Adaptive RRT step size where multiple candidate nodes are created with one random sample, up to a user provided max time step.
- Control design based on parameterization.

The paper is structured as follows. Section IV gives an brief overview of related work of kinodynamic planning. Section III then defines the problem, followed by solution methodology in Section IV, discussion of results in Section V and finally conclusion in Section VI.

II. Related Work

Using reachability for motion planning has been proposed before. Some methods propose solving the exact reachability problem by solving a two point boundary value problem(TPBVP) offline for dense points in state space. Then these points can be used online for motion planning. The points not included in this dataset are accounted for at run-time by either interpolation or fitting a function approximator like a SVM[5]. These methods are accurate up to the interpolation or approximation errors, but for highly nonlinear systems, they can have significant delays in finding a parent node as two-point query based procedures cannot provide efficient representation of whole reachable set.

Several methods have been proposed that do not perform exact reachability analysis. Two state-of-the-art methods that motivated our project are Reachability Guided RRT (RG-RRT)[3] and Rapidly-Exploring Random Reachable Set Tree(R3T)[4]. RG-RRT simulates trajectories for multiple random control inputs and stores the end points of these trajectories as keypoints representing the reachable set. A future random sample is then mapped to the nearest keypoint which is then added as the new node in the tree. Using a similar scheme, R3T linearizes the system at the node state and calculates the reachable set based on this linearized system. These reachable sets can be represented efficiently in terms of AH-polytopes. This representation enables efficient projection to the closest AH-polytope when the algorithm samples a new point in free space.

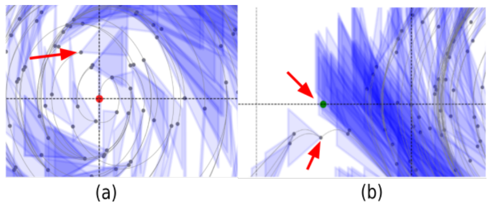

Fig. 1. R3T Issues: black dots represent nodes, blue triangles represent linearized reachable set, red is initial state, green is goal state. (a)many nodes, for example one marked with red arrow, lie far outside the reachable set of their parent node. (b) goal lies inside linearized reachable set but is unable to be reached due to nonlinear dynamics.

While the above two methods perform well, they have several shortcomings. First, linearized system usually leads to underapproximation of reachable sets that give good performance only for small time steps and fail otherwise. Fig1(a) shows a tree with nodes and their reachable sets constructed with R3T procedure. When a new sampled point is projected to reachable set and then attempted to reach using nonlinear dynamics, the final point ends up lying far away from projected point as well as the linearized reachable set (note again that a new node is the one obtained by nonlinear simulation and not the projected point). Second, even when goal state lies well within the linearized reachable sets, it might still be very difficult to reach there. Fig1(b) shows a scenario where all trajectories from nodes near the goal state curve away because of non-linearity. Finally, these methods do not also propose any efficient method to check collisions. R3T simply checks for collision on discretized points on the nonlinear path to a child node.

Our proposed approach that uses overapproximating reachable sets is made possible with zonotope representation. Zonotope based reachability analysis was first introduced in [6]. It was then extended to non-linear systems in [7] by finding overapproximating the linearization error in a admissible way. This overapproximating set was applied to a receding horizon motion planning frame to enable real-time safe planning in [8]. Their application in sampling based approaches have not been investigated yet and we adapt them here to overcome previously mentioned limitations.

III. Problem Statement

Consider the following continuous-time nonlinear system:

$$\dot{x} = f(x, u) \tag{1}$$

where $x \in X \subset \mathbb{R}^n$ is the continuous system state, $u \in U \subset \mathbb{R}^m$ is the control input.

Problem 1. *Given (1), an initial state x_0 and a goal state x_G, we aim to find a trajectory $\zeta : [0, T] \longrightarrow (x, u)$ that satisfy 1, $x(0) = x_0, x(T) = x_G, x(t) \in X, u(t) \in U, \forall t \in [0, T]$. Furthermore, if the state is constrained to lie in a set X_{free}, then the above solution should satisfy $x(t) \in X_{free}$.*

The continuous time dynamics can be forwarded propagated in time with standard schemes like Euler integration or RK4 given a maximum time horizon and intermediate time update step size.

IV. Method

In this section, we first first introduce zonotopes and how they can be used to represent reachable sets. Next, we explain how we parameterize a controller for offline reachable set computation. Then, we introduce slicing, collision check, and keypoint generation operations for zonotope based reachable sets. Finally, we propose OverR3T algorithm for planning in state space, which takes into account of everything introduced in this section online.

A. Zonotope Representation

In order to represent uncertainty in a dynamical system, many polytope representations have been introduced such as AH-polytope, H-polytope, and hyperboxes. Among these, zonotopes stand out as efficient for linear system propogation. Thus, we represent the location and uncertainty of our overapproximating reachable set with zonotopes.

Definition 1. *A zonotope is a set of the form*

$$\mathcal{Z} = \{p \in \mathbb{R}^n \mid p = c + \sum_{i=1}^{\ell} \beta_i \cdot g^{(i)}, -1 \leq \beta_i \leq 1\} \quad (2)$$

with center $c \in \mathbb{R}^n$ and m generators $g^{(1)}, \cdots, g^{(\ell)} \in \mathbb{R}^n$. Denote $\mathcal{Z} = \langle c, g \rangle$ where $g := [g^{(1)}, g^{(2)}, \cdots, g^{(\ell)}]$. Such an object can be easily stored in memory by representing $\mathcal{Z}$ as a single matrix, in which the first column represents the center and other columns represent generators.

B. Offline Reachable Set Computation

Using [7] and the associated toolbox [9], we can compute the overapproximating zonotope representation of the forward reachable set of (1) for a range of initial conditions under a range of control inputs. This method linearizes the system and over-approximates the linearization error in an admissible way. We refer to zonotope parameters k as the tuple consisting of initial state of a system k_I and the control parameters k_u. In order to perform "slicing"(introduced in IV-C.1), we require a constant zonotope parameter value throughout $t = [0, T]$. Therefore, instead of inputting control input directly, we design a controller of the form

$$u(t) = g(x, k_u, t) \quad (3)$$

where g denotes the parameterized controller and k_u denotes the control parameter value. Note that k_u is constant for $t = [0, T]$, where T is the user specified maximum time-horizon of a RRT extension.

Now we calculate the forward reachable set for the control parameterized system using [9], and results in

$$\begin{aligned} \Phi_t(I) =& \{y \in \mathbb{R}^n : \exists x \text{ solution of (1) using (3)}, \\ & x(0) \in I \wedge x(t) = y\} \quad (4) \\ \mathcal{R}_\tau(I) =& \bigcup_{t \in [(\tau-1)\Delta t, \tau \Delta t]} \Phi_t(I) \quad (5) \end{aligned}$$

where τ is a positive integer time index of a forward reachable set, I is a set of the initial conditions, Δt is a arbitrary chosen timestep of a reachable set, and $\mathcal{R}_\tau$ is a zonotope.

Since each $\mathcal{R}_\tau$ is an over-approximating forward reachable set, we have

$$\mathcal{R}_{0:T} = \bigcup \mathcal{R}_\tau, \forall \tau \in 1, 2, \cdots, T/\Delta t, \quad (6)$$

where $\mathcal{R}_{0:T}$ overapproximates the whole trajectory from $t = [0, T]$.

Both Δt and T can be arbitrarily chosen, but as their size increase, the forward reachable set becomes less accurate. However, choosing too small of a Δt will require significant storage space and too small of a T can take a long time in planning.

Example 1. *Suppose the user given maximum RRT time step is $T = 0.3s$, and we select $\Delta t = 0.01s$, splitting the reachable sets it into 30 intervals, each a zonotope. The green series of zonotopes in Fig. 2 shows a reachable set that was computed for a pendulum system.*

C. Zonotope Operations

Zonotopes are efficient for storage, but it remains unclear for 1) how to obtain accurate reachable set that doesn't consider all initial conditions and all possible control inputs, 2) how to efficiently check collision with obstacles, and 3) how to find meaningful keypoints in a zonotope that represent the final state of a system. In this section, we aim to introduce the following operations to answer these questions

1) Slicing Operations: Intuitively, a zonotope that represent the reachable set of all parameter values and an interval of initial conditions is going to be very conservative. Therefore, we exploit the constant parameter value we constructed earlier and a slicing operation to obtain more accurate reachable set online given some information of the system at hand. We say that, given a zonotope for an interval of initial conditions and range of control parameter values, we 'slice' the zonotope (each of 30 zonotopes in Example 1) to again get an overapproximated zonotope reachable set for the specific initial state and/or control parameter for which it was sliced.

Definition 2. *Given a zonotope $\mathcal{Z} = \langle c, g \rangle \subset \mathbb{R}^n$, its i-th dimension is* sliceable *if there is only one generator $g^{(j)}$ of $\mathcal{Z}$ that has nontrivial value in its i-th dimension for some j, i.e. there's only one nonzero element in the i-th row of g. Such generator $g^{(j)}$ is called the* sliceable generator *corresponding to the i-th dimension.*

A sliceable dimension will have zero dynamics, which means the i-th dimension of $g^{(j)}$ remains constant while other non-sliceable dimensions change.

Now we can slice the i-th dimension of $\mathcal{Z}$ with a particular value $z_i \in [c_i - g_i^{(j)}, c_i + g_i^{(j)}]$ by a slicing operator $\mathfrak{s}_i : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$:

$$\mathfrak{s}_i(z_i, \langle c, g \rangle) = \langle c + (z_i - c_i)/g_i^{(j)} \cdot g^{(j)}, g \setminus g^{(j)} \rangle \quad (7)$$

obtaining a zonotope with a new center and one fewer generator.

After slicing, the zonotope remains a overapproxiamting reachable set for the parameter value and/or initial state.

Example 2. *We construct reachable set of the pendulum system for range of initial states $k_I = [\theta, \dot{\theta}]$ and for control parameter k_u. Together, they form the zonotope parameter $k = [k_I, k_u]$.*

In Fig. 2, we visualize the unsliced FRS, sliced FRS wrt initial condition and sliced FRS wrt initial condition and control parameter.

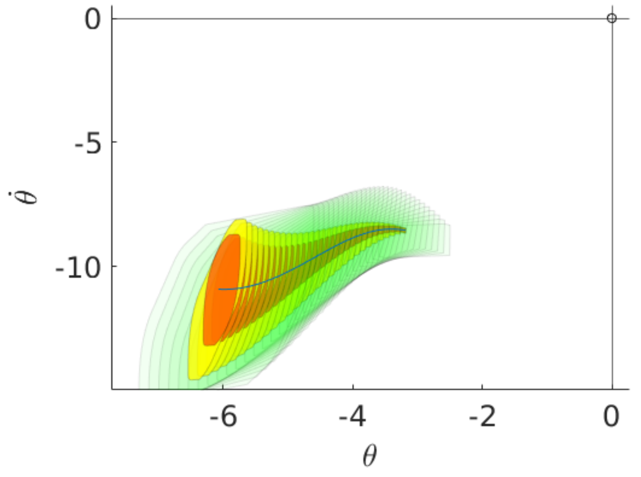

Fig. 2. Green zonotopes are a overapproximating reachable set computed for the pendulum system, for initial condition set $I = \{(\theta, \dot{\theta})|\theta = [-3.5, -2.5], \dot{\theta} = [-10.5, -9.5]\}$, and control parameter $k_u = [-0.5, 0.5]$, using parameterization that torque $u = 2k_u$Nm, for $\Delta t = 0.01s$ and $T = 0.3s$. Yellow zonotopes are sliced at initial condition parameters $k_I = (\theta, \dot{\theta}) = (-2.65, -8.56)$, and red zonotopes are sliced at the same initial condition parameters and control parameter $k_u = -0.46$. The blue line is a `ode45` simulation of the nonlinear dynamics at the same parameters.

2) Collision Check Using Zonotopes: To guarantee collision free trajectories, over-approximating reachable sets of the system should not collide with any obstacles during online planning. If we represent obstacles as zonotopes, we need to be able to perform collision checking between two zonotopes efficiently. This is done by converting the collision checking problem into a problem of checking whether a point belongs to an H-polytope as the following.

Given two zonotopes $\mathcal{Z} = \langle c, g\rangle$ and $\tilde{\mathcal{Z}} = \langle \tilde{c}, \tilde{g}\rangle$. Define zonotope $\hat{\mathcal{Z}} := \mathcal{Z} \oplus \langle 0, \tilde{g}\rangle$ where $\oplus$ denotes for Minkowski sum. Since zonotope is just the special case of H-polytope, we can convert $\hat{\mathcal{Z}}$ into an H-polytope $\mathcal{H}$ as in [10, Thm. 7]. Then checking whether $\mathcal{Z}$ and $\tilde{\mathcal{Z}}$ collides is equivalent as checking if $\tilde{c}$ satisfies the linear constraints imposed by $\mathcal{H}$ [11, Thm. 15].

We also tried to use an optimization based method for collision checking proposed [12, Prop. 2], but it is 5 times slower in practice, so it is not used. Furthermore, to improve efficiency, we reverse the order of collision check by checking the last zonotope first, since the last zonotope is the most conservative estimate and therefore the largest.

3) Finding keypoints of $\mathcal{R}_\tau$: Like previous algorithms, we would like to project the randomly sampled point to the closest reachable set. Finding projection to zonotopes is an optimization problem that is not suited for frequent use in online algorithms. Therefore, we associate some keypoints, representative of reachable sets, to speed up the closest reachable set search in initial phase (described in more detail in Section IV-E).

When a node is added to the tree, its reachable set is obtained by slicing the original reachable set w.r.t initial state of node k_I and stored in the tree. Now this reachable set, for each time interval, is further sliced at different control parameter values k_u and the centers of resulting zonotopes are taken as keypoints. Note the keypoints doesn't have to be selected with fixed spacing, since slicing can be done on any value of a parameter.

D. Online Algorithm

The overall algorithm is outlined in Algorithm 1. It starts with standard RRT steps of initializing a tree with initial state as the root node. However, each node stores not only the state but also its reachable set sliced at k_I(the state of the node) and keypoints. A new point is sampled in state space(*Line 4*) and the closest reachable set in tree and its corresponding closest keypoint is found(*Line 5*). The reachable set is first sliced w.r.t this keypoint parameter and checked for collision. If found safe, then the keypoint is mapped to the control input and the nonlinear simulation is done to arrive at a new location x_{new} in RRT time step T(*Line 5*,Algorithm 2). Since the reachable set used for collision checking is based on overapproximation, the path to x_{new} obtained using actual nonlinear dynamics, is guaranteed to lie inside the reachable set and hence is collision free. x_{new}, along with its newly computed reachable set, is added as the new node to the tree(*Line 8*). Every time a new node is added to the tree, it is checked whether goal is within the reachable set of this node or not(*Line 11*). If it is, then the closest keypoint in the reachable set is found(*Line 12*), mapped to corresponding control input, and forward simulated in time. If the new state is within the tolerance limit of goal state, it is said to be reached and the tree is returned(*Line 14,15*). Otherwise, the algorithm continues.

E. FindClosest

A naive way to find closest reachable set is to implement KDTree search algorithm on all keypoints and choose the reachable set of closest keypoint as the closest reachable set. However, this might not be efficient and instead we use same algorithms as R3T[4][13] to find closest keypoint in which they first use KDTree and some simple checks to find candidate closest keypoints and then chose the closest among them by directly using axis aligned box approximation of zonotopes instead of keypoints. Using this method, once we find closest reachable set, we find corresponding closest keypoint and use it to map to control input as explained in previous section. Note that number of keypoints(and hence number of control parameterizations) are limited for efficiency. Therefore, finding closest zonotope and not just the closest keypoint also has the advantage that interpolation/function approximation techniques can be used to find better keypoints than those used in dataset to generate reachable set and will be explored in future work.

V. Experiment & Result

A. Problem setup, parameterization

We test our system for the pendulum swing up problem for following dynamics:

$$J\ddot{\theta} = -Mgl\sin\theta + u - b\theta \qquad (8)$$

Algorithm 1 OverR3T

```
Require: $x_0$, $x_g$, $\mathcal{S}$, $\epsilon$ ▷ root node, goal location, sampling function, tolerance for finding goal
1: $Tree$ = new $Tree()$                                   ▷ search tree
2: $Tree$.add($x_0$)
3: while time < max time do     ▷ not in reachable sets of any node in the tree
4:     $x_s \leftarrow S()$
5:     $k_u, \mathcal{R}_\tau(x_c) \leftarrow$ FindClosest($Tree$,$x_s$) ▷ returns closest keypoint's control parameter and its corresponding zonotope in a reachable set
6:     $x_{\text{new}} \leftarrow$ Extend($x_c, k_u$)
7:     if $x_{\text{new}} \neq \emptyset$ then
8:         $Tree$.add($x_{\text{new}}$)
9:         Rewire($Tree$, $x_{\text{new}}$)                ▷ Optional
10:    end if
11:    if $x_g \in \mathcal{R}_{0:T}(x_{\text{new}})$ then
12:        $k_u \leftarrow$ FindClosestKeypoint($\mathcal{R}_{0:T}(x_{new}), x_g$) ▷ returns closest keypoint's control parameter to $x_g$ in $\mathcal{R}_{0:T}(x_{new})$
13:        $g \leftarrow$ Extend($x_{\text{new}}, k_u$)
14:        if $g \neq \emptyset \wedge$ Dist($g, x_g$) $< \epsilon$ then
15:            $Tree$.add($g$) return $Tree$
16:        end if
17:    end if
18: end while
```

Algorithm 2 Extend

```
Require: $x_c, k_u, U, T, \Delta t$, obstacles ▷ closest state, keypoint's control parameter, keypoint to control-input map, RRT time step, simulation time step
1: $\mathcal{R}_{\text{0:T}} \leftarrow$ getReachSet($x_c$)
2: if collision_check($\mathfrak{s}(k_u, \mathcal{R}_{\text{0:T}})$, obstacles) then    ▷ $\mathfrak{s}$ is slicing w.r.t $k_u$
3:     return $\emptyset$                        ▷ Collides with obstacle
4: else
5:     for $t = 1 : \Delta t : T$ do      ▷ Do nonlinear simulation
6:         $u \leftarrow U(x, k_u)$
7:         $x \leftarrow x + f(x, u)\Delta t$
8:     end for
9: end if
10: return $x$
```

where M is mass of ball at end of massless rod, l is the length of rod, $J = Ml^2$ is moment of inertia, b is damping constant, u is the control input torque. The values used for simulations are: $M = 1kg, l = 0.5m, b = 0.1m^2kg/s$. The goal state is the upright orientation which corresponds to $\theta = \pm\pi, \dot{\theta} = 0$.

The control input u is parameterized and we tried following three forms: $u = K(\theta - k_u), K(\dot{\theta} - k_u), Kk_u$, where K is a constant. The reachable sets were computed offline for a range of control parameters and a range of initial conditions. The keypoints then correspond to zonotope centers that are obtained by slicing at different parameter values in this range. We found the best performing controller to be $u = Kk_u$ and the discussion henceforth will be based on that. Other parameterizations were found to be not exploring the state space efficiently. For simulations, we used $K = 2$ and $0.5 \leq k_u \leq 0.5$ so that the control input is limited to $|u| \leq 1$ Nm.

B. Offline Computation

We performed offline computation for initial state $I = \{(\theta, \dot{\theta}) | \theta = [-3.5, 3.5], \dot{\theta} = [-9.5, 9.5]\}$, and control parameter $k_u = [-0.5, 0.5]$. In order to control conservatism, we further reduce the initial condition interval into $I_{ij} = \{(\theta, \dot{\theta}) | \theta = [-0.5 + i, 0.5 + i], \dot{\theta} = [-0.5 + j, 0.5 + j]\}$, for $i = -3, -2 \ldots, 3, j = -9, -8 \ldots, 9$, for a total of 133 reachable sets that take 1.7MB of space.

C. Result

Fig.3 illustrates how a new node is added to the tree for first time when only root node is present. When a new point is sampled, the closest reachable set among all reachable sets for different intervals is found first using *FindClosest*. Then, the controller corresponding to closest keypoint is used to get to the green point which is added as a new node to the tree. We now provide comparison results with RG-RRT and R3T.

1) Obstacle free region: In order to compare our algorithm to others, we use only the last time interval zonotope (index 30 from Example 1) to do closest reachable set search in absence of obstacles. We call this *fixed-time step* run which is unlike the scenario in Fig.3 where all time interval zonotopes are used for finding closest reachable set and which we will call *adaptive-time step* run from hereon. Fig.5 and Fig.6 show the results for R3T and the proposed OverR3T algorithms with control input limit of $|u| \leq$1 Nm. The circular trajectory taken by state represents the pendulum swinging to both sides and slowly increasing the amplitude to finally reaching the upright location.

2) Obstacles: We add a rectangular obstacle in $(\theta, \dot{\theta})$ space to test collision avoidance. R3T ensures collision safety by naively checking for collision at discretized points when making a path to new node. For OverR3T, we first tried running it with fixed-time step(0.3s) as in the case without obstacles. However, we note that our method is affected adversely in the presence of obstacles since the overapproximated reachable set, being large, often detects collision and is unable to find a valid path. This was found to be due to long paths related with the fixed-time step. Therefore, we use adaptive-time step so that algorithm is free to make a path corresponding to any time interval(0-0.3s and not exactly 0.3 s) and is thus able to perform smaller maneuvers, and checking collision using fewer and less conservative zonotopes. We would like to emphasize that although in this report the advantage of adaptive time steps is realized only for our proposed algorithm in the obstacle case but in more complicated state spaces, other algorithms too, if they can be extended to have adaptive step sizes, are also sure to benefit from it.

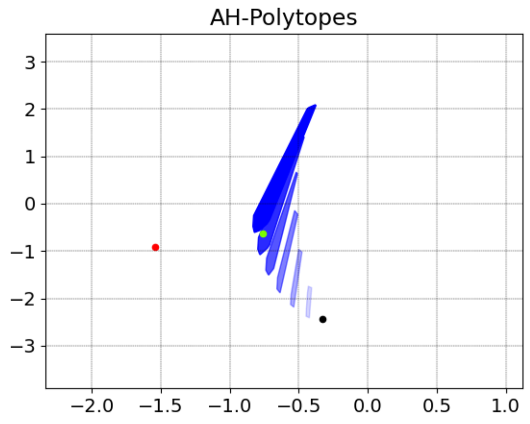

Fig. 3. We illustrate how to add new node in OverR3T: we start with the black root node, and sample red point in state space, green is the point obtained by nonlinear simulation and is added to tree as new node. Blue blocks are reachable sets of black node corresponding to equally spaced time intervals with boundaries at $(1/6, 2/6, 3/6, 4/6, 5/6, 1) \cdot 0.3$ seconds. The second last reachable set is found to be the nearest one and so new node lies within it.

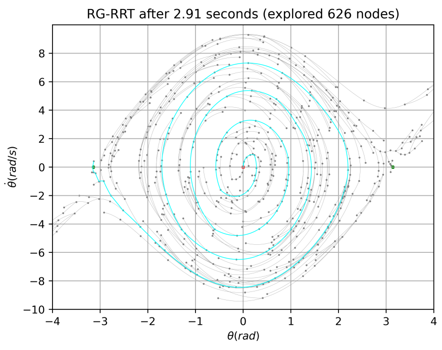

Fig. 4. RG-RRT run: Red dot is initial state, green dots represent goal state $\theta = \pm\pi, \dot{\theta} = 0$. The found path is shown in cyan.

D. Performance Tradeoffs

Table I shows the statistics recorded over 10 trials. RG-RRT peforms very fast for this pendulum example but it is known that it doesn't perform well for more complex spaces and for larger time steps[4]. The simulation here serves to show that in general it requires much more nodes than other algorithms.

It can also be seen that R3T becomes faster with larger time step but at the same time more nodes begin to lie far away from reachable sets(see Fig.1) and therefore would scale poorly with more complex spaces. On the other hand, OverR3T is faster and also requires much fewer nodes to reach the goal.

Finally, for obstacle case R3T seems to perform better in time compared to OverR3T which we believe is because the naive obstacle check done in R3T is very fast for the case study at hand. Evaluating performance for obstacles will be done for more applications in future. The table also shows the difference between OverR3T performance with fixed-time

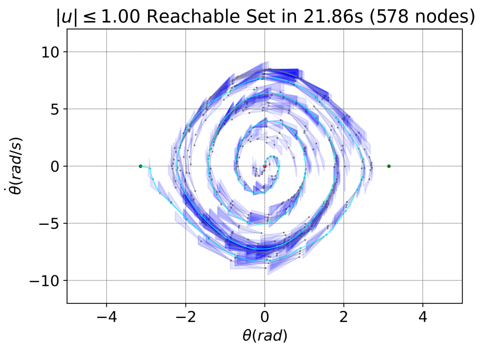

Fig. 5. R3T run: Red dot is initial state, green dots represent goal state $\theta = \pm\pi, \dot{\theta} = 0$. Dark Blue blocks represent reachable sets and the found path is shown in cyan.

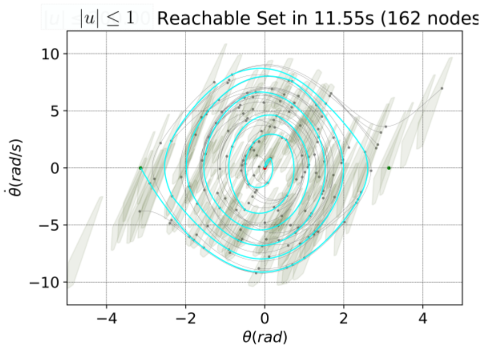

Fig. 6. OverR3T run: Red dot represents initial state, green sots goal state. Gray blocks represent the reachable set corresponding to last time index and the found path is shown in cyan.

and adaptive-time step. It was found that the planner can explore very fast and expand tree quickly towards the goal node with fixed-time step(Fig.6) but requires adaptive-step to make smaller and finer paths(Fig. 8).

TABLE I
PATH PLANNING PERFORMANCE FOR DIFFERENT PLANNERS

	T (s)	Path Cost	Time (s)	N. of Nodes
RG-RRT	0.1	84.64	5.04	1192
R3T	0.1	76.19	73.99	980
R3T	0.3	78.15	26.86	636
R3T + Obstacle	0.3	78.09	65.0	836
OverR3T	0.3	97.26	20.6	208.5
OverR3T + Obstacle (fixed-step)	0.3	88.07	124.18	311
OverR3T + Obstacle(adaptive-step)	0.3	81.47	84.33	151.7

VI. Conclusion and Future Work

In this work, we proposed using zonotope reachable sets in a RRT styled planning algorithm to aid planning. We show that this algorithm performs better than the state of the art reachability based planning, R3T, when no obstacle is present and perform worse than R3T when there are obstacles. We would like to test our algorithm with more case studies, for example, on dubins car with obstacle. Furthermore, we would like to explore theoretical guarantees that overapproximated

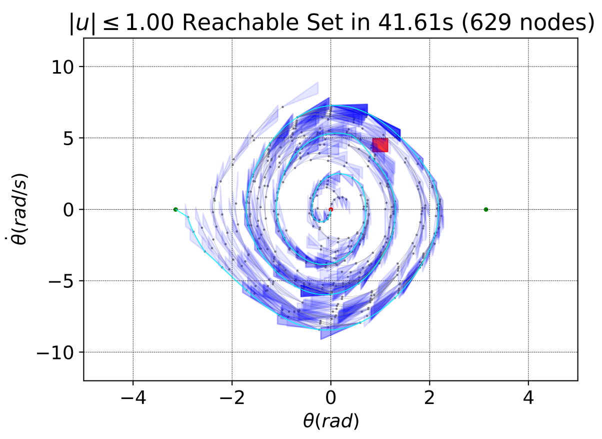

Fig. 7. R3T with obstacle: Red box is the obstacle.

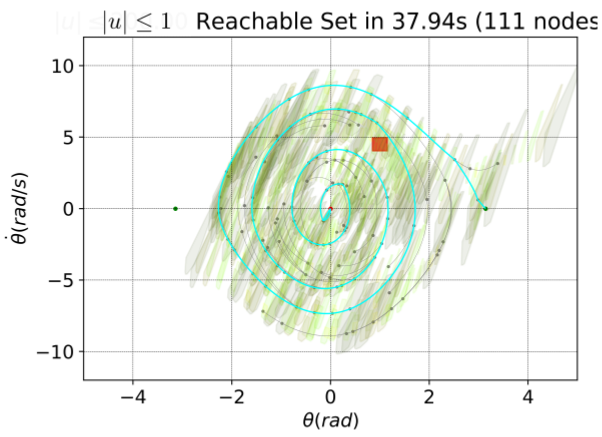

Fig. 8. OverR3T with obstacle: Red box is the obstacle. The color gradations for blocks represents reachable sets corresponding to different time intervals(see Fig.3). Gray is for last time interval whereas light green for for first time interval.

reachable sets can provide. Also, as discussed in performance trade-off, working with fixed-time step is better for rapid tree expansion and working with adaptive-time step is better for making smaller paths for more intricate motion. We would like to explore optimal combination of them so that the algorithms adapts between them based on whether tree is able to expand or not. Furthermore, our method to do collision check might be better suited to be used for complex shaped obstacles since zonotopes are generalization of polytopes and might not have shown their usefulness in current implementation of rectangular obstacles. So, we will test it for more obstacles in future. Finally, we can explore more options for control parameterization to find better convergence properties.

References

[1] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[2] Dustin J Webb and Jur Van Den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061. IEEE, 2013.

[3] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *2009 IEEE International Conference on Robotics and Automation*, pages 2859–2865, 2009.

[4] Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3t: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4245–4251, 2020.

[5] Ross Allen and Marco Pavone. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In *AIAA Guidance, Navigation, and Control Conference*, page 1374, 2016.

[6] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 291–305, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[7] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048, 2008.

[8] *Safe, Aggressive Quadrotor Flight via Reachability-Based Trajectory Design*, volume Volume 3, Rapid Fire Interactive Presentations: Advances in Control Systems; Advances in Robotics and Mechatronics; Automotive and Transportation Systems; Motion Planning and Trajectory Tracking; Soft Mechatronic Actuators and Sensors; Unmanned Ground and Aerial Vehicles of *Dynamic Systems and Control Conference*, 10 2019. V003T19A010.

[9] Matthias Althoff. An introduction to cora 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[10] Matthias Althoff, Olaf Stursberg, and Martin Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4:233–249, 05 2010.

[11] Shreyas Kousik, Patrick Holmes, and Ram Vasudevan. Safe, aggressive quadrotor flight via reachability-based trajectory design. In *ASME 2019 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, 2019.

[12] Joseph K Scott, Davide M Raimondo, Giuseppe Roberto Marseglia, and Richard D Braatz. Constrained zonotopes: A new tool for set-based estimation and fault detection. *Automatica*, 69:126–136, 2016.

[13] Albert Wu, Sadra Sadraddini, and Russ Tedrake. The nearest polytope problem: Algorithms and application to controlling hybrid systems. In *2020 American Control Conference (ACC)*, pages 1815–1822. IEEE, 2020.