MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Motion Planning with Invariant Set Trees

Weiss, A.; Danielson, C.; Berntorp, K.; Kolmanovsky, I.V.; Di Cairano, S.

TR2017-128 August 2017

Abstract

This paper introduces the planning algorithm SAFERRT, which extends the rapidly-exploring random tree (RRT) algorithm by using feedback control and positively invariant sets to guarantee collision-free closed-loop path tracking. The SAFERRT algorithm steers the output of a system from a feasible initial value to a desired goal, while satisfying input constraints and non-convex output constraints. The algorithm constructs a tree of local state-feedback controllers, each with a randomly sampled reference equilibrium and corresponding positively invariant set. The positively invariant sets indicate when it is possible to safely transition from one local controller to another without violating constraints. The tree is expanded from the desired goal until it contains the initial condition, at which point traversing the tree yields a dynamically feasible and safe closed-loop trajectory. We demonstrate SAFERRT on a spacecraft rendezvous example.

IEEE Conference on Control Technology and Applications

Motion Planning with Invariant Set Trees

Avishai Weiss, Claus Danielson, Karl Berntorp, Ilya Kolmanovsky, and Stefano Di Cairano

***Abstract*— This paper introduces the planning algorithm** SAFERRT**, which extends the rapidly-exploring random tree** (RRT) **algorithm by using feedback control and positively invariant sets to guarantee collision-free closed-loop path tracking. The** SAFERRT **algorithm steers the output of a system from a feasible initial value to a desired goal, while satisfying input constraints and non-convex output constraints. The algorithm constructs a tree of local state-feedback controllers, each with a randomly sampled reference equilibrium and corresponding positively invariant set. The positively invariant sets indicate when it is possible to safely transition from one local controller to another without violating constraints. The tree is expanded from the desired goal until it contains the initial condition, at which point traversing the tree yields a dynamically feasible and safe closed-loop trajectory. We demonstrate** SAFERRT **on a spacecraft rendezvous example.**

I. INTRODUCTION

Path planners generate a collision-free path between an initial state and a goal state in a generally non-convex environment. The presence of obstacles in the environment, and hence the non-convexity, renders this problem PSPACE-hard [1]. Fortunately, many heuristic techniques have been developed to simplify the problem. Recently, sampling-based planning methods, such as rapidly-exploring random trees (RRT), have become popular. In these methods, the search space of possible paths is reduced to a graph search amongst randomly selected samples [2]–[4].

In RRT, each generated sample is checked for collision; a collision-free sample is added as a vertex, and connections are made to surrounding vertices for tree expansion. Some versions of RRT provide geometric paths. In these cases, the considered system is assumed to achieve prefect tracking. However, in many applications, the equations of motion (i.e., the differential constraints) significantly restrict the reachable set and must therefore be accounted for. Notably, the kinodynamic version of RRT incrementally grows a tree of trajectories in the state space by sampling control inputs and simulating the motion of the system with these random control inputs over a time horizon [2], [3]. Hence, the trajectories that are generated by RRT are dynamically feasible by construction. One drawback with kinodynamic RRT is that exploration via random selection of control inputs is inefficient when the dynamics are complex and/or unstable. To remedy this, [5] proposed CL-RRT, which uses closed-loop prediction for trajectory generation. For systems subject to unmeasured disturbances, model uncertainty, and unmodeled dynamics, the preceding approaches do not guarantee that the resulting trajectories will satisfy the system output constraints.

In this work we present an extension of RRT, called SAFERRT, that builds a tree of local state-feedback controllers along with corresponding positively invariant sets. These positively invariant sets incrementally partition the state space into regions where the closed-loop trajectories produced by the local controllers are guaranteed to be safe. A set is positively invariant if any closed-loop state trajectory that starts inside the set remains in the set for all future times. Thus, by generating controllers whose corresponding positively invariant sets satisfy the input and output constraints, it is possible to guarantee that the closed-loop trajectories will be collision free and thus safe, avoiding the need for a computationally expensive collision check. Furthermore, by constructing a tree of feedback controllers that stabilize their randomly sampled reference equilibria, paths generated by SAFERRT inherently satisfy the systems' equations of motion and produce dynamically feasible trajectories without the need for a steering function.

The concept of using constraint-admissible positively invariant sets for path planning is based on [6], which developed the underlying idea for the spacecraft obstacle avoidance. The authors in [7] expanded upon [6] and posed the following questions about the algorithm:

1. How do we design the local controllers such that their positively invariant sets satisfy the input and output constraints?
2. How do we sample the output set such that the controller graph contains a path from an initial vertex whose positively invariant set contains the initial state to a goal vertex whose positively invariant set contains the goal state?
3. How do we provide good performance?

The first question was addressed in [7], which introduced a method for computing local controllers and compared it with a previous method that uses fixed-gain controllers with scaled invariant sets as in [6]. The method in [7] designs local controllers by solving a semi-definite program (SDP) to maximize the volume of the positively invariant set that satisfies state and input constraints. This approach increases the number of edges in the graph used for path planning which can lead to better performance, albeit at the expense of solving a computationally burdensome SDP in place of a simpler linear program (LP).

We leave the treatment of the third question as well as the treatment of systems affected by model uncertainty or

A. Weiss, C. Danielson, K. Berntorp, and S. Di Cairano are with Mitsubishi Electric Research Laboratories, Cambridge MA, USA. I. Kolmanovsky is with the University of Michigan, Ann Arbor MI, USA.

e-mail: {weiss,danielson,berntorp,dicairano}@merl.com, ilya@umich.edu

unmeasured disturbances exploiting robust control invariant sets [8], [9] to future work.

This paper focuses on the second question. In this work, a tree is expanded from the desired goal equilibrium by adding vertices of new controllers, along with their randomly sampled reference equilibria and corresponding constraint admissible positively invariant sets, until the initial state lies inside a positively invariant set of a vertex in the tree, at which point it is possible to generate a dynamically feasible and safe closed-loop trajectory from initial state to desired goal. By using a tree and simple feedback controllers to generate control inputs, the algorithm is ensured to have low computational burden. Related ideas are used in [10]–[13].

The following notation and definitions will be used. A set $\mathcal{O}$ is positively invariant (PI) for an autonomous system $x(t+1) = f(x(t))$ if $x(t+1) = f(x(t)) \in \mathcal{O}$ for every state $x(t) \in \mathcal{O}$. If $V(x)$ is a Lyapunov function for the stable autonomous system $x(t+1) = f(x(t))$, then any level-set $\mathcal{O} = \{x \in \mathbb{R}^n : V(x) \leq l\}$ is a PI set since $V(f(x)) \leq V(x)$. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a set of vertices $\mathcal{V}$ together with a list of ordered pairs $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ called edges. Vertices $i, j \in \mathcal{V}$ are called adjacent if $(i, j) \in \mathcal{E}$ is an edge. A path is a sequence of adjacent vertices. Two vertices $i, j \in \mathcal{V}$ are called connected if there exists a path connecting them. A cycle is a path that begins and ends with the same node. A tree $\mathcal{T}$ is a graph with no cycles. In this paper, a vertex $v_i \in \mathcal{V}$ of a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ will be used to index the triples of equilibrium points $\bar{x}_i$, controllers κ_i, and sets $\mathcal{O}_i$.

II. Path-Planning Problem and Algorithm

In this section we define the path-planning problem and describe the RRT algorithm for solving this problem. In Section III we will present our variant of RRT.

A. Path-Planning Problem

The objective of the path-planning problem is to drive the output $y(t)$ of a dynamic system to a specified goal position y_f. The dynamics of the system are represented by the following discrete-time model

$$x(t+1) = f\big(x(t), u(t)\big) \tag{1a}$$

$$y(t) = g\big(x(t)\big) \tag{1b}$$

where $x(t) \in \mathbb{R}^{n_x}$ is the state, $u(t) \in \mathbb{R}^{n_u}$ is the control input, and $y(t) \in \mathbb{R}^{n_y}$ is the output. The input $u(t)$ and output $y(t)$ are subject to constraints $u(t) \in \mathcal{U}$ and $y(t) \in \mathcal{Y}$. In particular, the output set $\mathcal{Y} \subseteq \mathbb{R}^{n_y}$ is generally non-convex, but can be described as a union of convex sets

$$\mathcal{Y} = \bigcup\nolimits_{k \in \mathcal{I}_\mathcal{Y}} \mathcal{Y}_k. \tag{2}$$

We assume that each output $\bar{y} \in \mathcal{Y}$ corresponds to a unique feasible equilibrium of the system (1), i.e., for each output $\bar{y} \in \mathcal{Y}$ there exists a state $\bar{x} \in \mathbb{R}^{n_x}$ and feasible input $\bar{u} \in \text{int}(\mathcal{U})$ such that $\bar{x} = f(\bar{x}, \bar{u})$ and $\bar{y} = g(\bar{x})$.

The path-planning problem is formally stated below.

Problem 1: Find a feasible input trajectory $u(t) \in \mathcal{U}$ for $t \in \mathbb{Z}_{\geq 0}$ that produces a feasible output trajectory $y(t) \in \mathcal{Y}$ for all $t \in \mathbb{Z}_{\geq 0}$ that converges to the target output $\lim_{t\to\infty} y(t) = y_f$. □

The main difficulty of Problem 1 is that the output set $\mathcal{Y}$ is generally non-convex. This problem is especially difficult when the dynamics (1) are complex or nonlinear and thus produce output trajectories $y(t)$ with complex geometry.

B. Path-Planning Algorithm

A popular method for solving Problem 1 is the rapidly-exploring random tree (RRT) algorithm [3]. Algorithm 1 details one of several versions of RRT. RRT constructs a tree $\mathcal{T}$ of equilibrium states connected by directed edges that indicate the existence of a control input that steers the state from one equilibrium point to another. The tree $\mathcal{T}$ is rooted at the initial state x_0 of the system (1) and the tree $\mathcal{T}$ is expanded until it contains the equilibrium state $\bar{x}_f$, corresponding to the target output y_f.

Algorithm 1 Rapidly-Exploring Random Tree

Input: $\{x_0, y_f, \mathcal{T}\}$
$\mathcal{T}$.`init`$(\bar{x}_f)$
while $\|g(\bar{x}_f) - g(\bar{x}_{\text{new}})\| > \epsilon$ **do**
  $\bar{x}_{\text{rand}} \leftarrow$ `RANDOM_STATE`();
  $\bar{x}_{\text{near}} \leftarrow$ `NEAREST_NEIGHBOR`$(\bar{x}_{\text{rand}}, \mathcal{T})$;
  $u_{\text{new}} \leftarrow$ `SELECT_INPUT`$(\bar{x}_{\text{rand}}, \bar{x}_{\text{near}})$;
  $\bar{x}_{\text{new}} \leftarrow$ `NEW_STATE`$(\bar{x}_{\text{near}}, u_{\text{new}})$;
  $\mathcal{T}$.`add_vertex`$(\bar{x}_{\text{new}})$
  $\mathcal{T}$.`add_edge`$(\bar{x}_{\text{new}}, \bar{x}_{\text{near}})$
end while

The tree $\mathcal{T}$ is initialized with the current state x_0 of the system (1). During each iteration, Algorithm 1 samples a random output $y_{\text{rand}} \in \mathcal{Y}$ from the environment $\mathcal{Y}$ and computes the corresponding equilibrium state $\bar{x}_{\text{rand}} =$ `RANDOM_STATE`(). The algorithm then finds the vertex $v_{\text{near}} \in \mathcal{V}$ of the tree $\mathcal{T}$ whose equilibrium state $\bar{x}_{\text{near}} =$ `NEAREST_NEIGHBOR`$(\bar{x}_{\text{rand}}, \mathcal{T})$ is closest to the random state $\bar{x}_{\text{rand}}$ in terms of the Euclidean distance

$$\texttt{NEAREST\_NEIGHBOR}(\bar{x}_{\text{rand}}, \mathcal{T}) = \arg\min_{v \in \mathcal{V}} \|\bar{x}_v - \bar{x}_{\text{rand}}\|.$$

Next, the algorithm selects a control input $u_{\text{new}} =$ `SELECT_INPUT`$(\bar{x}_{\text{rand}}, \bar{x}_{\text{near}})$ that results in a collision-free trajectory that steers the considered system from the state $\bar{x}_{\text{near}}$ toward the sampled state $\bar{x}_{\text{rand}}$, and is given by

$$\texttt{SELECT\_INPUT}(\bar{x}_{\text{rand}}, \bar{x}_{\text{near}}) = \operatorname*{argmin}_{u \in \mathcal{U}} \|\bar{x}_{\text{rand}} - f(\bar{x}_{\text{near}}, u)\|.$$

The resulting state

$$\bar{x}_{\text{new}} = \texttt{NEW\_STATE}(\bar{x}_{\text{near}}, u_{\text{new}}) = f(\bar{x}_{\text{near}}, u_{\text{new}})$$

is checked for collision. There are various methods for collision checking and it is often the most computationally burdensome step in the algorithm. If a collision occurs a new point is sampled, otherwise the vertex is added to the vertex list $\mathcal{V}$ of the tree $\mathcal{T}$ and the edge between $\bar{x}_{\text{new}}$ and $\bar{x}_{\text{near}}$ is added to the edge list $\mathcal{E}$. Algorithm 1 terminates when the new output $y_{\text{new}} = g(\bar{x}_{\text{new}})$ is within ϵ-distance from the target output y_f, i.e., $\|y_f - g(\bar{x}_{\text{new}})\| \leq \epsilon$.

III. SAFERRT ALGORITHM

This section describes the SAFERRT algorithm.

A. Path-Planning Algorithm

The SAFERRT algorithm is detailed in Algorithm 2. The main difference between the SAFERRT (Algorithm 2) and the basic RRT (Algorithm 1) is that SAFERRT uses information about the controller that will be used to track the equilibrium points $\bar{x}_v$ of the search-tree, $\mathcal{T}$, where $\bar{x}_v$ is the equilibrium point associated with the vertex $v \in \mathcal{V}$. Thus, SAFERRT leverages closed-loop information in the planning phase.

Algorithm 2 SAFERRT

1: **Input:** $\{x_0, y_f, \mathcal{T}\}$
2: $\mathcal{T}$.`init`$(\bar{x}_f)$
3: **while** $x_0 \notin \mathcal{O}_{\rm new}$ **do**
4: $\quad \bar{x}_{\rm rand} \leftarrow$ `RANDOM_STATE`$()$;
5: $\quad v_{\rm near} \leftarrow$ `NEAREST_NEIGHBOR`$(\bar{x}_{\rm rand}, \mathcal{T})$;
6: $\quad \bar{x}_{\rm new} \leftarrow$ `NEW_STATE`$(\bar{x}_{\rm rand}, v_{\rm near})$;
7: $\quad \{\kappa_{\rm new}, \mathcal{O}_{\rm new}\} \leftarrow$ `GENERATE_SAFE_SET`$(\bar{x}_{\rm new})$;
8: $\quad v \leftarrow \{\bar{x}_{\rm new}, \kappa_{\rm new}, \mathcal{O}_{\rm new}\}$
9: $\quad \mathcal{T}$.`add_vertex`(v)
10: $\quad \mathcal{T}$.`add_edge`$(\bar{x}_{\rm new}, \bar{x}_{\rm near})$
11: **end while**

As in Algorithm 1, the vertices $v \in \mathcal{V}$ of the search-tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ produced by Algorithm 2 include an equilibrium point $\bar{x}_v$ of the system (1). However, in Algorithm 2, the vertices also include a local feedback controller κ_v, used to stabilize the equilibrium point $\bar{x}_v$, and a subset $\mathcal{O}_v \subseteq \mathbb{R}^{n_x}$ of states $x \in \mathbb{R}^{n_x}$, for which it is safe to use the local controller κ_v. The safe set $\mathcal{O}_v$ is a level-set of a Lyapunov function $V_v(x)$ for the system (1) in closed-loop with the local controller κ_v, given by

$$\mathcal{O}_v = \big\{x \in \mathbb{R}^{n_x} : V_v(x - \bar{x}_v) \leq \rho\big\}. \tag{3}$$

Since $V_v(x-\bar{x}_v)$ is a Lyapunov function, any state trajectory $x(t)$ that initially resides $x(t_0) \in \mathcal{O}_v$ in the set $\mathcal{O}_v$, will remain inside the set $\mathcal{O}_v$ for all $t \geq t_0$. This property of the set $\mathcal{O}_v$ is called positive invariance and is fundamental to our algorithm. The scale-factor ρ of the positively invariant set (3) is chosen such that the positively invariant set $\mathcal{O}_v$ does not intersect any of the obstacles in the environment $\mathcal{Y}$ and satisfies the input constraints $\mathcal{U}$, i.e., $g\big(\mathcal{O}_v\big) \subseteq \mathcal{Y}$ and $\kappa_v(\mathcal{O}_v) \subseteq \mathcal{U}$. Such a positively invariant set $\mathcal{O}_v$ is called constraint-admissible. Hence, if the local controller κ_v is engaged when the state $x(t_0) \in \mathcal{O}_v$ is inside the constraint-admissible positively invariant set $\mathcal{O}_v$, it is guaranteed that the output $y(t) = g(x(t)) \in \mathcal{Y}$ does not collide with an obstacle since $g(\mathcal{O}_v) \subseteq \mathcal{Y}$ for all $t \geq t_0$. Hence, we often refer to constraint-admissible positively invariant sets as *safe sets*.

Similarly to Algorithm 1, Algorithm 2 selects a random equilibrium state $\bar{x}_{\rm rand}$ by sampling an output $y_{\rm rand} \in \mathcal{Y}$ from the output space $\mathcal{Y}$ and finding the corresponding equilibrium state $\bar{x}_{\rm rand}$ and input $\bar{u}_{\rm rand}$. The random equilibrium state $\bar{x}_{\rm rand}$ is moved to a nearby equilibrium state $\bar{x}_{\rm near}$ in the search-tree $\mathcal{T}$ (Line 5). However, in the SAFERRT algorithm, `NEAREST_NEIGHBOR`$(\bar{x}_{\rm rand}, \mathcal{T})$ is defined in terms of the safe sets $\mathcal{O}_v$, using the *Minkowski function* [14]

$$\Phi_{\mathcal{O}_v - \bar{x}_v}(x) = \min_{\lambda \geq 0} \ \lambda \tag{4a}$$

$$\text{subject to} \ \ x \in \lambda(\mathcal{O}_v - \bar{x}_v). \tag{4b}$$

The Minkowski function (4), applied to $x = \bar{x}_{\rm rand}$, measures the distance between the random equilibrium state $\bar{x}_{\rm rand}$ and the equilibrium point $\bar{x}_v$ for vertex $v \in \mathcal{V}$, by determining how much we need to scale the safe set $\mathcal{O}_v$ in order to contain $\bar{x}_{\rm rand} \in \lambda(\mathcal{O}_v - \bar{x}_v)$. If the Minkowski function is less than one, i.e., $\Phi_{O_v}(\bar{x}_{\rm rand}) \leq 1$, then $\bar{x}_{\rm rand}$ is inside the safe set O_v.

The nearest neighbor $v_{\rm near}$ is then chosen as the vertex $v \in \mathcal{V}$ in the search-tree $\mathcal{T}$ closest to $\bar{x}_{\rm rand}$ in terms of the Minkowski function (4),

$$\texttt{NEAREST\_NEIGHBOR}(\bar{x}_{\rm rand}, \mathcal{T}) = \arg\min_{v \in \mathcal{V}} \Phi_{\mathcal{O}_v - \bar{x}_v}(\bar{x}_{\rm rand}). \tag{5}$$

This nearest neighbor rule (5) tends to choose the vertices $v \in \mathcal{V}$ with the largest safe sets $\mathcal{O}_v$, since these sets are more likely to need less scaling $\lambda(\mathcal{O}_v - \bar{x}_v)$ to contain the random state $\bar{x}_{\rm rand}$. Thus, the output sample points $y_v \in \mathcal{Y}$ are farther apart than in the standard RRT. This means that Algorithm 2 generally requires less sampling to find a path through the environment $\mathcal{Y}$.

Next, Algorithm 2 selects a new equilibrium point $\bar{x}_{\rm new}$ by moving the random equilibrium point $\bar{x}_{\rm rand}$ closer to the nearest point $\bar{x}_{\rm near}$ using the rule

$$\texttt{NEW\_STATE}(\bar{x}_{\rm rand}, v_{\rm near}) = \bar{x}_{\rm near} + \frac{\alpha}{\lambda}\Big(\bar{x}_{\rm rand} - \bar{x}_{\rm near}\Big) \tag{6}$$

where $\lambda = \Phi_{\mathcal{O}_{\rm near} - \bar{x}_{\rm near}}(\bar{x}_{\rm rand})$ is the Minkowski function (4), $\alpha \in (0, 1)$ is a user-determined step-size, and $\mathcal{O}_{\rm near}$ is the safe set associated with vertex $v_{\rm near} \in \mathcal{V}$, generated during previous iterations of the algorithm. By the properties of the Minkowski function (4), the new equilibrium point $\bar{x}_{\rm new} \in \mathcal{O}_{\rm near}$, selected by (6), will be inside the safe set $\mathcal{O}_{\rm near}$ associated with the equilibrium point $\bar{x}_{\rm near}$. Hence, it is possible to safely move from the new equilibrium point $\bar{x}_{\rm new}$ to the existing equilibrium point $\bar{x}_{\rm near}$ using the local controller $\kappa_{\rm near}$ without collision. Thus SAFERRT does not require collision checking. Since Algorithm 2 is initialized with the equilibrium point $\bar{x}_f$ corresponding to the target output y_f, it follows that there exists a path in the search-tree $\mathcal{T}$ from the new equilibrium point $\bar{x}_{\rm new}$ to the target state $\bar{x}_f$.

Unlike the standard RRT, Algorithm 2 has an additional step (Line 7) where we compute the safe set $\mathcal{O}_{\rm new}$ for the new equilibrium point $\bar{x}_{\rm new}$. We use a pre-designed local controller $u = \kappa(x, \bar{x}_{\rm new})$ that asymptotically stabilizes the equilibrium state $\bar{x}_{\rm new}$. Stability is certified by a Lyapunov function $V_{v_{\rm new}}(x - \bar{x}_{\rm new})$. The safe set (3) can be computed by scaling the unit level-set $\mathcal{O}^0_{\rm new} = \{x : V_{v_{\rm new}}(x - \bar{x}_{\rm new}) \leq 1\}$ using a scaling factor ρ, such that $g\big(\rho(\mathcal{O}^0_{\rm new} - \bar{x}_{\rm new}) +$

$\bar{x}_{\text{new}}\big) \subseteq \mathcal{Y}$. This can be posed as an optimization problem

$$\rho_{\text{new}} = \quad \arg\max_{\rho} \quad \rho \tag{7a}$$

$$\text{subject to} \quad g\big(\rho(\mathcal{O}^0_{\text{new}} - \bar{x}_{\text{new}}) + \bar{x}_{\text{new}}\big) \subseteq \mathcal{Y}. \tag{7b}$$

Since the local controller $u = \kappa(x, \bar{x}_{\text{new}})$ asymptotically stabilizes the equilibrium state $\bar{x}_{\text{new}}$, level-sets of the Lyapunov function $V_{v_{\text{new}}}(x - \bar{x}_{\text{new}})$ are invariant, and thus (7) always has a strictly positive solution. The optimization problem (7) is generally non-convex, since $\mathcal{Y}$ is generally non-convex and $g(\cdot)$ is generally nonlinear. However, since the decision variable $\rho \in \mathbb{R}_+$ is a scalar, there are many computationally efficient methods for solving (7). In fact, if the dynamics (1) are linear and the constraints are polyhedral unions, this problem has a closed-form solution [7]. The new safe set $\mathcal{O}_{\text{new}}$ and controller κ_{new} are then given by

$$\begin{aligned} &\texttt{GENERATE\_SAFE\_SET}(\bar{x}_{\text{new}}) = \\ &\quad = \big\{\kappa(x, \bar{x}_{\text{new}}), \rho_{\text{new}}(\mathcal{O}^0_{\text{new}} - \bar{x}_{\text{new}}) + \bar{x}_{\text{new}}\big\}. \end{aligned} \tag{8}$$

Finally, Algorithm 2 adds the new equilibrium point $\bar{x}_{\text{new}}$, new controller κ_{new}, and new safe set $\mathcal{O}_{\text{new}}$ to the vertex list $\mathcal{V}$ for the search-tree $\mathcal{T}$, and adds an edge between v_{new} and v_{near} to the edge list $\mathcal{E}$. Algorithm 2 terminates when the safe set $\mathcal{O}_{\text{new}}$ of the new equilibrium point $\bar{x}_{\text{new}}$ contains $x_0 \in \mathcal{O}_{\text{new}}$ the initial state x_0. By the construction of the search-tree $\mathcal{T}$, this guarantees that there is a path in the search-tree $\mathcal{T}$ from the initial state x_0 to the target state $\bar{x}_f$. The next section describes how this path is used to produce a closed-loop trajectory $x(t)$ for the system (1) that converges $y(t) = g(x(t)) \to y_f$ to the target output y_f.

B. Path-Tracking Algorithm

In this section we describe how the search-tree $\mathcal{T}$ produced by Algorithm 2 can be used to solve Problem 1.

Algorithm 3 Closed-loop Execution

```
initial target equilibrium point x̄_v = x̄_1
for each time t ∈ ℕ do
    if x(t) ∈ O_{i+1} then
        update target equilibrium point x̄_v = x̄_{i+1}
    else
        use same target equilibrium point x̄_v = x̄_i
    end if
    u(t) = κ_v(x(t), x̄_v)
end for
```

Algorithm 3 summarizes the real-time execution of SAFERRT. Offline, the path-tracking algorithm searches the search-tree $\mathcal{T}$ for a sequence of equilibrium points $\bar{x}_1, \ldots, \bar{x}_f$ such that the initial state $x_0 = x(0) \in \mathcal{O}_1$ is contained in the safe set $\mathcal{O}_1$ of the first equilibrium point $\bar{x}_1$ and the final equilibrium point $\bar{x}_f$ corresponds to the target output $y_f = g(\bar{x}_f)$. Online at each time instance $t \in \mathbb{N}$, the path tracking algorithm uses the local controller $u(t) = \kappa_v(x(t), \bar{x}_v)$ to drive the system to the currently targeted equilibrium point $\bar{x}_v$, where $\bar{x}_1$ is the first targeted equilibrium point. The target equilibrium $\bar{x}_v$ is updated as $\bar{x}_v = \bar{x}_{i+1}$ when the state $x(t)$ reaches the safe set $\mathcal{O}_{i+1}$ of the next equilibrium point $\bar{x}_{i+1}$ in the sequence. The last targeted equilibrium point is the equilibrium point $\bar{x}_f$ corresponding to the target output y_f.

By the construction of the search-tree $\mathcal{T}$ in Algorithm 2, each equilibrium point $\bar{x}_v = \bar{x}_i$ is inside the safe set $\mathcal{O}_{i+1}$ of the next equilibrium point $\bar{x}_{i+1}$. Furthermore, the trajectory $x(t)$ from $\bar{x}_i$ to $\bar{x}_{i+1}$ is safe since the set $\mathcal{O}_{i+1}$ is a constraint-admissible positively invariant set. Algorithm 3 therefore guarantees that the output trajectory $y(t) \in \mathcal{Y}$ does not collide with any obstacles in the environment $\mathcal{Y}$ and converges $y(t) = g(x(t)) \to y_f$ to the target output $y_f = g(\bar{x}_f)$. A formal proof can be found in [7].

IV. Example: Spacecraft Maneuver Planning

In this section we use SAFERRT to design a spacecraft rendezvous maneuver.

A. SafeRRT for Spacecraft Maneuver Planning

Spaceraft rendezvous maneuvers are typically planned using the relative motion dynamics of the spacecraft in the non-inertial Hill's frame, which tracks the position and orientation of the target spacecraft on a nominal circular orbit. When the spacecraft separation distances are much smaller than the orbital radius, the linearized Hill-Clohessy-Wiltshire (HCW) equations [15] can be used to model the motion of a chaser spacecraft relative to the target spacecraft in the orbital plane,

$$\ddot{r}_1 = 3n^2 r_1 + 2n\dot{r}_2 + u_1, \tag{9a}$$

$$\ddot{r}_2 = -2n\dot{r}_1 + u_2, \tag{9b}$$

where the origin $[r_1, r_2] = [0, 0]$ is the position of the target spacecraft, r_1 is the radial position of the chaser spacecraft relative to the target spacecraft, r_2 is the in-orbital-track position of the chaser spacecraft relative to the target spacecraft, and n denotes the mean motion of the nominal orbit. The inputs u_1 and u_2 are external thrust forces normalized by the spacecraft mass acting on the spacecraft along the radial and orbital velocity directions, respectively. Assuming a sampling period of ΔT seconds, the dynamics (9) are discretized as

$$x(t+1) = Ax(t) + Bu(t), \tag{10a}$$

$$y(t) = Cx(t) \tag{10b}$$

where $x = [r_1, r_2, \dot{r}_1, \dot{r}_2]^{\mathrm{T}}$. The C matrix selects the relative position elements $y = [r_1, r_2]$ of the state x.

Since the spacecraft dynamics (10) are linear, we can simplify the general nonlinear expressions for safe sets (3), distance metric (4), and control design (7) used by SAFERRT. The local controllers are linear state-feedback controllers

$$u = F_v(x - \bar{x}_v) + \bar{u}_v, \tag{11}$$

where $\kappa_v = F_v$ is the controller gain, and $\bar{x}_v$ and $\bar{u}_v$ are an equilibrium state and input pair for the vertex $v \in \mathcal{V}$. Controller (11) is designed to asymptotically stabilize the equilibrium point $\bar{x}_v$. Thus, the matrix $A + BF_v$ is Schur. A quadratic Lyapunov function is of the form

$$V_v(x) = (x - \bar{x}_v)^{\mathrm{T}} P_v (x - \bar{x}_v), \tag{12}$$

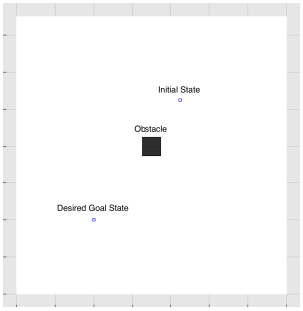


Fig. 1: Initial position and target position of the chaser spacecraft with an obstacle between. The white area is the output set $\mathcal{Y} \subset \mathbb{R}^2$ and the black square is the obstacle.

where P_v satisfies

$$(A + BF_v)^{\mathrm{T}} P_v (A + BF_v) - P_v \prec 0, \quad P_v \succ 0.$$

Hence, the safe sets (3) used by SAFERRT are the ellipsoidal sub-level sets of the Lyapunov function (12)

$$\mathcal{O}_v = \left\{ x \in \mathbb{R}^{n_x} : (x - \bar{x}_v)^{\mathrm{T}} P_v (x - \bar{x}_v) \leq \rho \right\}. \tag{13}$$

For ellipsoidal sets, the Minkowski function (4) has a closed-form solution given by

$$\Phi_{\mathcal{O}_v - \bar{x}_v}(x_{\mathrm{rand}}) = ||x_{\mathrm{rand}} - \bar{x}_v||_{P_v}.$$

The method (7) to compute the safe sets $\mathcal{O}_v$ for linear systems is detailed in [7]. It uses a single feedback gain matrix $F_v = F$ for every vertex $v \in \mathcal{V}$, and thus each of the local controllers (11) has a Lyapunov function that shares a common Lyapunov matrix $P_v = P$. The scale factor ρ in (13) is chosen to maximize the volume of $\mathcal{O}_v$ for each $v \in \mathcal{V}$ while satisfying input and output constraints. If the constraints are described as polyhedral unions, the optimal scale factor has an analytic expression [7].

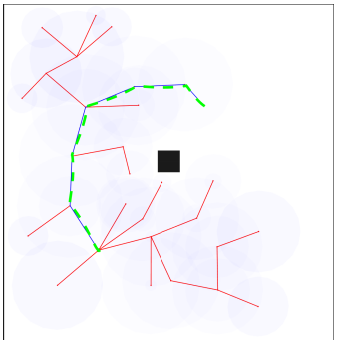
B. Results

We now provide simulations to illustrate the SAFERRT algorithm using (7) for safe set $\mathcal{O}_v$ computations, for two different step-sizes α. We consider a target spacecraft on a 415 km circular orbit, yielding $n = 1.1 \times 10^{-3}$ s^{-1}. The dynamics (9) are discretized with a sample period of $\Delta T = 30$ s. We apply SAFERRT to the problem of planning a maneuver around the obstacle shown in Fig. 1. The obstacle is a square with a 100 meter side length located at $[300, 400]^{\mathrm{T}}$ meters from the target spacecraft. The output set $\mathcal{Y}$ is the set difference of the bounding box $[-400, 1000] \times [-400, 1100]$ meters and the obstacle set. The component sets $\mathcal{Y}_1, \ldots, \mathcal{Y}_4$ that comprise the output set $\mathcal{Y} = \mathcal{Y}_1 \cup \cdots \cup \mathcal{Y}_4$ are obtained by flipping each of the 4 constraints that define the obstacle set and intersecting with the bounding box. The chaser spacecraft's normalized thrust forces must satisfy the constraints

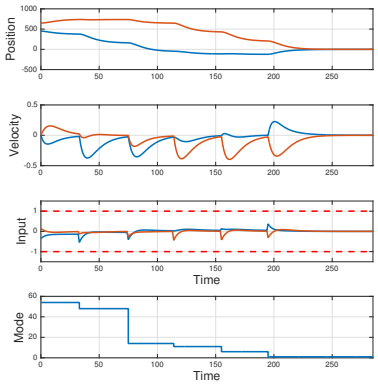
$$-10^{-2} \leq u_1, u_2 \leq 10^{-2} \quad \mathrm{N} \cdot \mathrm{kg}^{-1}. \tag{14}$$

The chaser spacecraft is initialized at $x_0 = [450, 650, 0, 0]^{\mathrm{T}}$ and the desired goal is the target spacecraft at the origin.

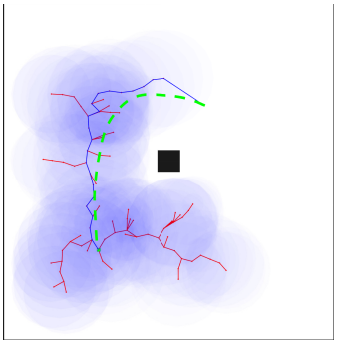
Fig. 2 shows the results of the SAFERRT algorithm with the `GENERATE_SAFE_SET` function of method (7). The



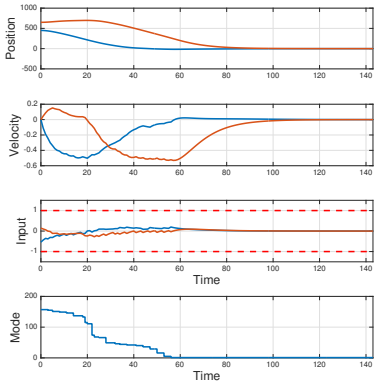
(a) $\alpha = 0.95$



(b) $\alpha = 0.95$



(c) $\alpha = 0.05$



(d) $\alpha = 0.05$

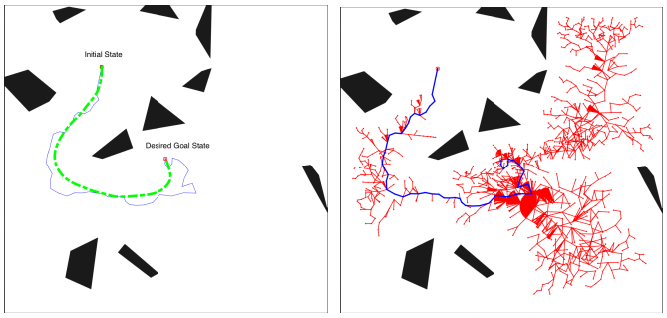
Fig. 2: (a) and (b) show the safe sets $\mathcal{O}_v$ and the edges of the search-tree $\mathcal{T}$ produced by Algorithm 2 using (7) to generate the safe sets for different values of the step-size α. (c) and (d) show the closed-loop state, input, and vertex trajectories produced by Algorithm 3 using the search-trees $\mathcal{T}$ shown in (a) and (b).

linear state-feedback gain $F_v = F$ is given by the linear quadratic regulator (LQR) with penalty matrices $Q = \mathrm{diag}(10^2, 10^2, 10^7, 10^7)$ and $R = (2\times10^7)I_2$ where $\mathrm{diag}(V)$ is a diagonal matrix with the elements of V on the diagonal and $I_2 \in \mathbb{R}^{2\times2}$ is the identity matrix. The Lyapunov matrix $P_v = P$ is the corresponding solution to the discrete-time algebraic Riccati equation. In Figs. 2a and 2c, the white area is the output set $\mathcal{Y} \subset \mathbb{R}^2$, the black square is the obstacle, the solid lines are the edges between vertices $\mathcal{V}$ in the search-tree $\mathcal{T}$, the highlighted solid blue line is the path through the tree connecting the initial and final outputs as labeled in Fig. 1, the transparent ellipsoids are the position slices of the safe positively invariant safe sets $\mathcal{O}_v$, and the dashed green line is the closed-loop chaser spacecraft trajectory as executed by Algorithm 3. Observe that each safe set $\mathcal{O}_v$ is scaled using (7) to a different size depending on both input and output constraints applicable to that vertex.

The results in Fig. 2a are for the step-size $\alpha = 0.95$, whereas Fig. 2c shows the results for $\alpha = 0.05$. These two extremes are chosen to highlight the range of performance of the algorithm. When the step size is large, the `NEW_STATE` function (6) generates a new sample near the boundary of the safe set $\mathcal{O}_v$. The search-tree $\mathcal{T}$ therefore covers more of the state-space with each additional sample, and hence requires fewer samples to connect initial and desired goal states. However, as a consequence, during closed-loop execution of the path, Algorithm 3 must maintain the currently targeted equilibrium point $\bar{x}_i$ until the trajectory nearly converges, before the state $x(t)$ reaches the safe set $\mathcal{O}_{i+1}$ of the following equilibrium target in the path. As convergence is exponential, the trajectory slows down significantly before it is able to switch to the next vertex in the path; this phenomenon can be observed in the velocity plot of Fig. 2b.

In contrast, when the step size is small as in Fig. 2c, many more samples must be generated to expand the search-tree before a path connects initial and desired goal states. This produces better closed-loop trajectories, but adds additional computational burden. The chaser spacecraft trajectory need not nearly converge to the currently targeted equilibrium point $\bar{x}_i$ because the density of the safe-sets in the search-tree results in the trajectory constantly entering into the safe set $\mathcal{O}_{i+1}$ of the following equilibrium target in the path. The trajectory is thus able to quickly switch between the sequence of target equilibria along the path, maintaining velocity and more quickly reaching the desired goal. As a result of the lack of convergence to intermediate equilibria along the path contained in the search-tree, the trajectory deviates from and smoothes out the path. Note that despite the deviation, the trajectory is guaranteed to be safe due to the positive invariance properties of the safe sets $\mathcal{O}_v$. Furthermore, since the safe sets were designed to satisfy both the input and output constraint, the closed-loop trajectories do not violate the input constraints as evidenced in Figs. 2b and 2d.

Finally, we highlight SAFERRTon a more challenging problem of navigating through a debris field. Fig. 3 shows the results with the `GENERATE_SAFE_SET` function of method (7) and $\alpha = 0.05$. Although the path through the debris field is jagged, Algorithm 3 produces a smooth closed-loop trajectory that avoids obstacles and satisfies thrust constraints.



(a) (b)

Fig. 3: Debris field navigation by SAFERRT using (7) to generate the safe sets (omitted for visual clarity). (a) closed-loop trajectory in dashed, along with path through the tree. (b) edges of the tree $\mathcal{T}$.

V. CONCLUSION

We presented SAFERRT, which constructs a tree of feedback controllers along with their randomly sampled reference equilibria and corresponding safe sets. By using constraint-admissible positively invariant sets, SAFERRT avoids the need for computationally expensive collision checking amongst vertices, since the dynamically feasible closed-loop trajectories are guaranteed to evolve inside the safe sets corresponding to their targeted equilibrium point. Also, closed-loop feedback control provides robustness to model uncertainty.

REFERENCES

[1] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Conference on Foundations of Computer Science*, 1979.

[2] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, 2001.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, 2011.

[5] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman *et al.*, "A perception-driven autonomous urban vehicle," *J. Field Robot.*, 2008.

[6] A. Weiss, C. Petersen, M. Baldwin, R. Erwin, and I. Kolmanovsky, "Safe positively invariant sets for spacecraft obstacle avoidance," *J. Guidance, Control, and Dynamics*, 2015.

[7] C. Danielson, A. Weiss, K. Berntorp, and S. Di Cairano, "Path planning using positive invariant sets," in *Conf. Decision and Control*, 2016.

[8] I. Kolmanovsky and E. G. Gilbert, "Theory and computation of disturbance invariant sets for discrete-time linear systems," *Mathematical problems in engineering*, 1998.

[9] E. Kerrigan, "Robust constraints satisfaction: Invariant sets and predictive control," Ph.D. dissertation, Dep. of Engineering, University of Cambridge, 2000.

[10] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *Int. Conf. Robot. Automation*, 2017.

[11] W. McConley, B. Appleby, M. Dahleh, and E. Feron, "A computationally efficient Lyapunov-based scheduling procedure for control of nonlinear systems with stability guarantees," *IEEE Trans. Autom. Control*, 2000.

[12] F. Blanchini, F. Pellegrino, and L. Visentini, "Control of manipulators in a constrained workspace by means of linked invariant sets," *J. Robust and Nonlinear Control*, 2004.

[13] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *Int. J. Robot. Res.*, vol. 29, no. 8, pp. 1038–1052, 2010.

[14] F. Blanchini and S. Miani, *Set-Theoretic Methods in Control*. Birkhäuser, 2008.

[15] B. Wie, *Spacecraft Dynamics and Control*. AIAA, 2010.