Fast Nonlinear Controller Synthesis using Reachability Analysis

Kristina Miller, Chuchu Fan and Sayan Mitra

***Abstract*— We present an algorithm for synthesizing controllers for nonlinear systems and reach-avoid requirements. The synthesized controllers consist of a reference trajectory and a tracking controller which drives the system's state towards the reference. Our key idea is to use a locally optimal reachability analysis algorithm to compute tracking error bounds relative to *arbitrary, and possibly piece-wise continuous*, reference trajectories. The overall synthesis algorithm uses these these error bounds to search for reference trajectories of increasing length, that work for increasingly smaller parts of the starting set. We also give a MILP-based subroutine for finding reference trajectories, given the tracking error characterization. The size of this problem grows at most linearly with the shapes of the obstacles and goals. The overall algorithm come with soundness guarantees. Our experimental results with a prototype implementation of a tool show that it can solve the synthesis problem for simple nonlinear vehicle models in interesting scenarios, with sub-second level running time, and that it fails gracefully in extreme cases.**

I. Introduction

Controller synthesis algorithms generate correct-by-construction controllers that guarantee that the system under control meets some higher-level tasks. By reducing designing and testing cycles, synthesis can help create safe autonomous systems that involve complex interactions of dynamics and decision logic. In general, however, synthesis problems are known to have high computational complexity as they involve solving two-player games.

Over the past decade, effective synthesis algorithms have been created by restricting the nature of the high-level task specifications, for example, to the so called class of *generalized reactivity* specifications [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. A preeminent solution approach is based on *discrete abstractions* [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]: First, a discrete ovaer-approximation of the control system is computed, and then a discrete controller or *an automaton* is synthesized by solving a two-player game [**?**]. Several algorithms employed this approach, first for linear and more recently to nonlinear models, and they are implemented in tools like CoSyMA [**?**], Pessoa [**?**], LTLMop [**?**], [**?**], Tulip [**?**], [**?**], and SCOTS [**?**]. Despite the advances, an enduring challenge here is the discrete abstraction step which leads to a severe state space explosion for nonlinear systems, even with 3-4 continuous dimensions.

An alternative attack on the problem has gained some traction recently as it can handle nonlinear models and shows promise for scalability. This approach decomposes the synthesis problem into two subtasks: (a) first a *tracking controller (g_t)* is designed to drive the plant to *arbitrary waypoints*; and (b) given the tracking error performance of g_t, sequences of waypoints are synthesized to meet the higher-level goals. Step (a) requires domain knowledge, and for typical vehicles and manipulators there are well-known controllers that can be tuned and used out of the box. Therefore, the problem boils down to characterizing the tracking errors and using that for solving (b). Within this broad solution space, a handful of specific algorithms have been explored in detail. FastTrack [**?**] uses Hamilton-Jacobi-based reachability analysis to produce the tracking error bounds. Convex optimization is used for the same in [**?**]. Reachability-based trajectory design (RTD) [**?**] computes forward reachable sets to guarantee not-at-fault in robotic decision making. Similar ideas are used in [**?**], [**?**] to define safe error bounds. Our previous work [**?**] proposed a similar algorithm for linear models where the waypoints were effectively computed using satisfiability (SAT)-solvers. These methods have been applied to different vehicle models, including for receding horizon real-time planning [**?**].

In this paper, we contribute a new algorithm in the above theme. Our algorithm can handle nonlinear models and reach-avoid goals or specifications. We use a locally optimal reachability analysis algorithm [**?**] to compute the tracking errors. We show that under reasonable assumptions, a small number of such reachtube computations can be used to characterize the tracking errors relative to arbitrary, and *possibly piece-wise continuous*, reference trajectories defined by the waypoints (Proposition 1). We give a novel procedure (Proposition 3) for solving (b) with this tracking error characterization using Mixed Integer Linear Programming (MILP). Our overall synthesis algorithm (Algorithm 4) is implemented in a software tool and it has the desirable soundness properties[1]. The algorithm partitions only the set of initial states of the system, increasingly finely, and looks for reference trajectories with increasing number of waypoints, until it either finds solutions that provably meet the reach-avoid specifications, or fails by hitting the partitioning or waypoint thresholds.

We present experimental evaluation of the tool on a simple nonlinear vehicle model in four different scenarios. The tool can indeed take any other nonlinear vehicular models with arbitrary state variables. Our experimental results show that our tool can successfully find safe reference trajectories in

Kristina Miller and Sayan Mitra are with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign IL, USA {kmmille2, mitras}@illinois.edu

Chuchu Fan is with the Department of Computing and Mathematical Sciences, California Institute of Technology, Los Angeles CA, USA chuchu@caltech.edu

[1]The tool and the related full version of the paper is available to download from this link.

all scenarios with sub-second level running time when the requirements are reasonable, and it fails predictably when the algorithm parameters are too extreme.

Other related approaches: Model Predictive Control (MPC) is another, very different, approach which has been synthesize controllers with reach-avoid type specifications. Implicit MPC [**?**] relies on solving an optimal control problem with explicit solutions [**?**], [**?**], [**?**] or with LP [**?**], QP [**?**], [**?**], or MILP solvers [**?**], [**?**], [**?**]. Probabilistic Road Maps (PRM) [**?**], Rapidly-exploring Random Trees (RRT) [**?**], and variants [**?**] are widely used for generating feasible trajectories but they only provide probabilistic guarantees. We defer a detailed comparisons to a longer version of this paper.

II. Preliminaries and Problem Statement

For an n-dimensional vector $x \in \mathbb{R}^n$, $x^{(i)}$ denotes the i^{th} entry. Euclidian norm of x is denoted by $\|x\|$. $B_\epsilon(x) \triangleq \{y \in \mathbb{R}^n \mid \|y - x\| \leq \epsilon\}$ is the ϵ-ball, $\epsilon > 0$, centered at x. Given a matrix $H \in \mathbb{R}^{n \times m}$ and a vector $b \in \mathbb{R}^n$, an (H, b)-polytope, denoted by $Poly(H, b)$, is the set $\{x \in \mathbb{R}^n | Hx \leq b\}$. Each row of the polytope defines a halfspace $H^{(i)}x \leq b^{(i)}$. The faces of the polytope are defined by $H^{(i)}x = b^{(i)}$. The number of faces of the polytope (rows in H) is $\mathsf{dP}(H) = n$. The sum of $x \in \mathbb{R}^n$ and $S \subseteq \mathbb{R}^n$, denoted by $x \oplus S$, is the set $\{x + y \mid y \in S\}$.

Definition 1 *An (n, m)-dimensional control system $\mathcal{A}$ is a 4-tuple $\langle \mathcal{X}, \Theta, \mathbf{U}, f \rangle$ where (i) $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space, (ii) $\Theta \subseteq \mathcal{X}$ is the initial set, (iii) $\mathbf{U} \subseteq \mathbb{R}^m$ is the input space, and (iv) $f : \mathcal{X} \times \mathbf{U} \to \mathcal{X}$ is the* dynamic function *that is Lipschitz continuous with respect to the first argument.*

Trajectories and reachtubes. An *input trajectory* $u : [0, T] \to \mathbf{U}$, is a continuous function, and we denote the set of all input trajectories by $\mathcal{U}$.

Given an input $u \in \mathcal{U}$, time bound $T \geq 0$, and an initial state $x_0 \in \Theta$, a *solution* of $\mathcal{A}$ is a continuous trajectory $\xi_u : [0, T] \to \mathcal{X}$ that satisfies (i) $\xi_u(0) = x_0$ and (ii) for any $t \in [0, T]$, the time derivative of ξ_u at t satisfies the differential equation:

$$\dot{\xi}_u(t) = f(\xi_u(t), u(t)). \tag{1}$$

For any $x_0 \in \Theta, u \in \mathcal{U}$, ξ_u is a state trajectory and we call such a pair (ξ_u, u) a *state-input trajectory pair*. The *time bound or duration* of ξ_u is denoted by $\xi_u.\mathsf{ltime} = T$.

Given a trajectory $\xi : [0, T] \to \mathcal{X}$ and a time $t > 0$, the *time shift* of ξ is a function $(\xi + t) : [t, t + T] \to \mathcal{X}$ defined as $\forall, t' \in [t, t + T], (\xi + t)(t') = \xi(t' - t)$. Strictly speaking, for $t > 0, \xi + t$ is not a trajectory. The *concatenation* of two trajectories $\xi_1 \frown \xi_2$ is a new trajectory in which ξ_1 is followed by ξ_2 shifted by $\xi_1.\mathsf{ltime}$. Many trajectories can be concatenated in the same way.

A (state) *reachtube* is a data structure for representing over-approximations of state trajectories. Specifically, in this paper reachtubes are represented as sequences of time-stamped convex sets and we require them to be suffix closed.

Definition 2 (Reachtube) *Given a control system $\mathcal{A} = \langle \mathcal{X}, \Theta, \mathbf{U}, f \rangle$, a subset $S \subseteq \Theta$, and a time-bound $T > 0$, an (S, T)-reachtube is a sequence of time-stamped sets $\{(R_i, t_i)\}_{i=0}^k$ such that*

(1) $t_0 = 0$ and $t_k = T$, and each $R_i \subseteq \mathcal{X}$ is convex.

(2) $\forall x_0 \in S, u \in \mathcal{U}, i \in \{0, \cdots, k\}$, the trajectory $\xi_u(t)$ with $\xi_u(0) = x_0$, satisfies (i) $\xi_u(t_i) \in R_i$; and (ii) if $i < k$ then $\forall t \in [t_i, t_{i+1}], \xi_u(t) \in \texttt{ConvexHull}(R_i, R_{i+1})$.

An (S, T)-reachtube $\{(R_i, t_i)\}_{i=0}^k$ is said to be *suffix closed* if $\forall j \in \{0, \cdots, k-1\}$, the suffix $\{(R_j, t_j), \ldots, (R_k, t_k)\}$, is a $(R_j, T - t_j)$-reachtube. For a closed control system (i.e., with no inputs), the definition for reachtube is the same as above with $U = \emptyset$. If a closed system is globally asymptotically stable, for example, and a reactube is represented by appropriate sub-level sets of its Lyapunov function, then the reachtube will be suffix closed. This property is used in the design on the `RecTrec` procedure.

The reachtubes can be computed using any reachability analysis tool, of which there are many [**?**], [**?**], [**?**]. For our experiments we use the method proposed in [**?**] which is also implemented in the C2E2 tool [**?**],[**?**].

Reach-avoid requirement in the workspace. The state space of $\mathcal{A}$ is $\mathcal{X} \subseteq \mathbb{R}^n$, while its *workspace* is a subspace $\mathcal{W} \subseteq \mathbb{R}^d$, with $d = 2$ or 3, which is the physical space in which the robots have to avoid obstacles and reach goals. Given a state vector $x \in \mathcal{X}$, its projection to $\mathcal{W}$ is denoted by $x \downarrow p$. That is, $x \downarrow p$ is a point in $\mathbb{R}^2$ for ground vehicles on the plane and a point in $\mathbb{R}^3$ for aerial and underwater vehicles. The vector x may include other variables like velocity, heading, pitch, etc., but $x \downarrow p$ only has the position in Cartesian coordinates. We assume that the goal set $G := Poly(H_G, b_G)$ and the unsafe set $\mathbf{O}$ (obstacles) are specified by polytopes in $\mathcal{W}$; $\mathbf{O} = \cup O_i$, where $O_i := Poly(H_{O,i}, b_{O,i})$ for each obstacle i.

The projection of a trajectory $\xi_u : [0, T] \to \mathcal{X}$ to $\mathcal{W}$ is written as $\xi_u \downarrow p : [0, T] \to \mathcal{W}$ and defined as $(\xi_u \downarrow p)(t) = \xi_u(t) \downarrow p$. We say that a trajectory $\xi_u(t)$ *satisfies a reach-avoid requirement given by unsafe set* $\mathbf{O}$ *and goal set* G if $\forall t \in [0, \xi_u.\mathsf{ltime}], \xi_u(t) \downarrow p \notin \mathbf{O}$ and $\xi_u(\xi_u.\mathsf{ltime}) \downarrow p \in G$. See Figure 1 for an example of a reach avoid requirement.

Tracking and reference controllers. The controllers we study in this paper combine a reference controller g_r and a tracking controller g_t to meet reach-avoid specifications.

A *reference state trajectory* (or *reference trajectory* for brevity) is a trajectory over $\mathcal{X}$ that the control system tries to follow. We denote reference trajectories by ξ_{ref}. Similarly, a *reference input trajectory* (or *reference input*) is a trajectory over $\mathbf{U}$ and we denote them as u_{ref}. A *reference controller* g_r for $\mathcal{A}$ is a state-input pair $(\xi_{\mathsf{ref}}, u_{\mathsf{ref}})$. Note these ξ_{ref} and u_{ref} are not necessarily solutions of (1). Figure 1 shows reference and actual solution trajectories.

Given a reference controller $g_r = (\xi_{\mathsf{ref}}, u_{\mathsf{ref}})$, a tracking controller g_t attempts to drive the trajectory of the closed-loop system to follow ξ_{ref}.

Definition 3 (Tracking controller) *Given a reference controller $g_r = (\xi_{ref}, u_{ref})$ for $\mathcal{A}$, a tracking controller is a (state*

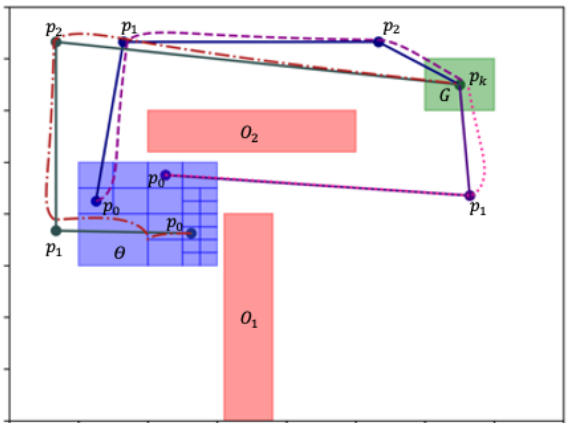

Fig. 1: Example reach-avoid scenario. The obstacles are labeled O_i, the initial set is labeled Θ, and the goal set is labeled G. Various reference trajectories that solve the scenario are shown. The waypoints of the reference trajectory are labeled p_i and the reference trajectories are the solid line. The dashed lines represents an example of an actual trajectory ξ_g.

feedback) function $g_t : \mathcal{X} \times \mathcal{X} \times \mathbf{U} \to \mathbf{U}$*, i.e., at time* t*, it takes in a current state of the system* x*, a reference trajectory state* $\xi_{ref}(t)$*, and a reference input* $u_{ref}(t)$*, and gives an input* $g_t(x, \xi_{ref}(t), u_{ref}(t)) \in \mathbf{U}$ *for* $\mathcal{A}$*.*

Thus, for a reference trajectory ξ_{ref}, a reference input u_{ref}, a tracking controller g_t, and an initial state $x_0 \in \Theta$, the resulting trajectory ξ_g of the closed control system ($\mathcal{A}$ closed with g) satisfies:

$$\dot{\xi}_g(t) = f\left(\xi_g(t), g_t\left(\xi_g(t), \xi_{\text{ref}}(t), u_{\text{ref}}(t)\right)\right), \forall\ t \in [0, T] \backslash D, \tag{2}$$

where D is the set of points in time where the second or third arguments of g_t are discontinuous[2]. For a broad class of nonlinear systems such as cars, drones, and underwater vehicles, there are many available techniques for designing tracking controllers to minimize the tracking error. Going forward, we will assume that the tracking controller is given and we will develop algorithms for automatically finding the reference controllers. However, in order to find reference trajectories (ξ_{ref}) concretely, we will first characterize the error tracking performance of g_t relative to *arbitrary* ξ_{ref}.

Definition 4 (Synthesis problem) *Given a* (n, m)*-dimensional nonlinear system* $\mathcal{A} = \langle \mathcal{X}, \Theta, \mathbf{U}, f \rangle$*, goal and unsafe sets* $G, \mathbf{O}$ *in the workspace* $\mathcal{W}$*, and a tracking controller* g_t*, we are required to find a partition* $\{\Theta_j\}_j$ *of* Θ*, and a reference controller* $(\xi_{j,ref}, u_{j,ref})$ *for each partition, such that* $\forall x_0 \in \Theta_j$*, the unique trajectory* ξ_g *of the closed system (*$\mathcal{A}$ *closed with* g_t *and* g_r*) starting from* x_0 *satisfies the reach-avoid requirement.*

We emphasize that the synthesized reference controllers $(\xi_{j,\text{ref}}, u_{j,\text{ref}})$ can be arbitrary functions (not necessarily state-input pairs), but, for each initial state $x_0 \in \Theta_j$, the closed loop trajectory ξ_g following ξ_{ref} *has to be* a valid state trajectory with corresponding input u generated by g_t and g_r.

In order to achieve this decomposition of the synthesis problem, we will have to carefully characterize the nature of the tracking error that the reference controller has to cope with. The rest of this section gives the related definitions and we finish with an example.

Tracking error dynamics. We will use a given tracking controller (found using standard approaches) to automatically synthesize reference controllers. We will study the tracking error, namely the difference between the closed system trajectory $\xi_g(t)$ and *any arbitrary* reference trajectory $\xi_{\text{ref}}(t)$. That is, at a fixed time t, the *tracking error* function $e : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^n$ takes the values of the system trajectory $\xi_g(t)$ and the reference trajectory $\xi_{\text{ref}}(t)$ and returns a scalar value that defines their difference: When $\xi_g(\cdot)$ and $\xi_{\text{ref}}(\cdot)$ are fixed, we write $e(t) = e(\xi_g(t), \xi_{\text{ref}}(t))$ which makes it a function of time.

One thing to remark here is that if $\xi_{\text{ref}}(t)$ is discontinuous, then $e(t)$ is also discontinuous. In this case, the derivative of $e(t)$ cannot be defined at the points of discontinuity.

Example 1 *We will use the kinematic bicycle model as an example. The state is* (x, y, θ) *and the inputs are* (v, ω)*.*

$$\dot{x} = v\cos(\theta); \quad \dot{y} = v\sin(\theta); \quad \dot{\theta} = \omega \tag{3}$$

A standard tracking controller is given by:

$$v = v_{ref}\cos(e_\theta) + k_1 e_x; \quad \omega = \omega_{ref} + v_{ref}(k_2 e_y + k_3 \sin(e_\theta)) \tag{4}$$

The error of the true state from the reference state is:

$$e_x = (x_{ref} - x)\cos(\theta) + (y_{ref} - y)\sin(\theta)$$
$$e_y = -(x_{ref} - x)\sin(\theta) + (y_{ref} - y)\cos(\theta); \quad e_\theta = \theta_{ref} - \theta$$

When written in matrix form, we see that the error is given in the local frame of $\mathcal{A}$*. The derivative of the error can be written as:* $\dot{e}_x = \omega e_y - v - v_{ref}\cos(e_\theta)$*;* $\dot{e}_y = -\omega e_x + v_{ref}\sin(e_\theta)$*;* $\dot{e}_\theta = \omega_{ref} - \omega$*. Replacing* v *and* ω *with the control law given of (4), the error dynamics are written as a closed loop system:*

$$\dot{e}_x = (\omega_{ref} + v_{ref}(k_2 e_y + k_3 \sin(e_\theta)))e_y - k_1 e_x$$
$$\dot{e}_y = -(\omega_{ref} - v_{ref}(k_2 e_y + k_3 \sin(e_\theta)))e_x + v_{ref}\sin(e_\theta)$$
$$\dot{e}_\theta = -v_{ref}(k_2 e_y + k_3 \sin(e_\theta)). \tag{5}$$

III. Synthesis Algorithm

In this section, we present our synthesis algorithm and the subroutines used therein. The main idea is that once the tracking controller g_t is fixed, the tracking error e between an arbitrary reference trajectory ξ_{ref} and the actual trajectory ξ_g of the closed system can be bounded, independent from the concrete value of the reference. Once we know the error bound, say e, we can use that to bloat $\mathbf{O}$ and shrink G, and then synthesize the actual ξ_{ref} such that it is e away from the obstacles (inside the goal set). This then ensures that all trajectories following this ξ_{ref} will meed the reach-avoid specification.

[2] ξ_g is a standard solution of ODE with piece-wise continuous right hand side.

Outline: In Section III-A, we show that the tracking error can be bounded using reachability analysis, independent from ξ_{ref} and u_{ref} by the `RecTrec` function. In Section III-B, we present `Waypt2Traj` for constructing reference trajectories from workspace waypoints. In Section III-D, we introduce our tool to find such ξ_{ref} by solving a mixed-integer linear program (MILP). In Section III-C, we present the overall synthesis algorithm, which uses `RecTrec` to construct waypoints which when passed through `Waypt2Traj` gives the solutions to the synthesis problem.

A. Bounding tracking error with reachtubes

The *Reachability-based Tracking Error Computation (*`RecTrec`*)* procedure computes tracking error bounds using reachability analysis. Let us fix a control system $\mathcal{A} = \langle \mathcal{X}, \Theta, \mathbf{U}, f \rangle$ and a tracking controller g_t for $\mathcal{A}$ for the rest of the paper. Let us consider *an arbitrary state-input reference trajectory pair* $(\xi_{\mathsf{ref}}, u_{\mathsf{ref}})$. Let $\varepsilon_0 > 0$ be a constant such that for every $x_0 \in \Theta$, $||e(0)|| = ||e(\xi_g(0), \xi_{\mathsf{ref}}(0))|| \leq \varepsilon_0$. Let $E \subseteq \mathbb{R}^n$ be such that $B_{\varepsilon_0}(0) \subseteq E$ and let $T \gg \xi_{\mathsf{ref}}.\mathsf{ltime}$. Suppose we have a precomputed, suffix-closed (E,T)-reachtube $\{(R_i, t_i)\}_{i=0}^{k}$ for the tracking error dynamics $e(t)$. In the remainder of this section, we will see how this reachtube can be used to compute a bound on the tracking $e(t)$ for all $t \leq T$.

Consider the prefix of the reachtube $\{(R_i, t_i)\}_{i=0}^{j}$, $j \leq k$, such that $\xi_{\mathsf{ref}}.\mathsf{ltime} \in [t_{j-1}, t_j]$. From Definition 2, $\{(R_i, t_i)\}_{i=0}^{j}$ is guaranteed to contain $e(\xi_{\mathsf{ref}}(t), \xi_g(t))$ for all $t \in [0, \xi_{\mathsf{ref}}.\mathsf{ltime}]$ and for all $x_0 \in \Theta$.

However, E might be much larger than the initial error bound $B_{\varepsilon_0}(0)$ which will make the computed reachtube over-approximation too conservative. To overcome this hurdle, we use `ErrBound` (Algorithm 1) to find a prefix of $\{(R_i, t_i)\}_{i=0}^{k}$ that is a less conservative approximation of $e(t)$.

Inn more detail, `ErrBound` (Algorithm 1) takes as input an (E,T)-reachtube $\{(R_i, t_i)\}_{i=0}^{k}$ (as discussed above), the initial error bound ε_0, a minimum time bound Tmin of ξ_{ref}. This minimum time bound is needed since we do not know the length of ξ_{ref} ahead of time. However, if we assume $\xi_{\mathsf{ref}}.\mathsf{ltime} \geq \mathsf{Tmin}$, then we can bound the error over $t \in [0, \xi_{\mathsf{ref}}.\mathsf{ltime}]$. At Line 1, it first finds the index $\mathsf{idx}_{\mathsf{init}}$ such that $R_{\mathsf{idx}_{\mathsf{init}}}$ is the smallest superset of $B_{\varepsilon_0}(0)$ among the R_i's. At Line 2, the algorithm finds the minimum index $\mathsf{idx}_{\mathsf{end}}$ such that $t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}$ is greater than Tmin. Then, `ErrBound` computes ℓ as the maximum radius of $\mathtt{ConvexHull}(R_i, R_{i+1})$ for the R_i's between indices $\mathsf{idx}_{\mathsf{init}}$ and $\mathsf{idx}_{\mathsf{end}}$ (line 3). If the reachtube reaches a fixpoint at $R_{\mathsf{idx}_{\mathsf{end}}}$ (i.e. $R_{\mathsf{idx}_{\mathsf{end}}} \supseteq R_{\mathsf{idx}_{\mathsf{end}}+1}$), we can show that ℓ is an upper bound of $\|e(t)\|$ for any time $t \in [0, \xi_{\mathsf{ref}}.\mathsf{ltime}]$ where $\xi_{\mathsf{ref}}.\mathsf{ltime} > \mathsf{Tmin}$.

Algorithm 1: `ErrBound`

input : $\{(R_i, t_i)\}_{i=0}^{k}, \varepsilon_0, \mathsf{Tmin}$

1 $\mathsf{idx}_{\mathsf{init}} \leftarrow \underset{i \in \{i \mid B_{\varepsilon_0}(0) \subseteq R_i\}}{\arg\min} \mathtt{Radius}(B_{\varepsilon_0}(0))$

2 $\mathsf{idx}_{\mathsf{end}} \leftarrow \min\{i \mid t_i - t_{\mathsf{idx}_{\mathsf{init}}} > \mathsf{Tmin}\} - 1$

3 $\ell \leftarrow \max\{\mathtt{Radius}(\mathtt{ConvexHull}(R_i, R_{i+1})) \mid i = \mathsf{idx}_{\mathsf{init}}, \cdots, \mathsf{idx}_{\mathsf{end}} - 1\}$

4 **return** ℓ, $\mathsf{idx}_{\mathsf{end}}$

Lemma 1 *Consider any differentiable reference trajectory ξ_{ref} with $\xi_{ref}.\mathsf{ltime} > \mathsf{Tmin}$. Suppose with inputs parameters (E,T)-reachtube $\{(R_i, t_i)\}_{i=0}^{k}$ for the tracking error dynamics relative to ξ_{ref}, $\varepsilon_0 > 0$, and Tmin (as defined above), the* `ErrBound` *subroutine produces output ℓ and idx_{end}. In addition, suppose $R_{\mathsf{idx}_{end}} \supseteq R_{\mathsf{idx}_{end}+1}$. Then,*

$$\|e(t)\| \leq \ell, \forall\ t \in [0, \xi_{ref}.\mathsf{ltime}], and \tag{6}$$

$$e(\xi_{ref}.\mathsf{ltime}) \in R_{\mathsf{idx}_{end}}. \tag{7}$$

Proof: First, since $B_{\varepsilon_0}(0) \subseteq R_{\mathsf{idx}_{\mathsf{init}}}$, by the suffix closure property of $\{(R_i, t_i)\}_{i=\mathsf{idx}_{\mathsf{init}}}^{k}$, we know that the reachtube contains all the trajectories of $e(t)$ starting from $B_{\varepsilon_0}(0)$. From Line 2, it follows that $\mathsf{Tmin} \in [t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}, t_{\mathsf{idx}_{\mathsf{end}}+1} - t_{\mathsf{idx}_{\mathsf{init}}})$. There are two cases to consider:

First, let $0 \leq t < t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}$, that is, the index of the reachtube covering $\mathsf{idx}_{\mathsf{init}}$ to $\mathsf{idx}_{\mathsf{end}}$. From Definition 2, we know that $\forall i = \mathsf{idx}_{\mathsf{init}}, \cdots, \mathsf{idx}_{\mathsf{end}} - 1$, $\forall t \in [t_i - t_{\mathsf{idx}_{\mathsf{init}}}, t_{i+1} - t_{\mathsf{idx}_{\mathsf{init}}}]$, $e(t) \in \mathtt{ConvexHull}(R_i, R_{i+1})$. By taking the maximum radius of those convex hulls, we have $\forall t \in [0, t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}], \|e(t)\| \leq \ell$.

Finally, we consider $t \geq t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}$. Since $R_{\mathsf{idx}_{\mathsf{end}}} \supseteq R_{\mathsf{idx}_{\mathsf{end}}+1}$, we know that for $t \in [t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}, t_{\mathsf{idx}_{\mathsf{end}}+1} - t_{\mathsf{idx}_{\mathsf{init}}}], e(t) \in \mathtt{ConvexHull}(R_{\mathsf{idx}_{\mathsf{end}}}, R_{\mathsf{idx}_{\mathsf{end}}+1}) = R_{\mathsf{idx}_{\mathsf{end}}}$. Similarly, for $t \geq t_{\mathsf{idx}_{\mathsf{end}}+1} - t_{\mathsf{idx}_{\mathsf{init}}}$, we can get $e(t) \in R_{\mathsf{idx}_{\mathsf{end}}}$ and we say that the reachtube reaches a fixpoint at $R_{\mathsf{idx}_{\mathsf{end}}}$. Therefore, we have $\forall t \in [t_{\mathsf{idx}_{\mathsf{end}}} - t_{\mathsf{idx}_{\mathsf{init}}}, \xi_{\mathsf{ref}}.\mathsf{ltime}], \|e(t)\| \leq \ell$. ∎

For the output error bound ℓ from the `ErrBound` subroutine, Lemma 1 gives the following immediately.

Corollary 1 $\forall t \in [0, \xi_{ref}.\mathsf{ltime}]$, $\xi_g(t) \in B_\ell(\xi_{ref}(t))$.

Discontinuous references: Now we consider the case when ξ_{ref} is piece-wise continuous. Let ξ_{ref} be a concatenation of s continuous state trajectories $\xi_{\mathsf{ref},1} \frown \cdots \frown \xi_{\mathsf{ref},s}$. We say that ξ_{ref} is a piece-wise reference trajectory. For an arbitrary $x_0 \in \Theta$, let $\tau_j > 0$, $j = 1, \cdots, s-1$ be the switching times when the closed system trajectory ξ_g switches from tracking $\xi_{\mathsf{ref},j}$ to track $\xi_{\mathsf{ref},j+1}$. So, each $(\xi_{\mathsf{ref},j}, u_{\mathsf{ref},j})$ is a valid continuous state-input trajectory pair and the corresponding error $e_j(t)$ is continuous. The value of $e_j(t)$ can be bounded using the `ErrBound` subroutine, *provided* we know the error $e_j(\tau_{j-1})$ at the beginning of tracking $\xi_{\mathsf{ref},j}$. However, the main challenge to glue these error bounds together is that $e(t)$ would be discontinuous because of the piece-wise $\xi_{\mathsf{ref}}(t)$. Therefore, at a time τ_j, even if we know the bound for $\|e_j(\tau_j)\|$, the set of initial states for $e_{j+1}(\tau_j)$ is not the same at $e_j(\tau_j)$.

An insight we can get from Example 1 is that although $e(t)$ is discontinuous at time τ_js, some of the variables influencing $e(t)$ are continuous. For example, $e_x(t)$ and $e_y(t)$ in Example 1, representing the errors in position, are

continuous since both the actual and reference positions of the vehicle are continuous. If we can further bound the value of *other* error components over the discontinuities, we could analyze the error bound for the entire piece-wise reference trajectory.

Let us write $e(t)$ as $[e_p(t), e_r(t)]$, where $e_p(t) = e(t) \downarrow p$ is the projection to $\mathcal{W}$ and $e_r(t)$ is the remaining components. In Example 1, $e_r(t) = e_\theta(t)$, which can always be bounded since both θ_{ref} and θ belongs to $[0, 2\pi]$.

Without loss of generality, let us assume that all tracking errors e_j for segment $\xi_{\mathsf{ref},j}$ share the same dymamics , and therefore, have the same reachtube. In this way, we can use a single reachtube for the tracking error across different segments of the piece-wise reference trajectory ξ_{ref}. This assumption makes sense for nonlinear vehicular models since the tracking controller is often designed in a local coordinate with the reference trajectory as the origin. Therefore, the error dynamics is often independent of the absolute values of the reference trajectory.

`RecTrec` (Algorithm 2) uses `ErrBound` subroutine to compute the error bounds for each segment of the piece-wise reference trajectory. It takes as input (1) (E, T)-reachtube $\{(R_i, t_i)\}_{i=0}^k$ and $\varepsilon_0 > 0$ satisfying the same conditions as the inputs to `ErrBound`, (2) $\mathsf{Tmin} > 0$ which is a minimum duration of each $\xi_{\mathsf{ref},j}$ segment, (3) $s > 0$ which is the number of segments in the piece-wise reference trajectory, and (4) $\Delta_d > 0$ which is an upper-bound for the error changes in the discontinuous e_r variables across the reference segments. For each $j = 1, \cdots, s$, $B_{\varepsilon_j}(0)$ is the set of tracking error when starting to track the jth segment $\xi_{\mathsf{ref},j}$.

Algorithm 2: `RecTrec`

input : $\{(R_i, t_i)\}_{i=0}^k, \varepsilon_0, \mathsf{Tmin}, s, \Delta_d$

for $j = 1, \cdots, s$ **do**

  $\ell_j, \mathsf{idx}_{\mathsf{end}} \leftarrow \texttt{ErrBound}(\{(R_i, t_i)\}_{i=0}^k, \varepsilon_{j-1}, \mathsf{Tmin})$

  $\varepsilon_j \leftarrow \texttt{Radius}(R_{\mathsf{idx}_{\mathsf{end}}}) + \Delta_d$

return $\ell_1, ..., \ell_s$

Proposition 1 *Consider any piece-wise continuous reference trajectory $\xi_{\mathsf{ref}} = \xi_{\mathsf{ref},1} \frown \cdots \frown \xi_{\mathsf{ref},s}$. Suppose with input parameters (E, T)-reachtube $\{(R_i, t_i)\}_{i=0}^k$ and ε_0 to satisfying the requirements of Lemma 1, and $\forall t$, $\|e_r(t)\| \leq \Delta_{e_r}$, the output from* `RecTrec` *is $\{\ell_j\}_{j=1}^s$. Then,*

$$\|e_j(t)\| \triangleq \|e(\xi_g(t), \xi_{\mathsf{ref},j}(t))\| \leq \ell_j,$$

for each $j = 1, \ldots, s$ and for all $t \in [0, \xi_{\mathsf{ref}}.\mathsf{ltime}]$.

The proposition is proved by induction applying Lemma 1 iteratively from the first segment onward.

B. Constructing reference trajectories from waypoints

If $\xi_{\mathsf{ref}}(t)$ is a *piece-wise linear reference trajectory*, we say $\xi_{\mathsf{ref}}(t) \downarrow p$ is piece-wise linear in the workspace $\mathcal{W}$. In $\mathcal{W}$, a piece-wise linear trajectory can be determined by the endpoints of each segment. We call such endpoints the *waypoints* of the piece-wise reference. In Figure 1, the points $p_0, \cdots, p_s$ are waypoints of $p(t) = \xi_{\mathsf{ref}}(t) \downarrow p$.

Consider any vehicle on the plane[3] with state variables p_x, p_y, θ, v (x-position, y-position, heading direction, linear velocity) and input variables a, ω (acceleration and angular velocity). Once the waypoints $\{p_i\}_{i=0}^s$ are fixed, and if we enforce constant speed $\bar{v}$ (i.e., $\xi_{\mathsf{ref}}(t) \downarrow v = \bar{v}$ for all $t \in [0, \xi_{\mathsf{ref}}.\mathsf{ltime}]$), then $\xi_{\mathsf{ref}}(t)$ can be uniquely defined by $\{p_i\}_{i=0}^s$ and $\bar{v}$ using `Waypt2Traj` (Algorithm 3). The semantics of ξ_{ref} and u_{ref} returned by `Waypt2Traj` is that the reference trajectory requires the vehicle to move at a constant speed $\bar{v}$ along the lines connecting the waypoints $\{p_i\}_{i=0}^s$. This constant speed $\bar{v}$ is an input to the main synthesis algorithm(Algorithm 4). Every mention of $\bar{v}$ in this section is the same as this input. $\xi_{\mathsf{ref}}(t), u_{\mathsf{ref}}(t)$ can also be constructed using `Waypt2Traj` making v an input and dropping a.

We notice that if $s = 1$, $\xi_{\mathsf{ref}}(t), u_{\mathsf{ref}}(t)$ returned by `Waypt2Traj` is a valid state-input trajectory pair. However, if $s > 1$, then the $\xi_{\mathsf{ref}}(t), u_{\mathsf{ref}}(t)$ returned is usually not a valid state-input trajectory pair. This is because $\theta_{\mathsf{ref}}(t)$ is discontinuous at the waypoints and no bounded inputs $u_{\mathsf{ref}}(t)$ can drive the vehicle to achieve such $\theta_{\mathsf{ref}}(t)$. Instead, $\xi_{\mathsf{ref}}(t)$ a concatenation of $\xi_{\mathsf{ref},i}$ where $(\xi_{\mathsf{ref},i}, u_{\mathsf{ref},i})$ are state-input trajectory pairs returned by the function $\texttt{Waypt2Traj}(\{p_{i-1}, p_i\}, \bar{v})$.

Algorithm 3: $\texttt{Waypt2Traj}(\{p_i\}_{i=0}^s)$

input : $\{p_i\}_{i=0}^s, \bar{v}$

$\forall t \in [0, \sum_{i=1}^s \frac{\|p_j - p_{j-1}\|_2}{\bar{v}}]$, $v_{\mathsf{ref}}(t) \leftarrow \bar{v}$,
$a_{\mathsf{ref}}(t) \leftarrow 0, \omega_{\mathsf{ref}}(t) \leftarrow 0$

$\forall i \geq 1, \forall t \in \Big[\sum_{j=1}^{i-1} \frac{\|p_j - p_{j-1}\|_2}{\bar{v}}, \sum_{j=1}^{i} \frac{\|p_j - p_{j-1}\|_2}{\bar{v}}\Big)$,
$p_{\mathsf{ref}}(t) \leftarrow p_{i-1} + \bar{v}t - \sum_{j=1}^{i-1} \|p_j - p_{j-1}\|_2$,
$\theta_{\mathsf{ref}}(t) \leftarrow \mathrm{mod}(\mathrm{atan2}((p_{y,i} - p_{y,i-1}), (p_{x,i} - p_{x,i-1}), 2\pi)$

$\xi_{\mathsf{ref}}(t) \leftarrow [p_{\mathsf{ref}}(t), \theta_{\mathsf{ref}}(t), v_{\mathsf{ref}}(t)]$

$u_{\mathsf{ref}}(t) \leftarrow [a_{\mathsf{ref}}(t), \omega_{\mathsf{ref}}(t)]$

return $\xi_{\mathsf{ref}}(\cdot), u_{\mathsf{ref}}(\cdot)$

Proposition 2 *Given a sequence of waypoints $\{p_i\}_{i=0}^s$ and a constant speed $\bar{v}$, $\xi_{\mathsf{ref}}(t), u_{\mathsf{ref}}(t)$ produced by* $\texttt{Waypt2Traj}(\{p_i\}_{i=0}^s, \bar{v})$ *satisfy:*

(1) $p_{\mathsf{ref}}(t) = \xi_{\mathsf{ref}}(t) \downarrow p$ is a piece-wise continuous function connecting $\{p_i\}_{i=0}^s$.

(2) At time $\tau_i = \sum_{j=1}^i \|p_j - p_{j-1}\|_2 / \bar{v}$, $p_{\mathsf{ref}}(\tau_i) = p_i$. We call $\{\tau_1, \ldots, \tau_k\}$ the switching times.

(3) $\xi_{\mathsf{ref}}(t) = \xi_{\mathsf{ref},1}(t) \frown \cdots \frown \xi_{\mathsf{ref},s}(t)$ and $u_{\mathsf{ref}}(t) = u_{\mathsf{ref},1}(t) \frown \cdots \frown u_{\mathsf{ref},s}(t)$, where $(\xi_{\mathsf{ref},i}, u_{\mathsf{ref},i})$ are state-input trajectory pairs returned by the function $\texttt{Waypt2Traj}(\{p_{i-1}, p_i\}, \bar{v})$

C. Main Synthesis Algorithm

Our main synthesis algorithm (Algorithm 4) uses reachability analysis on the error dynamics of the tracking controller g, to find reference trajectories from the initial set Θ

[3]A similar construction works for vehicles in 3D workspaces with additional variables.

leading to the goal set G. The key inputs to the algorithm are: (i) a list of obstacles $\mathbf{O}$ (represented as polytopes), (ii) a reachtube given by $\{R_i, t_i\}_{i=0}^{k}$ (used to compute the bounds as described in `ErrBound` and three parameters specifying the minimum time per segment (Tmin), the maximum number of line segments in a reference trajectory (Segmax), and the minimum size of initial set partitions (Partmin). The output from the algorithm is `Decided` which stores for each partition Θ' of the initial set Θ, a corresponding reference trajectory `Decided` or FAIL.

Algorithm 4: Controller synthesis algorithm

```
   input : O, G, {R_i, t_i}_{i=0}^k, Tmin, Segmax, Partmin
   initially: Undecided ← {Θ}, Decided ← ∅, seg ← 1
1  while (Undecided ≠ ∅) do
2      for Θ' ∈ Undecided do
3          {ℓ_i}_{i=1}^{Segmax} ← RecTrec({R_i, t_i}_{i=0}^k, ε_0, Tmin, Segmax, Δ_d)
             where ε_0 ← Radius(Θ')
4          while seg ≤ Segmax do
5              if CheckSAT(Θ', seg, O, G, {ℓ_i}_{i=1}^{seg}, Tmin) =
                 (SAT, {p_i}_{i=0}^{seg}) then
6                  ξ'_ref, u'_ref ← Waypt2Traj({p_i}_{i=0}^{seg})
7                  Decided ← Decided ∪ ⟨Θ', ξ'_ref, u'_ref⟩
8                  Undecided ← Undecided \ {Θ'}
9                  seg ← 1
10                 break
11             else
12                 seg ← seg + 1
13         if seg > Segmax then
14             seg ← 1
15             if Radius(Θ') > Partmin then
16                 Undecided ← Undecided ∪ Partition(Θ') \ {Θ'}
17             else
18                 Undecided ← Undecided \ {Θ'}
19                 Decided ← Decided ∪ ⟨Θ', FAIL⟩
20 return Decided
```

At a high-level, the algorithm proceeds as follows: It maintains a partition called `Undecided` of the initial set Θ. For each subset in `Undecided` it attempts to find a reference trajectory ξ_{ref} of increasing length. If a satisfying sequence of seg waypoints $\phi_{\text{waypoints}}$ is found for a subset Θ' of `Undecided` (line 5), then the triple containing the initial set, corresponding reference trajectory $\langle \Theta', \xi'_{\text{ref}}, u'_{\text{ref}} \rangle$ is added to the set `Decided` (line 7) and Θ' is removed from `Undecided`. (In the next section, we present an implementation of `CheckSAT` using MILP.)

On the other hand, if the satisfiability check on line 5 fails, then the length of the sequence of waypoints searched is increased, until the maximum limit Segmax is reached. Once the maximum limit is reached and yet a reference trajectory is not found, then Θ' is partitioned into smaller sets and added to `Undecided` (line 16). When a reference cannot be found for a Θ' that is smaller than the smallest size partition Partmin (with the longest Segmax), then the algorithm gives up on Θ' by adding $\langle \Theta', \text{FAIL} \rangle$ to `Decided`.

The key invariant of the algorithm is that (i) the set `Decided` always maintains a list of subsets of the initial set Θ' for each of which either a reference trajectory has been found or the algorithm has given up (by reaching the partitioning and segmenting bounds), and (ii) `Undecided` contains the remaining subsets of Θ for which decision is yet to be made. The soundness of the overall algorithm is given by Theorem 1. The proof follows the standard pattern of partitioning-based algorithms and will appear in the full version of the paper.

Theorem 1 *Suppose the inputs to Algorithm 4 satisfy the assumptions in Propositions 1 and 3 and the output is* `Decided` *Then, for each* $\langle \Theta', \xi_{ref}, u_{ref} \rangle \in$ `Decided` , $\forall x_0 \in \Theta'$*, the unique trajectory* ξ_g *of the closed system (*$\mathcal{A}$ *closed with* $g(\cdot, \xi_{ref}, u_{ref})$*) starting from* x_0 *satisfies the reach avoid requirement specified by* $\mathbf{O}, G$*.*

D. MILP for implementing `CheckSAT`

In this section, we briefly describe a MILP encoding of the `CheckSAT` function used in our synthesis algorithm. The key point to note here is that our encoding generates only linear, $O(\text{Segmax} \times (|\mathbf{O}|+|G|))$, number of constraints.

The following mixed-integer linear constraints define the search for the waypoints implemented by the `CheckSAT` subroutine. Here $p_0, \ldots, p_s$ are the decision variables, or the waypoints to be search for, each taking values in the workspace $\mathcal{W}$. The ℓ_i's are the constant tracking error bounds computed for the s segments using `RecTrec`.

$$p_0 = p_{\text{ref}}(0) \tag{8}$$

$$H_G^{(j)} p_s \leq b_G^{(j)} - \|H_G^{(j)}\| \ell_s, \forall\ j = 1, \ldots, \mathsf{dP}(H_G) \tag{9}$$

$$\begin{pmatrix} \left(-H_O^{(j)} p_{i-1} + (b_O^{(j)} + \|H_O^{(j)}\| \ell_i) < M(1-\alpha_j)\right) \\ \wedge \left(-H_O^{(j)} p_i + (b_O^{(j)} + \|H_o^{(j)}\| \ell_i) < M(1-\alpha_j)\right) \end{pmatrix} \tag{10}$$

$$\forall\ \mathit{Poly}(H, b) \in \mathbf{O}, j = 1, \ldots, \mathsf{dP}(H_G), i = 1, \ldots, s$$

$$\sum_{j=1}^{\mathsf{dP}(H)} \alpha_j \geq 1)) \sum_{j=1}^{n} |(p_i^{(j)} - p_{i-1}^{(j)})| > \bar{v}\mathsf{Tmin}) \tag{11}$$

Constraint (8) enforces the requirement that the first way point must be at the center of the initial set $(p_{\text{ref}}(0))$. Constraint (9) ensures that all trajectories following ξ_{ref} end in the goal set G, that is, p_s (the last waypoint) is inside the polytope $Poly(H_G, b_G)$ shrunken by the factor ℓ_s. Constraint (10) captures the requirement that for every segment defined by (p_{i-1}, p_i), both p_{i-1} and p_i must lie in the same halfspace outside the polytope $Poly(H_o, b_o)$ enlarged by the error factor ℓ_i (given by `RecTrec`). This must hold for every obstacle $O \in \mathbf{O}$. Here $\alpha_j \in \{0, 1\}$ and M is an arbitrarily large constant. Finally, constraint (11) captures the requirement that every line segment defined by (p_{i-1}, p_i) must have length at least $\bar{v}$Tmin. This is required to ensure that the bounds returned by `ErrBound` indeed bound the error over the segments. We note that fixing s, the number of constraints grow linearly with the number of surfaces in the obstacle and goal set polytopes.

Proposition 3 *Fix* $s \geq 1$ *as the number of line segments and* $p_{ref}(0) \in \mathcal{W}$ *as the initial position of the reference trajectory. Assume that:*

(1) $\mathcal{A}$ *closed with* g_r *and* g_t *is such that given any sequence of* $s+1$ *waypoints in* $\mathcal{W}$ *and any* $\bar{v}$*, the piece-wise reference* ξ_{ref} *and input* u_{ref} *are returned by Algorithm 3 satisfies the conditions in Lemmas 1 and Proposition 1 with the tracking error* $e(t)$*.*

(2) For the above ξ_{ref}, fix an ε_0 such that $e(0) \in B_{\varepsilon_0}(0)$, let $\{\ell_i\}_{i=0}^s$ be the error bounds constructed using Algorithm 2.

(3) $\phi_{waypoints}$ is satisfiable with waypoints $\{p_i\}_{i=0}^s$.

Let $\xi_{ref}(t), u_{ref}(t) = \texttt{Waypt2Traj}(\{p_i\}_{i=0}^s, \bar{v})$ and $p_{ref}(t) = \xi_{ref}(t) \downarrow p$. Let $\xi_g(t)$ be a trajectory of $\mathcal{A}$ closed with $g_t(\cdot, \xi_{ref}, u_{ref})$ starting from $\xi_g(0)$ with $e(0) \in B_{\varepsilon_0}$. Then $\xi_g(t)$ satisfies the reach avoid requirement

The function `CheckSAT` has as its inputs an initial set Θ, goal set G, unsafe set $\mathbf{O}$, bounds $\{\ell_i\}_{i=0}^s$, minimum time bound Tmin, number of line segments s, and a velcoty $\bar{v}$. It then tries to find $\{p_i\}_{i=0}^s$ that satisfies $\phi_{\text{waypoints}}$. If the problem is feasible, then it returns $(\text{SAT}, \{p_i\}_{i=0}^s\})$. If the problem is infeasible, then it returns UNSAT.

IV. Implementation and Experimentation

Our complete synthesis algorithm including all the discussed subroutines are implemented in Python and are made publicly available[4]. The reachtubes are computed using the method from [**?**] although with minor implementation changes, other tools and libraries could be used. `CheckSAT`'s MILP formulation is solved using Gurobi [**?**].

We will discuss experimental results using the kinematic car model of Example 1. The tool can be applied to other vehicle models. Here the vehicle controller g_t is the Lyapunov based tracking controller found in [**?**]. We created a number of reach-avoid scenarios to investigate the behavior of the synthesis algorithm (see Figure 2).

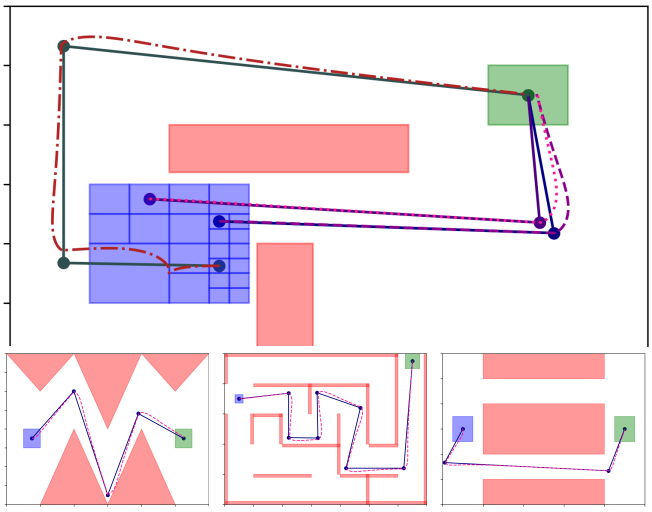

Fig. 2: Example scenarios with reach-avoid specifications. Top: Barrier (also seen in Figure 1), bottom left: Zigzag, bottom center: Maze, bottom right: Hallway. Also shown in the figures are example reference trajectories and actual trajectories, and partitions of the initial set.

In each scenario, the initial set Θ is centered at the same point. The size of the initial set is changed to explore how the execution time and number of partitions changes. Additionally, the effect of Tmin (minimum duration per segment) on the number of partitions and number of segments is also explored. Results are shown in Table I.

Scenario	Size of Θ	Tmin (s)	Execution Time (s)	Segs per ξ_{ref}	Num partitions
Barrier	0.707	0.25	2.583	2 - 4	22
	0.707	0.125	25.432	FAIL	FAIL
	0.565	0.25	0.428	4	4
	0.565	0.125	0.502	5	4
	0.353	0.25	0.0412	4	1
	0.353	0.125	0.0425	4	1
Zigzag	0.707	0.25	0.125	6	1
	0.707	0.125	32.98	FAIL	FAIL
	0.565	0.25	0.0868	5	1
	0.353	0.125	0.115	6	1
	0.353	0.25	0.172	7	1
	0.353	0.125	0.115	6	1
Maze	0.500	0.25	10.164	9	10
	0.500	0.125	88.748	FAIL	FAIL
	0.200	0.25	0.722	9	1
	0.200	0.125	0.764	9	1
Hallway	1.118	0.25	1.32	4	19
	1.118	0.125	1.126	4	19
	0.707	0.25	0.027	4	1
	0.707	0.125	0.026	4	1
	0.707	1	26.064	FAIL	FAIL

TABLE I: Results of the implementation. This table shows how the execution time, number of segments, and number of partitions change with the size of Θ and Tmin.

Key observations from these experiments are as follows. (1) The typicall running time of the synthesis algorithm on this $(3, 2)$-nonlinear control system system with $d = 2$-dimensional workspace is around a couple of seconds. This figure is promising and competitive relative to other tools[5] (2) As the minimum time bound Tmin increased, it became easier for Algorithm 4 to find a valid reference controller. That is, solutions were found with fewer partitions of Θ. This is due to the fact that the reachtubes shrink more over a larger Tmin and the corresponding tracking error bounds also shrink, which allows tighter reference tracking. If Tmin is too small, then the reachtubes do not shrink enough and the tracking error bounds become large. (3) However, we must be careful to not choose a Tmin that is too large. If Tmin is too large, it is possible that the line segments may no longer fit in the safe work space. This is what happened in the last case of the Hallway scenario. (4) The scenarios are marked FAIL when the algorithm reaches the partitioning limit without finding controllers for some of the partitions. This only happens under "extreme conditions" such as Tmin being too large or too small.

To show that using reachtubes to get tighter bounds on ℓ_i help to find reference controllers more reliably, we run our synthesis algorithm using a Lyapunov function to find ℓ_i. These results are shown in Table II. Note that using a Lyapunov function to compute the bounds requires the initial set to be partitioned more than if reachtubes are used to compute ℓ_i. The difference in execution time is significant, although this comes from the fact that searching a reachtube

[4]See the weblink given on the first page.

[5]A detailed parallel comparison of this algorithm with other approaches will be undertaken in the future. In general, direct comparisons are hard because of platform dependent constant.

for the error bounds takes more time than simply adding error at each time segment.

Method	Size of Θ	Execution Time (s)	Segs per ξ_{ref}	Num partitions
Reachtubes	0.707	2.524	2-4	22
Lyapunov	0.707	0.697	2-4	25

TABLE II: Comparison to bloating the obstacles using only the Lyapunov function vs. using reachtubes.

V. Conclusions

We presented an algorithm to synthesize reference controllers for nonlinear systems and reach-avoid requirements. The key novelty is to use locally optimal reachability analysis to compute tracking error bounds relative to *an arbitrary piece-wise continuous* reference trajectory, and then, to use these bounds for finding specific references using MILP. This algorithm was tested on various reach-avoid scenarios. The effects of changing various parameters to synthesize the controller were studied. In the future, it would be interesting to add actuation constraints to the MILP formulation and investigate approaches for handling model uncertainty.