

# Fast and Guaranteed Safe Controller Synthesis for Nonlinear Vehicle Models <sup>\*</sup>

Chuchu Fan<sup>1</sup>[0000-0003-4671-233X], Kristina Miller<sup>2</sup>[0000-0003-1016-1695], and  
Sayan Mitra<sup>2</sup>[0000-0001-7082-5516]

<sup>1</sup> [chuchu@caltech.edu](mailto:chuchu@caltech.edu)

Department of Computing and Mathematical Sciences  
California Institute of Technology.

<sup>2</sup> [kmmille2@illinois.edu](mailto:kmmille2@illinois.edu), [mitras@illinois.edu](mailto:mitras@illinois.edu)

Department of Electrical and Computer Engineering,  
University of Illinois at Urbana-Champaign.



**Abstract.** We address the problem of synthesizing a controller for nonlinear systems with reach-avoid requirements. Our controller consists of a reference controller and a tracking controller which drives the actual trajectory to follow the reference trajectory. We identify a type of reference trajectory such that the tracking error between the actual trajectory of the closed-loop system and the reference trajectory can be bounded. Moreover, such a bound on the tracking error is independent of the reference trajectory. Using such bounds on the tracking error, we propose a method that can find a reference trajectory by solving a satisfiability problem over linear constraints. Our overall algorithm guarantees that the resulting controller can make sure every trajectory from the initial set of the system satisfies the given reach-avoid requirement. We also implement our technique in a tool FACTEST. We show that FACTEST can find controllers for four vehicle models (3-6 dimensional state space and 2-4 dimensional input space) across eight scenarios (with up to 22 obstacles), all with running time at the sub-second range.

## 1 Introduction

Design automation and safety of autonomous systems is an important research area. Controller synthesis aims to provide correct-by-construction controllers that can guarantee that the system under control meets certain requirements. Controller synthesis is a type of program synthesis problem. The synthesized program or *controller*  $g$  has to meet the given requirement  $R$ , when it is run in

---

<sup>\*</sup> The authors acknowledge support from the DARPA Assured Autonomy under contract FA8750-19-C-0089, the Air Force Office of Scientific Research under grant AFOSR FA9550-17-1-0236, and the National Science Foundation under grant NSF CCF 1918531. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

(closed-loop) composition with a given physical process or *plant*  $\mathcal{A}$ . Therefore, a synthesis algorithm has to account for the combined behavior of  $g$  and  $\mathcal{A}$ .

Methods for designing controllers for asymptotic requirements like stability, robustness, and tracking, predate the algorithmic synthesis approaches for programs [16,30,3]. However, these classic control design methods normally do not provide formal guarantees in terms of handling bounded-horizon requirements like safety. Typical controller programs are small, well-structured, and at core, have a succinct logic (“bang-bang” control) or mathematical operations (PID control). This might suggest that controllers could be an attractive target for algorithmic synthesis for safety, temporal logic (TL), and bounded time requirements [38,18,9,34,1].

On the other hand, *motion planning (MP)*, which is an instance of the controller synthesis for robots is notoriously difficult (see [21] Chapter 6.5). A typical MP requirement is to make a robot  $\mathcal{A}$  track certain waypoints while meeting some constraints. A popular paradigm in MP, called sampling-based MP, gives practical, *fully automatic*, randomized, solutions to hard problem instances by only considering the geometry of the vehicle and the free space [20,15,21,14]. However, they do not ensure that the dynamic behavior of the vehicle will actually follow the planned path without running into obstacles. Ergo, MP continues to be a central problem in robotics<sup>3</sup>.

In this paper, we aim to achieve faster control synthesis with guarantees by exploiting a separation of concerns that exists in the problem: (A) how to drive a vehicle/plant to a *given waypoint*? and (B) Which *waypoints* to choose for achieving the ultimate goal? (A) can be solved using powerful control theoretic techniques—if not completely automatically, but at least in a principled fashion, with guarantees, for a broad class of  $\mathcal{A}$ ’s. Given a solution for (A), we solve (B) algorithmically. A contribution of the paper is to identify characteristics of a solution of (A) that make solutions of (B) effective. Consider nonlinear control systems  $\mathcal{A} : \frac{d}{dt}x = f(x, u)$  and reach-avoid requirements defined by a goal set  $G$  that the trajectories should reach, and obstacles  $\mathbf{O}$  the trajectories should avoid. The above separation leads to a two step process: (A) Find a state feedback tracking controller  $g_{\text{trk}}$  that drives the actual trajectory of the closed-loop system  $\xi_g$  to follow a reference trajectory  $\xi_{\text{ref}}$ . (B) Design a reference controller  $g_{\text{ref}}$ , which consists of a reference trajectory  $\xi_{\text{ref}}$  and a reference input  $u_{\text{ref}}$ . The distance between  $\xi_g$  and  $\xi_{\text{ref}}$  is called the tracking error  $e$ . If we can somehow know beforehand the value of  $e$  without knowing  $\xi_{\text{ref}}$ , we can use such error to bloat  $\mathbf{O}$  and shrink  $G$ , and then synthesize  $\xi_{\text{ref}}$  such that it is  $e$  away from the obstacles (inside the goal set). For linear systems, this was the approach used in [7], but for nonlinear systems, the tracking error  $e$  will generally change with  $\xi_{\text{ref}}$ , and the two steps get entangled.

For a general class of nonlinear vehicles (such as cars, drones, and underwater vehicles), the tracking controller  $g_{\text{trk}}$  is always designed to minimize the tracking

---

<sup>3</sup> In the most recent International Conference on Robotics and Automation, among the 3,512 submissions “Path and motion planning” was the second most popular key phrase.

error. The convergence of the error can be proved by a Lyapunov function for certain types of  $\xi_{\text{ref}}$ . We show how, under reasonable assumptions, we can use Lyapunov functions to bound the value of the tracking error *even when the waypoints changes* (Lemma 2). This error bound is independent of  $\xi_{\text{ref}}$  so long as  $\xi_{\text{ref}}$  satisfies the assumptions. For step (B) we introduce a SAT-based trajectory planning methods to find such  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$  by solving a satisfiability (SAT) problem over quantifier free linear real arithmetic (Theorem 1). Moreover, the number of constraints in the SMT problem scales linearly to the increase of number of obstacles (and not with the vehicle model). Thus, our methods can scale to complex requirements and high dimensional systems.

Putting it all together, our final synthesis algorithm (Algorithm 2) guarantees that any trajectory following the synthesized reference trajectory will satisfy the reach-avoid requirements. The resulting tool FACTEST is tested with four nonlinear vehicle models and on eight different scenarios, taken from MP literature, which cover a wide range of 2D and 3D environments. Experiment results show that our tool scales very well: it can find the small covers  $\{\Theta_j\}_j$  and the corresponding reference trajectories and control inputs satisfying the reach-avoid requirements most often in less than a second, even with up to 22 obstacles. We have also compared our SAT-based trajectory planner to a standard RRT planner, and the results show that our SAT-based method resoundingly outperforms RRT. To summarize, our main contributions are:

1. A method (Algorithm 2) for controller synthesis separating tracking controller  $g_{\text{trk}}$  and search for reference controller  $g_{\text{ref}}$ .
2. Sufficient conditions for tracking controller error performance that makes the decomposition work (Lemma 2 and Lemma 3).
3. An SMT-based effective method for synthesizing reference controller  $g_{\text{ref}}$ .
4. The FACTEST implementation of the above and its evaluation showing very encouraging results in terms of finding controllers that make any trajectories of the closed-loop system satisfy reach-avoid requirements (Section 6).

**Related works** *Model Predictive Control (MPC)*. MPC [25,4,45,49] has to solve a constrained, discrete-time, optimal control problem. MPC for controller synthesis typically requires model reduction for casting the optimization problem as an LP [4], QP [2,36], MILP [45,33,34]. However, when the plant model is nonlinear [8,22], it may be hard to balance speed and complex requirements as the optimization problem become nonconvex and nonlinear.

*Discrete abstractions*. Discrete, finite-state, abstraction of the control system is computed, and then a discrete controller is synthesized by solving a two-player game [42,10,24,17,47]. CoSyMA [28], Pessoa [37], LTLMop [46,18], Tulip [48,9], and SCOTS [38] are based on these approaches. The discretization step often leads to a severe state space explosion for higher dimensional models.

*Safe motion planning*. The idea of bounding the tracking error through pre-computation has been used in several techniques: FastTrack [11] uses Hamilton-Jacobi reachability analysis to produce a “safety bubble” around planned paths. Reachability based trajectory design for dynamical environments (RTD) [44]

computes an offline forward reachable sets to guarantee that the robot is not-at-fault in any collision. In [40], a technique based on convex optimization is used to compute tracking error bounds. Another technique [43,23] uses motion primitives expanded by safety funnels, which defines similar ideas of safety tubes.

*Sampling based planning.* Probabilistic Road Maps (PRM) [15], Rapidly-exploring Random Trees (RRT) [19], and fast marching tree (FMT) [12] are widely used in actual robotic platforms. They can generate feasible trajectories through known or partially known environments. Compared with the deterministic guarantees provided by our proposed method, these methods come with stochastic guarantees. Also, they are not designed to be robust to model uncertainty or disturbances. MoveIT [5] is a tool designed to implement and benchmark various motion planners on robots. The motion planners in MoveIT are from the open motion planning library (OMPL) [41], which implements motion planners abstractly.

*Controlled Lyapunov function (CLF).* CLF have been used to guarantee that the overall closed-loop controlled system satisfies a reach-while-stay specification [35]. Instead of asking for a CLF for the overall closed-loop system, our method only needs a Lyapunov function for the tracking error, which is a weaker local requirement. CLF is often a difficult requirement to meet for nonlinear vehicle models.

## 2 Preliminaries and problem Statement

Let us denote real numbers by  $\mathbb{R}$ , non-negative real numbers by  $\mathbb{R}_{\geq 0}$ , and natural numbers by  $\mathbb{N}$ . The  $n$ -dimensional *Euclidean space* is  $\mathbb{R}^n$ . For a vector  $x \in \mathbb{R}^n$ ,  $x^{(i)}$  is the  $i^{\text{th}}$  entry of  $x$  and  $\|x\|_2$  is the 2-norm of  $x$ . For any matrix  $A \in \mathbb{R}^{n \times m}$ ,  $A^\top$  is its *transpose*;  $A^{(i)}$  is the  $i^{\text{th}}$  row of  $A$ . Given a  $r \geq 0$ , an  $r$ -ball around  $x \in \mathbb{R}^n$  is defined as  $B_r(x) = \{x' \in \mathbb{R}^n \mid \|x' - x\|_2 \leq r\}$ . We call  $r$  the radius of the ball. Given a matrix  $H \in \mathbb{R}^{r \times n}$  and a vector  $b \in \mathbb{R}^r$ , an  $(H, b)$ -polytope is denoted by  $\text{Poly}(H, b) = \{x \in \mathbb{R}^n \mid Hx \leq b\}$ . Each row of the inequality  $H^{(i)}x \leq b^{(i)}$  defines a *halfspace*. We also call  $H^{(i)}x = b^{(i)}$  the *surface* of the polytope. Let  $\text{dP}(H) = r$  denotes the number of rows in  $H$ . Given a set  $S \subseteq \mathbb{R}^n$ , the radius of  $S$  is defined as  $\sup_{x, y \in S} \|x - y\|_2 / 2$ .

*State space and workspace.* The state space of control systems will be a subspace  $\mathcal{X} \subseteq \mathbb{R}^n$ . The *workspace* is a subspace  $\mathcal{W} \subseteq \mathbb{R}^d$ , for  $d \in \{2, 3\}$ , 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,  $x \downarrow p = [p_x, p_y]^\top \in \mathbb{R}^2$  for ground vehicles on the plane and  $x \downarrow p = [p_x, p_y, p_z]^\top \in \mathbb{R}^3$  for aerial and underwater vehicles. When  $x$  is clear from context we will write  $x \downarrow p$  as simply  $p$ . The vector  $x$  may include other variables like velocity, heading, pitch, etc., but  $p$  only has the position in Cartesian coordinates. We assume that the goal set  $G := \text{Poly}(H_G, b_G)$  and the unsafe set  $\mathbf{O}$  (obstacles) are specified by polytopes in  $\mathcal{W}$ ;  $\mathbf{O} = \cup O_i$ , where  $O_i := \text{Poly}(H_{O_i}, b_{O_i})$  for each obstacle  $i$ .

*Trajectories and reach-avoid requirements.* A trajectory  $\xi$  over  $\mathcal{X}$  of duration  $T$  is a function  $\xi : [0, T] \rightarrow \mathcal{X}$ , that maps each time  $t$  in the time domain  $[0, T]$  to a point  $\xi(t) \in \mathcal{X}$ . The *time bound or duration* of  $\xi$  is denoted by  $\xi.\text{ltime} = T$ . The projection of a trajectory  $\xi : [0, T] \rightarrow \mathcal{X}$  to  $\mathcal{W}$  is written as  $\xi \downarrow p : [0, T] \rightarrow \mathcal{W}$  and defined as  $(\xi \downarrow p)(t) = \xi(t) \downarrow p$ . We say that a trajectory  $\xi(t)$  *satisfies a reach-avoid requirement given by unsafe set  $\mathbf{O}$  and goal set  $G$*  if  $\forall t \in [0, \xi.\text{ltime}], \xi(t) \downarrow p \notin \mathbf{O}$  and  $\xi(\xi.\text{ltime}) \downarrow p \in G$ . See Figure 1 for an example.

Given a trajectory  $\xi : [0, T] \rightarrow \mathcal{X}$  and a time  $t > 0$ , the *time shift* of  $\xi$  is a function  $(\xi + t) : [t, t + T] \rightarrow \mathcal{X}$  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  $\xi_1$  is followed by  $\xi_2$ . That is, for each  $t \in [0, \xi_1.\text{ltime} + \xi_2.\text{ltime}]$ ,  $(\xi_1 \frown \xi_2)(t) = \xi_1(t)$  when  $t \leq \xi_1.\text{ltime}$ , and equals  $\xi_2(t - \xi_1.\text{ltime})$  when  $t > \xi_1.\text{ltime}$ . Trajectories are closed under concatenation, and many trajectories can be concatenated in the same way.

## 2.1 Nonlinear control system

**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)  $f : \mathcal{X} \times \mathbf{U} \rightarrow \mathcal{X}$  is the dynamic function that is Lipschitz continuous with respect to the first argument.

A control system with no inputs ( $m = 0$ ) is called a *closed* system.

Let us fix a time duration  $T > 0$ . An *input trajectory*  $u : [0, T] \rightarrow \mathbf{U}$ , is a continuous trajectory over the input space  $\mathbf{U}$ . We denote the set of all possible input trajectories to be  $\mathcal{U}$ . Given an input signal  $u \in \mathcal{U}$  and an initial state  $x_0 \in \Theta$ , a *solution* of  $\mathcal{A}$  is a continuous trajectory  $\xi_u : [0, T] \rightarrow \mathcal{X}$  that satisfies (i)  $\xi_u(0) = x_0$  and (ii) for any  $t \in [0, T]$ , the time derivative of  $\xi_u$  at  $t$  satisfies the differential equation:

$$\frac{d}{dt}\xi_u(t) = f(\xi_u(t), u(t)). \quad (1)$$

For any  $x_0 \in \Theta, u \in \mathcal{U}$ ,  $\xi_u$  is a state trajectory and we call such a pair  $(\xi_u, u)$  a state-input trajectory pair.

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  $\xi_{\text{ref}}$ . Similarly, a *reference input trajectory* (or *reference input*) is a trajectory over  $\mathbf{U}$  and we denote them as  $u_{\text{ref}}$ . Note these  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$  are not necessarily solutions of (1). Figure 1 shows reference and actual solution trajectories.

We call a reference trajectory  $\xi_{\text{ref}}$  and a reference input  $u_{\text{ref}}$  together as a reference controller  $g_{\text{ref}}$ . Given  $g_{\text{ref}}$ , a *tracking controller*  $g_{\text{trk}}$  is a function that is used to compute the inputs for  $\mathcal{A}$  so that in the resulting closed system, the state trajectories try to follow  $\xi_{\text{ref}}$ .

**Definition 2.** Given an  $(n, m)$ -dynamical system  $\mathcal{A}$ , a reference trajectory  $\xi_{\text{ref}}$ , and a reference input  $u_{\text{ref}}$ , a tracking controller for the triple  $\langle \mathcal{A}, \xi_{\text{ref}}, u_{\text{ref}} \rangle$  is a (state feedback) function  $g_{\text{trk}} : \mathcal{X} \times \mathcal{X} \times \mathbf{U} \rightarrow \mathbf{U}$ .

At any time  $t$ , the tracking controller  $g_{\text{trk}}$  takes in a current state of the system  $x$ , a reference trajectory state  $\xi_{\text{ref}}(t)$ , and a reference input  $u_{\text{ref}}(t)$ , and gives an input  $g_{\text{trk}}(x, \xi_{\text{ref}}(t), u_{\text{ref}}(t)) \in \mathbf{U}$  for  $\mathcal{A}$ . The controller  $g$  for  $\mathcal{A}$  is determined by both the reference controller  $g_{\text{ref}}$  and the tracking controller  $g_{\text{trk}}$ . The resulting trajectory  $\xi_g$  of the closed control system ( $\mathcal{A}$  closed with  $g_{\text{ref}}$  and  $g_{\text{trk}}$ ) satisfies:

$$\frac{d}{dt}\xi_g(t) = f(\xi_g(t), g_{\text{trk}}(\xi_g(t), \xi_{\text{ref}}(t), u_{\text{ref}}(t))), \forall t \in [0, T] \setminus D, \quad (2)$$

where  $D$  is the set of points in time where the second or third argument of  $g_{\text{trk}}$  is discontinuous<sup>4</sup>.

## 2.2 Controller synthesis problem

**Definition 3.** Given a  $(n, m)$ -dimensional nonlinear system  $\mathcal{A} = \langle \mathcal{X}, \Theta, \mathbf{U}, f \rangle$ , its workspace  $\mathcal{W}$ , goal set  $G \subseteq \mathcal{W}$  and the unsafe set  $\mathbf{O} \subseteq \mathcal{W}$ , we are required to find (a) a tracking controller  $g_{\text{trk}}$ , (b) a partition  $\{\Theta_j\}_j$  of  $\Theta$ , and (c) for each partition  $\Theta_j$ , a reference controller  $g_{j,\text{ref}}$ , which consists of a state trajectory  $\xi_{j,\text{ref}}$  and an input trajectory  $u_{j,\text{ref}}$ , such that  $\forall x_0 \in \Theta_j$ , the unique trajectory  $\xi_g$  of the closed system as in Equation (2) starting from  $x_0$  reaches  $G$  and avoids  $\mathbf{O}$ .

Again,  $\xi_{j,\text{ref}}$  and  $u_{j,\text{ref}}$  in  $g_{j,\text{ref}}$  are not required to be a state-input pair, but, for each initial state  $x_0 \in \Theta_j$ , the closed loop trajectory  $\xi_g$  following  $\xi_{\text{ref}}$  is a valid state trajectory with corresponding input  $u$  generated by  $g_{\text{trk}}$  and  $g_{j,\text{ref}}$ . In this paper, we will decompose the controller synthesis problem: Part (a) will be delivered by design engineers with knowledge of vehicle dynamics, and parts (b) and (c) will be automatically synthesized by our algorithm. The latter being the main contribution of the paper.

*Example 1.* Consider a ground vehicle moving on a 2D workspace  $\mathcal{W} \subseteq \mathbb{R}^2$  as shown in Figure 1.

This scenario is called **Zigzag** and it is adopted from [32]. The red polytopes are obstacles. The blue and green polytopes are the initial set  $\Theta$  and the goal set  $G$ . There are also obstacles (not shown in the figure) defining the boundaries of the entire workspace. The black line is a projection of a reference trajectory to the workspace:  $\xi_{\text{ref}}(t) \downarrow p$ . This would not be a feasible state trajectory for a ground vehicle that cannot make sharp turns. The purple dashed curve is a real feasible state trajectory of the system starting from  $\Theta$  with a tracking controller  $g_{\text{trk}}$ , where  $g_{\text{trk}}$  will be introduced in Example 2.

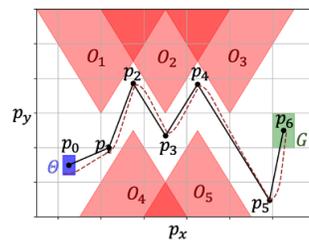


Fig. 1: Zigzag scenario for a controller synthesis problem. The initial set is blue, the goal set is green, and the unsafe sets are red. A valid reference trajectory is shown in black and a feasible trajectory is shown in purple.

<sup>4</sup>  $\xi_g$  is a standard solution of ODE with piece-wise continuous right hand side.

Consider the standard nonlinear bicycle model of a car [31]. The control system has 3 state variables: the position  $p_x, p_y$ , and the heading direction  $\theta$ . Its motion is controlled by two inputs: linear velocity  $v$  and rotational velocity  $\omega$ . The car's dynamics are given by:

$$\frac{d}{dt}p_x = v \cos(\theta), \frac{d}{dt}p_y = v \sin(\theta), \frac{d}{dt}\theta = \omega. \quad (3)$$

### 3 Constructing reference trajectories from waypoints

If  $\xi_{\text{ref}}(t) \downarrow p$  is a PWL (PWL) curve in the workspace  $\mathcal{W}$ , we call  $\xi_{\text{ref}}(t)$  a PWL reference trajectory. In  $\mathcal{W}$ , a PWL curve can be determined by the endpoints of each line segment. We call such endpoints the *waypoints* of the PWL reference trajectory. In Figure 1, the black points  $p_0, \dots, p_6$  are waypoints of  $p(t) = \xi_{\text{ref}}(t) \downarrow p$ .

Consider any vehicle on the plane<sup>5</sup> with state variables  $p_x, p_y, \theta, v$  ( $x$ -position,  $y$ -position, heading direction, linear velocity) and input variables  $a, \omega$  (acceleration and angular velocity). Once the waypoints  $\{p_i\}_{i=0}^k$  are fixed, and if we enforce constant speed  $\bar{v}$  (i.e.,  $\xi_{\text{ref}}(t) \downarrow v = \bar{v}$  for all  $t \in [0, \xi_{\text{ref}}.\text{time}]$ ), then  $\xi_{\text{ref}}(t)$  can be uniquely defined by  $\{p_i\}_{i=0}^k$  and  $\bar{v}$  using Algorithm 1. The semantics of  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$  returned by `Waypoints_to_Traj` is that the reference trajectory requires the vehicle to move at a constant speed  $\bar{v}$  along the lines connecting the waypoints  $\{p_i\}_{i=0}^k$ . In Example 1,  $\xi_{\text{ref}}(t), u_{\text{ref}}(t)$  can also be constructed using `Waypoints_to_Traj` moving  $v$  to input variables and dropping  $a$ .

We notice that if  $k = 1$ ,  $\xi_{\text{ref}}(t), u_{\text{ref}}(t)$  returned by Algorithm 1 is a valid state-input trajectory pair. However, if  $k > 1$ ,  $\xi_{\text{ref}}(t), u_{\text{ref}}(t)$  returned by Algorithm 1 is usually not a valid state-input trajectory pair. This is because  $\theta_{\text{ref}}(t)$  is discontinuous at the waypoints and no bounded inputs  $u_{\text{ref}}(t)$  can drive the vehicle to achieve such  $\theta_{\text{ref}}(t)$ . Therefore, when  $k > 1$ ,  $\xi_{\text{ref}}(t)$  is a PWL reference trajectory with no  $u_{\text{ref}}(t)$  such that  $\xi_{\text{ref}}, u_{\text{ref}}$  are solutions of (1).

---

#### Algorithm 1: `Waypoints_to_Traj`( $\{p_i\}_{i=0}^k, \bar{v}$ )

---

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

- 1  $\forall t \in [0, \sum_{i=1}^k \frac{\|p_j - p_{j-1}\|_2}{\bar{v}}], u_{\text{ref}}(t) = \bar{v}, a_{\text{ref}}(t) = 0, \omega_{\text{ref}}(t) = 0;$
- 2  $\forall i \geq 1, \forall t \in [\sum_{j=1}^{i-1} \frac{\|p_j - p_{j-1}\|_2}{\bar{v}}, \sum_{j=1}^i \frac{\|p_j - p_{j-1}\|_2}{\bar{v}}),$   
 $p_{\text{ref}}(t) = p_{i-1} + \bar{v}t - \sum_{j=1}^{i-1} \|p_j - p_{j-1}\|_2,$   
 $\theta_{\text{ref}}(t) = \text{mod}(\text{atan2}((p_{y,i} - p_{y,i-1}), (p_{x,i} - p_{x,i-1})), 2\pi);$
- 3  $\xi_{\text{ref}}(t) = [p_{\text{ref}}(t), \theta_{\text{ref}}(t), v_{\text{ref}}(t)];$
- 4  $u_{\text{ref}}(t) = [a_{\text{ref}}(t), \omega_{\text{ref}}(t)];$
- 5 **return**  $\xi_{\text{ref}}(t), u_{\text{ref}}(t);$

---

<sup>5</sup> A similar construction works for vehicles in 3D workspaces with additional variables.

**Proposition 1.** *Given a sequence of waypoints  $\{p_i\}_{i=0}^k$  and a constant speed  $\bar{v}$ ,  $\xi_{\text{ref}}(t), u_{\text{ref}}(t)$  produced by `Waypoints_to_Traj`( $\{p_i\}_{i=0}^k, \bar{v}$ ) satisfy:*

- $p_{\text{ref}}(t) = \xi_{\text{ref}}(t) \downarrow p$  is a piece-wise continuous function connecting  $\{p_i\}_{i=0}^k$ .
- At time  $t_i = \sum_{j=1}^i \|p_j - p_{j-1}\|_2 / \bar{v}$ ,  $p_{\text{ref}}(t_i) = p_i$ . We call  $\{t_i\}_{i=1}^k$  the concatenation time.
- $\xi_{\text{ref}}(t) = \xi_{\text{ref},1}(t) \frown \dots \frown \xi_{\text{ref},k}(t)$  and  $u_{\text{ref}}(t) = u_{\text{ref},1}(t) \frown \dots \frown u_{\text{ref},k}(t)$ , where  $(\xi_{\text{ref},i}, u_{\text{ref},i})$  are state-input trajectory pairs returned by the function `Waypoints_to_Traj`( $\{p_{i-1}, p_i\}, \bar{v}$ )

**Outline of synthesis approach** In this Section, we present an Algorithm `Waypoints_to_Traj` for constructing reference trajectories from arbitrary sequence of waypoints. In Section 4, we precisely characterize the type of vehicle tracking controller our method requires from designers. On our tool’s web-page [27], we show with several extra examples that indeed developing such controllers is non-trivial, far from automatic, yet bread and butter of control engineers. In Section 5, we present the main synthesis algorithm, which uses the tracking error bounds from the previous section, to construct waypoints, for each initial state, which when passed through `Waypoints_to_Traj` provide the solutions to the synthesis problem.

## 4 Bounding the error of a tracking controller

### 4.1 Tracking error and Lyapunov functions

Given a reference controller  $g_{\text{ref}}$ , a tracking controller  $g_{\text{trk}}$ , and an initial state  $x_0 \in \Theta$ , the resulting trajectory  $\xi_g$  of the closed control system ( $\mathcal{A}$  closed with  $g_{\text{ref}}$  and  $g_{\text{trk}}$ ) is a state trajectory that starts from  $x_0$  and follows the ODE (2). In this setting, we define the *tracking error* at time  $t$  to be a continuous function:

$$e : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^n.$$

When  $\xi_g(t)$  and  $\xi_{\text{ref}}(t)$  are fixed, we also 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. To start with, let us assume that  $g_{\text{ref}} = (\xi_{\text{ref}}, u_{\text{ref}})$  is a valid state-input pair so  $\xi_{\text{ref}}$  is a continuous state trajectory. Later we will see that the analysis can be extended to cases when  $\xi_{\text{ref}}$  is discontinuous but a concatenation of continuous state trajectories.

When  $(\xi_{\text{ref}}, u_{\text{ref}})$  is a valid state-input pair and  $e(t)$  satisfy an differential equation  $\frac{d}{dt}e(t) = f_e(e(t))$ , we use Lyapunov functions, which is a classic technique for proving stability of an equilibrium of an ODE, to bound the tracking error  $e(t)$ . The Lie derivative  $\frac{\partial V}{\partial e} f_e(e)$  below captures the rate of change of the function  $V$  along the trajectories of  $e(t)$ .

**Definition 4 (Lyapunov functions [16]).** *Fix a state-input reference trajectory pair  $(\xi_{\text{ref}}, u_{\text{ref}})$ , assume that the dynamics of the tracking error  $e$  for a*

closed control system  $\mathcal{A}$  with  $g_{\text{ref}}$  and  $g_{\text{trk}}$  can be rewritten as  $\frac{d}{dt}e(t) = f_e(e(t))$ , where  $f_e(0) = 0$ . A continuously differentiable function  $V : \mathbb{R}^n \rightarrow \mathbb{R}$  satisfying (i)  $V(0) = 0$ , (ii)  $\forall e \in \mathbb{R}^n, V(e) \geq 0$ , and (iii)  $\forall e \in \mathbb{R}^n, \frac{\partial V}{\partial e} f_e(e) \leq 0$ , is called a Lyapunov function for the tracking error.

*Example 2.* For the car of Example 1, with a continuous reference trajectory  $\xi_{\text{ref}}(t) = [x_{\text{ref}}(t), y_{\text{ref}}(t), \theta_{\text{ref}}(t)]^\top$ , we define the tracking error in a coordinate frame fixed to the car [13]:

$$\begin{pmatrix} e_x(t) \\ e_y(t) \\ e_\theta(t) \end{pmatrix} = \begin{pmatrix} \cos(\theta(t)) & \sin(\theta(t)) & 0 \\ -\sin(\theta(t)) & \cos(\theta(t)) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{\text{ref}}(t) - p_x(t) \\ y_{\text{ref}}(t) - p_y(t) \\ \theta_{\text{ref}}(t) - \theta(t) \end{pmatrix}. \quad (4)$$

With the reference controller function  $g$  defined as:

$$\begin{aligned} v(t) &= v_{\text{ref}}(t) \cos(e_\theta(t)) + k_1 e_x(t), \\ \omega(t) &= \omega_{\text{ref}}(t) + v_{\text{ref}}(t)(k_2 e_y(t) + k_3 \sin(e_\theta(t))), \end{aligned} \quad (5)$$

it has been shown in [13] when  $k_1, k_2, k_3 > 0$ ,  $\frac{d}{dt}\omega_{\text{ref}}(t) = 0$ , and  $\frac{d}{dt}v_{\text{ref}}(t) = 0$ ,

$$V([e_x, e_y, e_\theta]^\top) = \frac{1}{2}(e_x^2 + e_y^2) + \frac{1 - \cos(e_\theta)}{k_2} \quad (6)$$

is a Lyapunov function with negative semi-definite time derivative  $\frac{\partial V}{\partial x} f_e = -k_1 e_x^2 - \frac{v_{\text{ref}} k_3 \sin^2(e_\theta)}{k_2}$ .

## 4.2 Bounding tracking error using Lyapunov functions: Part 1

Consider a given closed control system,  $\mathcal{A}$  with  $g_{\text{ref}}$  and  $g_{\text{trk}}$ , in this section, we will derive upper bounds on the tracking error  $e$ . Later in Section 5, we will develop techniques that take the tracking error into consideration for computing reference trajectories  $\xi_{\text{ref}}$ .

To begin with, we consider state-input reference trajectory pairs  $(\xi_{\text{ref}}, u_{\text{ref}})$  where  $u_{\text{ref}}$  is continuous, and therefore,  $\xi_{\text{ref}}$  and  $\xi_g$  are differentiable. Let us assume that the tracking error dynamics ( $\frac{d}{dt}e(t) = f_e(e(t))$ ) has a Lyapunov function  $V(e(t))$ . The following is a standard result that follows from the theory of Lyapunov functions for dynamical systems.

**Lemma 1.** *Consider any state-input trajectory pair  $(\xi_{\text{ref}}, u_{\text{ref}})$ , an initial state  $x_0$ , the corresponding trajectory  $\xi_g$  of the closed control system, and a constant  $\ell > 0$ . If the tracking error  $e(t)$  has a Lyapunov function  $V$ , and if initially  $V(e(0)) \leq \ell$ , then for any  $t \in [0, \xi_{\text{ref}}.\text{itime}]$ ,  $V(e(t)) \leq \ell$ .*

This lemma is proved by showing that  $V(e(t)) = V(e(0)) + \int_0^t \frac{d}{dt} V(e(\tau)) d\tau \leq V(e(0))$ . The last inequality holds since  $\frac{d}{dt} V(e(\tau)) = \frac{\partial V}{\partial e} f_e(e) \leq 0$  for any  $\tau \in [0, t]$  according the definition of Lyapunov functions (Definition 4).

Lemma 1 says that if we can bound  $V(e(0)) = V(e(x_0, \xi_{\text{ref}}(0)))$ , we can bound  $V(e(\xi_g(t), \xi_{\text{ref}}(t)))$  at any time  $t$  within the domain of the trajectories,

regardless of the value of  $\xi_{\text{ref}}(t)$ . This could decouple the problem of designing the tracking controller  $g_{\text{trk}}$  and synthesizing the reference controller  $g_{\text{ref}}$  as a state-input trajectory pair  $(\xi_{\text{ref}}, u_{\text{ref}})$ .

*Example 3.* Given two waypoints  $p_0, p_1$  for the car in Example 1, take the returned value of `Waypoints_to_Traj` ( $\{p_0, p_1\}, \bar{v}$ ), move  $v_{\text{ref}}$  to  $u_{\text{ref}}$  and drop  $a_{\text{ref}}$ . Then, the resulting  $(\xi_{\text{ref}}, u_{\text{ref}})$  is a continuous and differentiable state-input reference trajectory pair. Moreover, if the robot is controlled by the tracking controller as in Equation (5),  $V(e(t)) = \frac{1}{2}(e_x(t)^2 + e_y(t)^2) + \frac{1 - \cos(e_\theta(t))}{k_2}$  is a Lyapunov function for the corresponding tracking error  $e(t) = [e_x(t), e_y(t), e_\theta(t)]^\top$ .

From Equation (4), it is easy to check that  $e_x^2(t) + e_y^2(t) = (x_{\text{ref}}(t) - p_x(t))^2 + (y_{\text{ref}}(t) - p_y(t))^2$  for any time  $t$ . Assume that initially the position of the vehicle satisfies  $[p_x(0), p_y(0)]^\top \in B_\ell([x_{\text{ref}}(0), y_{\text{ref}}(0)]^\top)$ . We check that  $V(e(0)) = \frac{1}{2}(e_x(0)^2 + e_y(0)^2) + \frac{1 - \cos(e_\theta(0))}{k_2} \leq \frac{\ell^2}{2} + \frac{2}{k_2}$ .

From Lemma 1, we know that  $\forall t \in [0, \xi_{\text{ref}}.\text{ltime}]$ ,  $V(e(t)) \leq \frac{\ell^2}{2} + \frac{2}{k_2}$ . Then we have  $(x_{\text{ref}}(t) - p_x(t))^2 + (y_{\text{ref}}(t) - p_y(t))^2 = (e_x(t)^2 + e_y(t)^2) \leq \ell^2 + \frac{4}{k_2}$ . That is, the position of the robot at time  $t$  satisfies  $[p_x(t), p_y(t)]^\top \in B_{\sqrt{\ell^2 + \frac{4}{k_2}}}([x_{\text{ref}}(t), y_{\text{ref}}(t)]^\top)$ .

### 4.3 Bounding tracking error using Lyapunov functions: Part 2

Next, let us consider the case where  $\xi_{\text{ref}}$  is discontinuous. Furthermore, let us assume that it is a concatenation of several continuous state trajectories  $\xi_{\text{ref},1} \frown \dots \frown \xi_{\text{ref},k}$ . In this case, we call  $\xi_{\text{ref}}$  a piece-wise reference trajectory. If we have a sequence of  $(\xi_{\text{ref},i}, u_{\text{ref},i})$ , each is a valid state-input trajectory pair and the corresponding error  $e_i(t)$  has a Lyapunov function  $V_i(e_i(t))$ , then we can use Lemma 1 to bound the error of  $e_i(t)$  if we know the value of  $e_i(0)$ . However, the main challenge to glue these error bounds together is that  $e(t)$  would be discontinuous with respect to the entire piece-wise  $\xi_{\text{ref}}(t)$ .

Without loss of generality, let us assume that the Lyapunov functions  $V_i(e_i(t))$  share the same format. That is,  $\forall i, V_i(e_i(t)) = V(e_i(t))$ . Let  $t_i$  be the concatenation time points when  $\xi_{\text{ref}}(t)$  (and therefore  $e(t)$ ) is discontinuous. We know that  $\lim_{t \rightarrow t_i^-} V(e(t)) \neq \lim_{t \rightarrow t_i^+} V(e(t))$  since  $\lim_{t \rightarrow t_i^-} e(t) \neq \lim_{t \rightarrow t_i^+} e(t)$ .

One insight we can get from Example 3 is that although  $e(t)$  is discontinuous at time  $t_i$ s, some of the variables influencing  $e(t)$  are continuous. For example,  $e_x(t)$  and  $e_y(t)$  in Example 3, which represent the error of the positions, are continuous since both the actual and reference positions of the vehicle are continuous. If we can further bound the term in  $V(e(t))$  that corresponds to the *other* variables, we could analyze the error bound for the entire piece-wise reference trajectory. With this in sight, 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.

Let us further assume that the Lyapunov function can be written in the form of  $V(e(t)) = \alpha(e_p(t)) + \beta(e_r(t))$ . Indeed, on the tool's webpage [27] we show that four commonly used vehicle models (car, robot, underwater vehicle, and hovercraft) have Lyapunov functions for the tracking error  $e(t)$  of this form. If

$\beta(e_r(t))$  can be further bounded, then the tracking error for the entire trajectory can be bounded using the following lemma.

**Lemma 2.** Consider  $\xi_{\text{ref}} = \xi_{\text{ref},1} \frown \cdots \frown \xi_{\text{ref},k}$ , and  $u_{\text{ref}} = u_{\text{ref},1} \frown \cdots \frown u_{\text{ref},k}$  as a piecewise reference and input with each  $(\xi_{\text{ref},i}, u_{\text{ref},i})$  being a state-input trajectory pair. Suppose (1)  $V(e(t)) = \alpha(e_p(t)) + \beta(e_r(t))$  be a Lyapunov function for the tracking error  $e(t)$  of each piece  $(\xi_{\text{ref},i}, u_{\text{ref},i})$ ; (2)  $e_p(t)$  is continuous and  $\alpha(\cdot)$  is a continuous function; (3)  $\beta(e_r(t)) \in [b_l, b_u]$ , and (4)  $V(e(0)) \leq \varepsilon_0$ . Then, the tracking error  $e(t)$  with respect to  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$  can be bounded by,

$$V(e(t)) \leq \varepsilon_i, \forall i \geq 1, \forall t \in [t_{i-1}, t_i],$$

where  $\forall i > 1, \varepsilon_i = \varepsilon_{i-1} - b_l + b_u$ ,  $\varepsilon_1 = \varepsilon_0$  being the bound on the initial tracking error, and  $t_i$ 's are the time points of concatenation<sup>6</sup>.

*Proof.* We prove this by induction on  $i$ . When  $i = 1$ , we know from Lemma 1 that if the initial tracking error is bounded by  $V(e(0))$ , then for any  $t \in [0, t_1]$ ,  $V(e(t)) \leq V(e(0)) \leq \varepsilon_0 = \varepsilon_1$ , so the lemma holds.

Fix any  $i \geq 1$ . If  $V(e(t_{i-1})) \leq \varepsilon_i$ , from Lemma 1 we have  $\forall t \in [t_{i-1}, t_i]$ ,  $V(e(t)) \leq \varepsilon_i$ . Also,  $\lim_{t \rightarrow t_i^-} V(e(t)) = \lim_{t \rightarrow t_i^-} \alpha(e_p(t)) + \beta(e_r(t)) \leq \varepsilon_i$ . Since  $\forall e_r(t) \in \mathbb{R}^{n-d}, \beta(e_r(t)) \in [b_l, b_u]$ , we have  $\lim_{t \rightarrow t_i^-} \alpha(e_p(t)) \leq \varepsilon_i - b_l$ , and  $\lim_{t \rightarrow t_i^-} \alpha(e_p(t)) = \lim_{t \rightarrow t_i^+} \alpha(e_p(t))$ . Therefore,

$$\varepsilon_{i+1} = \lim_{t \rightarrow t_i^+} V(e(t)) = \lim_{t \rightarrow t_i^+} \alpha(e_p(t)) + \beta(e_r(t)) \leq \varepsilon_i - b_l + b_u.$$

Another observation we have on the four vehicle models used in this paper is that not only  $V(e(t))$  can be written as  $\alpha(e_p(t)) + \beta(e_r(t))$  with  $\beta(e_r(t))$  being bounded, but also  $\alpha(e_p(t))$  can be written as  $\alpha(e_p(t)) = ce_p^T(t)e_p(t) = c\|p(t) - p_{\text{ref}}(t)\|_2^2$ , where  $c \in \mathbb{R}$  is a scalar constant;  $p(t) = \xi_g(t) \downarrow p$  and  $p_{\text{ref}}(t) = \xi_{\text{ref}}(t) \downarrow p$  are the actual position and reference position of the vehicle. In this case, we can further bound the position of the vehicle  $p(t)$ .

**Lemma 3.** In addition to the assumptions of Lemma 2, if  $\alpha(e_p(t)) = ce_p^T(t)e_p(t) = c\|p(t) - p_{\text{ref}}(t)\|_2^2$ , where  $c \in \mathbb{R}$ ,  $p(t) = \xi_g(t) \downarrow p$  and  $p_{\text{ref}}(t) = \xi_{\text{ref}}(t) \downarrow p$ . Then we have that at time  $t \in [t_{i-1}, t_i]$ ,

$$e_p^T(t)e_p(t) \leq \frac{\varepsilon_i - b_l}{c},$$

where  $\varepsilon_i$  and  $b_l$  are from Lemma 2, which implies that

$$p(t) \in B_{\ell_i}(p_{\text{ref}}(t)), \text{ with } \ell_i = \sqrt{\frac{\varepsilon_i - b_l}{c}}.$$

Note that Lemma 2 and 3 does not depend on the concrete values of  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$ . The lemmas hold for any piece-wise reference trajectory  $\xi_{\text{ref}}$  and reference input  $u_{\text{ref}}$  as long as the corresponding error  $e$  has a Lyapunov function (for each piece of  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$ ).

*Example 4.* Continue Example 3.

<sup>6</sup>  $\forall t \in [t_{i-1}, t_i], \xi_{\text{ref}}(t) = \xi_{\text{ref},i}(t - \sum_{j=1}^{i-1} \xi_{\text{ref},j}.\text{itime})$ .

Now let us consider the case of a sequence of waypoints  $\{p_i\}_{i=0}^k$ . Let  $(\xi_{\text{ref}}, u_{\text{ref}}) = \text{Waypoints\_to\_Traj}(\{p_i\}_{i=0}^k, \bar{v})$ . From Example 3, we know that  $V(e(t)) = \frac{1}{2}(e_x(t)^2 + e_y(t)^2) + \frac{1 - \cos(e_\theta(t))}{k_2}$  is a Lyapunov function for each segment of the piece-wise reference trajectory  $\xi_{\text{ref}}(t)$ . We also know that for any value of  $e_\theta$ , the term  $\frac{1 - \cos(e_\theta(t))}{k_2} \in [0, \frac{2}{k}]$ . From Lemma 2, we have that for  $t \in [t_{i-1}, t_i]$  where  $t_i$  are the concatenation time points, we have  $V(e(t)) \leq V(e(0)) + \frac{2(i-1)}{k_2}$ . Therefore, following Example 3, initially  $V(e(0)) \leq \frac{\ell^2}{2} + \frac{2}{k_2}$ . Then  $\forall t \in [t_{i-1}, t_i]$ ,  $V(e(t)) \leq \frac{\ell^2}{2} + \frac{2i}{k_2}$ , and the position of the robot satisfies  $[p_x(t), p_y(t)]^\top \in B_{\sqrt{\ell^2 + \frac{4i}{k_2}}}([x_{\text{ref}}(t), y_{\text{ref}}(t)]^\top)$ .

As seen in Figure 2, we bloat the black reference trajectory  $p_{\text{ref}}(t) = \xi_{\text{ref}}(t) \downarrow p$  by  $\ell_i = \sqrt{\ell^2 + \frac{4i}{k_2}}$  for the  $i^{\text{th}}$  line segment, the bloated tube contains the real position trajectories (purple curves)  $p(t)$  of the closed system.

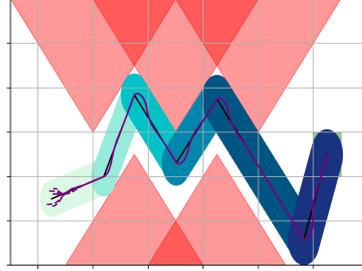


Fig. 2: Illustration of the error bounds computed from Lemma 3. The  $i^{\text{th}}$  line segment is bloated by  $\sqrt{\ell^2 + \frac{4i}{k_2}}$ . The closed-loop system's trajectory  $p(t)$  are purple curves and they are contained by the bloated-tube.

## 5 Synthesizing the reference trajectories

In Section 4.3, we have seen that under certain conditions, the tracking error  $e(t)$  between an actual closed-loop trajectory  $\xi_g(t)$  and a piece-wise reference  $\xi_{\text{ref}}(t)$  can be bounded by a piece-wise constant value, which depends on the initial tracking error  $e(0)$  and the number of segments in  $\xi_{\text{ref}}$ . We have also seen an example nonlinear vehicle model with PWL  $\xi_{\text{ref}}$  for which the tracking error can be bounded.

In this section, we discuss how to utilize such bound on  $e(t)$  to help find a reference controller  $g_{\text{ref}}$  consisting of a reference trajectory  $\xi_{\text{ref}}(t)$  and a reference input  $u_{\text{ref}}(t)$  such that closed-loop trajectories  $\xi_g(t)$  from a neighborhood of  $\xi_{\text{ref}}(0)$  that are trying to follow  $\xi_{\text{ref}}(t)$  are guaranteed to satisfy the reach-avoid requirement. The idea of finding a  $g_{\text{ref}}$  follows a classic approach in robot motion planning. The intuition is that if we know at any time  $t \in [0, \xi_{\text{ref}}.\text{time}]$ ,  $\|\xi_g(t) \downarrow p - \xi_{\text{ref}}(t) \downarrow p\|_2$  will be at most  $\ell$ , then instead of requiring  $\xi_{\text{ref}}(t) \downarrow p$  to be at least  $\ell$  away from the obstacles (inside the goal region), we will bloat the obstacles (shrink the goal set) by  $\ell$ . Then the original problem is reduced to finding a  $\xi_{\text{ref}}(t)$  such that  $\xi_{\text{ref}}(t) \downarrow p$  can avoid the bloated obstacles and reach the shrunk goal set.

### 5.1 Use PWL reference trajectories for vehicle models

Finding a reference trajectory  $\xi_{\text{ref}}(t)$  such that (a)  $\xi_{\text{ref}}(t)$  satisfies the reach-avoid conditions, and (b)  $\xi_{\text{ref}}(t)$  and  $u_{\text{ref}}(t)$  are concatenations of state-input trajectory

pairs  $\{(\xi_{\text{ref},i}, u_{\text{ref},i})\}_i$  and each pair satisfies the system dynamics, is a nontrivial problem. If we were to encode the problem directly as a satisfiability or an optimization problem, the solver would have to search for over the space of continuous functions constrained by the above requirements, including the nonlinear differential constraints imposed by  $f$ . The standard tactic is to fix a reasonable template for  $\xi_{\text{ref}}(t), u_{\text{ref}}(t)$  and search for instantiations of this template.

From Example 4, we see that if  $\xi_{\text{ref}}$  is a PWL reference trajectory constructed from waypoints in the workspace, the tracking error can be bounded using Lemma 2. A PWL reference trajectories connecting the waypoints in the workspace have the flexibility to satisfy the reach-avoid requirement. Therefore, in this section, we fix  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$  to be the reference trajectory and reference input returned by the `Waypoints_to_Traj`( $\cdot, \cdot$ ). In Section 5.2, we will see that the problem of finding such PWL  $\xi_{\text{ref}}(t)$  can be reduced to a satisfiability problem over quantifier-free linear real arithmetic, which can be solved effectively by off-the-shelf SMT solvers (see Section 6 for empirical results).

## 5.2 Synthesizing waypoints for a linear reference trajectory

Algorithm 1 says that  $\xi_{\text{ref}}(t)$  and  $u_{\text{ref}}(t)$  can be uniquely constructed given a sequence of waypoints  $\{p_i\}_{i=0}^k$  in the workspace  $\mathcal{W}$  and a constant velocity  $\bar{v}$ . From Proposition 1,  $p_{\text{ref}}(t) = \xi_{\text{ref}}(t) \downarrow p$  connects the waypoints in  $\mathcal{W}$ . Also, let  $t_i = \sum_{j=1}^i \|p_j - p_{j-1}\|_2 / \bar{v}$  be the concatenation time,  $\forall t \in [t_{i-1}, t_i]$ ,  $p(t)$  is the line segment connecting  $p_{i-1}$  and  $p_i$ . We want to ensure that  $p(t) = \xi_g(t) \downarrow p$  satisfy the reach-avoid requirements. From Lemma 3, for any  $t \in [t_{i-1}, t_i]$ , we can bound  $\|p(t) - p_{\text{ref}}(t)\|_2$  with the constant  $\ell_i$ , then the remaining problem is to ensure that,  $p_{\text{ref}}(t)$  is at least  $\ell_i$  away from the obstacles and  $p_{\text{ref}}(\xi_{\text{ref}}.ltime)$  is inside the goal set with  $\ell_k$  distance to any surface of the goal set.

Let us start with one segment  $p(t)$  with  $t \in [t_{i-1}, t_i]$ . To enforce that  $p(t)$  is  $\ell_i$  away from a polytope obstacle, a sufficient condition is to enforce both the endpoints of the line segment to lie out at least one surface of the polytope bloated by  $\ell_i$ .

**Lemma 4.** *If  $p_{\text{ref}}(t)$  with  $t \in [t_{i-1}, t_i]$  is a line segment connecting  $p_{i-1}$  and  $p_i$  in  $\mathcal{W}$ . Given a polytope obstacle  $O = \text{Poly}(H_O, b_O)$  and  $\ell_i > 0$ , if*

$$\bigvee_{s=1}^{\text{dP}(H_O)} \left( (H_O^{(s)} p_{i-1} > b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i) \wedge (H_O^{(s)} p_i > b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i) \right) = \text{True},$$

then  $\forall t \in [t_{i-1}, t_i]$ ,  $B_{\ell_i}(p_{\text{ref}}(t)) \cap O = \emptyset$ .

*Proof.* Fix any  $s$  such that  $(H_O^{(s)} p_{i-1} > b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i) \wedge (H_O^{(s)} p_i > b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i)$  holds. The set  $S = \{q \in \mathbb{R}^d \mid H_O^{(s)} q > b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i\}$  defines a convex half space. Therefore, if  $p_{i-1} \in S$  and  $p_i \in S$ , then any point on the line segment connecting  $p_{i-1}$  and  $p_i$  is in  $S$ . Therefore, for any  $t \in [t_{i-1}, t_i]$ ,  $H_O^{(s)} p_{\text{ref}}(t) > b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i > b_O^{(s)}$ , which means  $p_{\text{ref}}(t) \notin O$ .

The distance between  $p_{\text{ref}}(t)$  and the surface  $H_O^{(s)}q = b_O^{(s)}$  is  $\frac{|H_O^{(s)}p_{\text{ref}}(t) - b_O^{(s)}|}{\|H_O^{(s)}\|_2} > \ell_i$ . Therefore, for any  $p \in B_{\ell_i}(p_{\text{ref}}(t))$  we have  $\|p - p_{\text{ref}}(t)\|_2 \leq \ell_i$  and thus  $p \notin O$ .

Furthermore,  $\bigwedge_{s=1}^{\text{dP}(H_O)} H_O^{(s)}q \leq b_O^{(s)} + \|H_O^{(s)}\|_2 \ell_i$  defines a new polytope that we get by bloating  $\text{Poly}(H_O, b_O)$  with  $\ell_i$ . Basically, it is constructed by moving each surface of  $\text{Poly}(H_O, b_O)$  along the surface's normal vector with the direction pointing outside the polytope.

Similarly, we can define the condition when  $p_{\text{ref}}(\xi.\text{time}) = p_k$  is inside the goal shrunk by  $\ell_k$ .

**Lemma 5.** *Given a polytope goal set  $G = \text{Poly}(H_G, b_G)$  and  $\ell_k > 0$ , if*

$$\bigwedge_{s=1}^{\text{dP}(H_G)} \left( H_G^{(s)}p_k \leq b_O^{(s)} - \|H_G^{(s)}\|_2 \ell_k \right) = \text{True}, \text{ then } B_{\ell_k}(p_k) \subseteq G.$$

Putting them all together, we want to solve the following satisfiability problem to ensure that each line segment between  $p_{i-1}$  and  $p_i$  is at least  $\ell_i$  away from all the obstacles and  $p_k$  is inside the goal set  $G$  with at least distance  $\ell_k$  to the surfaces of  $G$ . In this way,  $\xi_g(t)$  starting from a neighborhood of  $\xi_{\text{ref}}(0)$  can satisfy the reach-avoid requirement.

$$\begin{aligned} \phi_{\text{waypoints}}(p_{\text{ref}}(0), k, \mathbf{O}, G, \{\ell_i\}_{i=1}^k) = & \exists p_0, \dots, p_k, \\ & p_0 == p_{\text{ref}}(0) \\ & \bigwedge_{s=1}^{\text{dP}(H_G)} \left( H_G^{(s)}p_k \leq b_O^{(s)} - \|H_G^{(s)}\|_2 \ell_k \right) \\ \bigwedge_{i=1}^k \left( \bigwedge_{\text{Poly}(H,b) \in \mathbf{O}} \left( \bigvee_{s=1}^{\text{dP}(H)} \left( H^{(s)}p_{i-1} > b^{(s)} + \ell_i \|H^{(s)}\|_2 \wedge H^{(s)}p_i > b^{(s)} + \ell_i \|H^{(s)}\|_2 \right) \right) \right) \end{aligned}$$

Notice that the constraints in  $\phi_{\text{waypoints}}$  are all linear over real arithmetic. Moreover, the number of constraints in  $\phi_{\text{waypoints}}$  is  $O\left(\sum_{\text{Poly}(H,b) \in \mathbf{O}} k \text{dP}(H) + \text{dP}(H_G)\right)$ . That is, fixing  $k$ , the number of constraints will grow linearly with the total number of surfaces in the obstacle and goal set polytopes. Fixing  $\mathbf{O}$  and  $G$ , the number of constraints will grow linear with the number of line segments  $k$ .

**Theorem 1.** *Fix  $k \geq 1$  as the number of line segments,  $p_{\text{ref}}(0) \in \mathcal{W}$  as the initial position of the reference trajectory. Assume that*

- (1) *A closed with  $g_{\text{ref}}$  and  $g_{\text{trk}}$  is such that given any sequence of  $k+1$  waypoints in  $\mathcal{W}$  and any  $\bar{v}$ , the piece-wise reference  $\xi_{\text{ref}}$  (and input  $u_{\text{ref}}$ ) returned by Algorithm 1 satisfy the conditions in Lemmas 2 and 3 with Lyapunov function  $V(e(t))$  for the tracking error  $e(t)$ .*
- (2) *For the above  $\xi_{\text{ref}}$ , fix an  $\varepsilon_0$  such that  $V(e(0)) \leq \varepsilon_0$ , let  $\{\ell_i\}_{i=1}^k$  be error bounds for positions constructed using Lemma 2 and Lemma 3 from  $\varepsilon_0$ .*
- (3)  *$\phi_{\text{waypoints}}(p_{\text{ref}}(0), k, \mathbf{O}, G, \{\ell_i\}_{i=1}^k)$  is satisfiable with waypoints  $\{p_i\}_{i=0}^k$ .*

Let  $\xi_{\text{ref}}(t), u_{\text{ref}}(t) = \text{Waypoints\_to\_Trajectory}(\{p_i\}_{i=0}^k, \bar{v})$ , and  $p_{\text{ref}}(t) = \xi_{\text{ref}}(t) \downarrow p$ . Let  $\xi_g(t)$  be a trajectory of  $\mathcal{A}$  closed with  $g_{\text{trk}}(\cdot, \xi_{\text{ref}}, u_{\text{ref}})$  starting from  $\xi_g(0)$  with  $V(e(\xi_g(0), \xi_{\text{ref}}(0))) \leq \varepsilon_0$ , then  $\xi_g(t)$  satisfies the reach-avoid requirement.

*Proof.* Since  $\xi_{\text{ref}}(t), u_{\text{ref}}(t)$  are a PWL reference trajectory and a reference input respectively constructed from the waypoints  $\{p_i\}_{i=0}^k$ , they satisfy Assumption (1). Moreover,  $V(e(\xi_g(0), \xi_{\text{ref}}(0))) \leq \varepsilon_0$  satisfies Assumption (2). Using Lemma 2 and Lemma 3, we know that for  $t \in [t_{i-1}, t_i), \|\xi_g(t) \downarrow p - \xi_{\text{ref}}(t) \downarrow p\|_2 \leq \ell_i$ .

Finally, since  $\{p_i\}_{i=0}^k$  satisfy the constraints in  $\phi_{\text{waypoints}}$ , using Lemma 4 and Lemma 5, we know that for any time  $t \in [0, t_k], \xi_g(t) \downarrow p \notin \mathbf{O}$  and  $\xi_g(t_k) \in G$ . Therefore the theorem holds.

### 5.3 Partitioning the initial set

Starting from the entire initial set  $\Theta$ , fix  $\xi_{\text{ref}}(0) \in \Theta$  and an  $\varepsilon_0$  such that  $\forall x \in \Theta, V(e(x, \xi_{\text{ref}}(0))) \leq \varepsilon_0$ , then we can use Lemma 2 and Lemma 3 to construct the error bounds  $\{\ell_i\}_{i=1}^k$  for positions, and next use  $\{\ell_i\}_{i=1}^k$  to solve  $\phi_{\text{waypoints}}$  and find the waypoints and construct the reference trajectory.

However, if the initial set  $\Theta$  is too large,  $\{\ell_i\}_{i=1}^k$  could be too conservative so  $\phi_{\text{waypoints}}$  is not satisfiable. In the first two figures on the top row of Figure 3, we could see that if we bloat the obstacle polytopes using the largest  $\ell_i$ , then no reference trajectory is feasible. In this case, we partition the initial set  $\Theta$  to several smaller covers  $\Theta_j$  and repeat the above steps from each smaller cover  $\Theta_j$ . In Lemma 2 and Lemma 3 we could see that the values of  $\{\ell_i\}_{i=1}^k$  decrease if  $\varepsilon_0$  decreases. Therefore, with the partition of  $\Theta$ , we could possibly find a reference trajectory more and more easily. As shown in Figure 3 bottom row, after several partitions, a reference trajectory for each  $\Theta_j$  could be found.

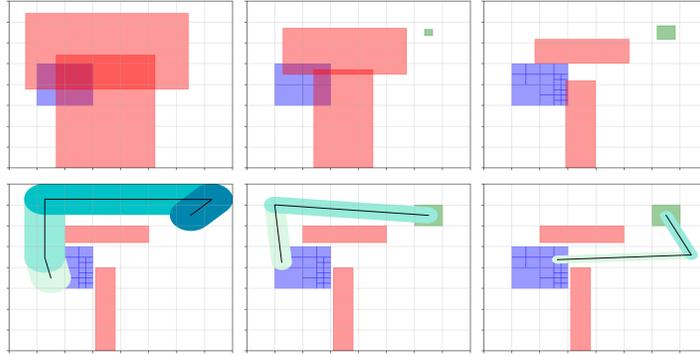


Fig. 3: Top row: each step attempting to find a reference trajectory in the space where obstacles (goal set) are bloated (shrunk) by the error bounds  $\{\ell_i\}_i$ . From left to right: Without partition,  $\{\ell_i\}_i$  are too large so a reference trajectory cannot be found.  $\Theta$  is partitioned, but  $\{\ell_i\}_i$  for the left-top cover are still too large. With further partitions, a reference trajectory could be found. Bottom row: It is shown that the bloated tubes for each cover (which contain all other trajectories from that cover) can fit between the original obstacles.

#### 5.4 Overall synthesis algorithm

Taking partitioning into the overall algorithm, we have Algorithm 2 to solve the controller synthesis problem defined in Section 2.2. Algorithm 2 takes in as inputs (1) an  $(n, m)$ -dimensional control system  $\mathcal{A}$ , (2) a tracking controller  $g_{\text{trk}}$ , (3) Obstacles  $\mathbf{O}$ , (4) a goal set  $G$ , (5) a Lyapunov function  $V(e(t))$  for the tracking error  $e$  that satisfies the conditions in Lemma 2 and Lemma 3 for any PWL reference trajectory and input, (6) the maximum number of line segments allowed  $\text{Seg}_{\text{max}}$ , (7) the maximum number of partitions allowed  $\text{Part}_{\text{max}}$ , and (8) a constant velocity  $\bar{v}$ . The algorithm returns a set  $\text{RefTrajs}$ , such that for each triple  $\langle \Theta_j, \xi_{j,\text{ref}}, u_{j,\text{ref}} \rangle \in \text{RefTrajs}$ , we have  $\forall x_0 \in \Theta_j$ , the unique trajectory  $\xi_g$  of the closed system ( $\mathcal{A}$  closed with  $g_{\text{trk}}(\cdot, \xi_{j,\text{ref}}, u_{j,\text{ref}})$ ) starting from  $x_0$  satisfies the reach-avoid requirement. The algorithm also returns  $\langle \text{Cover}, \text{None} \rangle$ , which means that the algorithm fails to find controllers for the portion of the initial set in  $\text{Cover}$  within the maximum number of partitions  $\text{Part}_{\text{max}}$ .

---

#### Algorithm 2: Controller synthesis algorithm

---

```

input :  $\mathcal{A} = \langle \mathcal{X}, \Theta, \mathbf{U}, f \rangle, g_{\text{trk}}, \mathbf{O}, G, V(e(t)), \text{Seg}_{\text{max}}, \text{Part}_{\text{max}}, \bar{v}$ 
initially:  $\text{Cover} \leftarrow \{\Theta\}, \text{prt} \leftarrow 0, k \leftarrow 1, \text{RefTrajs} \leftarrow \emptyset$ 
1 while  $(\text{Cover} \neq \emptyset) \wedge (\text{prt} \leq \text{Part}_{\text{max}})$  do
2   for  $\Theta \in \text{Cover}$  do
3      $\xi_{\text{init}} \leftarrow \text{Center}(\Theta)$ ;
4      $\varepsilon_0 \leftarrow a$  such that  $\forall x \in \Theta, V(e(x, \xi_{\text{init}})) \leq a$ ;
5      $\{\ell_i\}_{i=1}^k \leftarrow \text{GetBounds}(V(e(t)), \varepsilon_0)$ ;
6     while  $k \leq \text{Seg}_{\text{max}}$  do
7       if  $\text{CheckSAT}(\xi_{\text{init}} \downarrow p, k, \mathbf{O}, G, \{\ell_i\}_{i=1}^k) == \text{SAT}$  then
8          $p_0, \dots, p_k \leftarrow \text{GetValue}(\phi_{\text{waypoints}})$ ;
9          $\xi_{\text{ref}}, u_{\text{ref}} \leftarrow \text{Waypoints\_to\_Traj}(\{p_i\}_{i=0}^k, \bar{v})$ ;
10         $\text{RefTrajs} \leftarrow \text{RefTrajs} \cup \langle \Theta, \xi_{\text{ref}}, u_{\text{ref}} \rangle$ ;
11         $\text{Cover} \leftarrow \text{Cover} \setminus \{\Theta\}$ ;
12         $k \leftarrow 1$ ;
13        Break;
14      else
15         $k \leftarrow k + 1$ 
16      if  $k > \text{Seg}_{\text{max}}$  then
17         $\text{Cover} \leftarrow \text{Cover} \cup \text{Partition}(\Theta) \setminus \{\Theta\}$ ;
18         $\text{prt} \leftarrow \text{prt} + 1$ ;
19         $k \leftarrow 1$ ;
20 return  $\text{RefTrajs}, \langle \text{Cover}, \text{None} \rangle$ ;

```

---

In Algorithm 2,  $\text{Cover}$  is the collection of covers in  $\Theta$  that the corresponding  $\xi_{\text{ref}}$  and  $u_{\text{ref}}$  have not been discovered. Initially,  $\text{Cover}$  only contains  $\Theta$ . The **for**-loop from Line 2 will try to find a  $\xi_{\text{ref}}$  and a  $u_{\text{ref}}$  for each  $\Theta \in \text{Cover}$  until the maximum allowed number for partitions is reached. At line 3, we fix the initial state of  $\xi_{\text{ref}}(0) = \xi_{\text{init}}$  to be the center of the current cover  $\Theta$ . Then at Line 4, we get the initial error bounds  $\varepsilon_0$  after fixing  $\xi_{\text{init}}$ . Using  $\varepsilon_0$  and the Lyapunov function  $V(e)$ , we can construct the error bounds  $\{\ell_i\}_{i=1}^k$  for the positions of the vehicle using Lemma 2 and Lemma 3 at Line 5.

If the **if** condition at Line 7 holds with  $\{p_i\}_{i=0}^k$  being the waypoints that satisfy  $\phi_{\text{waypoints}}$ , then from Theorem 1 we know that the  $\xi_{\text{ref}}, u_{\text{ref}}$  constructed

using  $\{p_i\}_{i=0}^k$  at Line 9 will be such that, the unique trajectory  $\xi_g$  of the closed system ( $\mathcal{A}$  closed with  $g_{\text{trk}}(\cdot, \xi_{\text{ref}}, u_{\text{ref}})$ ) starting from  $x_0 \in \Theta$  satisfies the reach-avoid requirement. Otherwise the algorithm will increase the number of segments  $k$  in the PWL reference trajectory (Line 15). When the maximum number of line segments allowed is reached but the algorithm still could not find  $\xi_{\text{ref}}, u_{\text{ref}}$  that can guarantee the satisfaction of reach-void requirement from the current cover  $\Theta$ , we will partition the current  $\Theta$  at Line 17 and add those partitions to **Cover**. At the same time,  $k$  will be reset to 1.

**Theorem 2 (Soundness).** *Suppose the inputs to Algorithm 2,  $\mathcal{A}$ ,  $g_{\text{trk}}$ ,  $\mathbf{O}$ ,  $G$ ,  $V(e(t)), \bar{v}$  satisfy the conditions of Theorem 1. Let the output be **RefTrajs** =  $\{\langle \Theta_j, \xi_{j,\text{ref}}, u_{j,\text{ref}} \rangle\}_j$  and **Cover**, **None**, then we have (1).  $\Theta \subseteq \cup \Theta_j \cup \text{Cover}$ , and (2). for each triple  $\langle \Theta_j, \xi_{j,\text{ref}}, u_{j,\text{ref}} \rangle$ , we have  $\forall x_0 \in \Theta_j$ , the unique trajectory  $\xi_g$  of the closed system ( $\mathcal{A}$  closed with  $g_{\text{trk}}(\cdot, \xi_{j,\text{ref}}, u_{j,\text{ref}})$ ) starting from  $x_0$  satisfies the reach-avoid requirement.*

The theorem follows directly from the proof of Theorem 1.

## 6 Implementation and evaluation

We have implemented our synthesis algorithm (Algorithm 2) in a prototype open source tool we call **FACTEST**<sup>7</sup> (FAst ConTrollEr SynThesis framework). Our implementation uses Pypoman<sup>8</sup>, Yices 2.2 [6], SciPy<sup>9</sup> and NumPy<sup>10</sup> libraries. The inputs to **FACTEST** are the same as the inputs in Algorithm 2. **FACTEST** terminates in two ways. Either it finds a reference trajectory  $\xi_{j,\text{ref}}$  and reference input  $u_{j,\text{ref}}$  for every partition  $\Theta_j$  of  $\Theta$  so that Theorem 2 guarantees they solved the controller synthesis problem. Otherwise, it terminates by failing to find reference trajectories for at least one subset of  $\Theta$  after partitioning  $\Theta$  up to the maximum specified depth.

### 6.1 Benchmark scenarios: vehicle models and workspaces

We will report on evaluating **FACTEST** in several 2D and 3D scenarios drawn from motion planning literature (see Figures 4). Recall, the state space  $\mathcal{X}$  dimension corresponds to the vehicle model, and is separate from the dimensionality of the workspace  $\mathcal{W}$ . We will use four nonlinear vehicle models in these different scenarios: (a) the kinematic vehicle model (car) [31] introduced in Example 1, (b) a bijective mobile robot (robot) [13], (c) a hovering robot (hovercraft), and (d) an autonomous underwater vehicle (AUV) [29]. The dynamics and tracking controllers ( $g_{\text{trk}}$ ) of the other three models are described on the **FACTEST** website [27]. Each of these controllers come with a Lyapunov function that meets the assumptions of Lemmas 2 and 3 so the tracking error bounds given by the lemmas  $\{\ell\}_{i=1}^k$  can be computed.

<sup>7</sup> All models and source code of **FACTEST** are available at [27].

<sup>8</sup> <https://pypi.org/project/pypoman/>

<sup>9</sup> <https://www.scipy.org/>

<sup>10</sup> <https://numpy.org/>

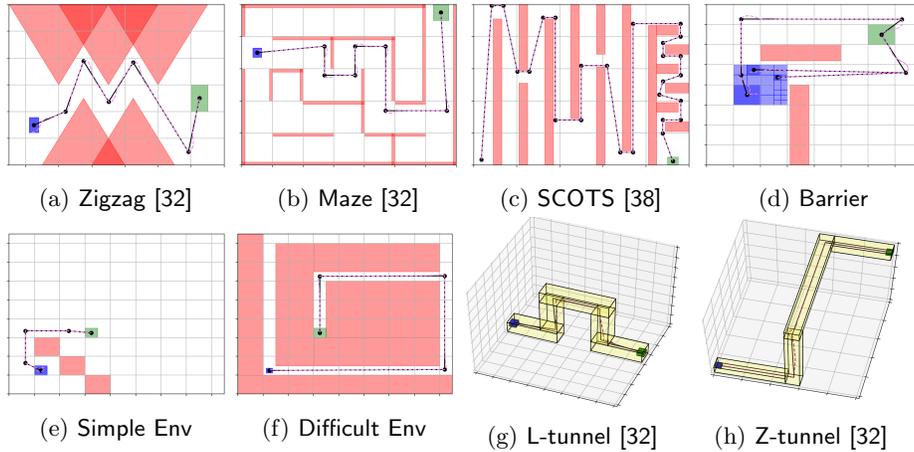


Fig. 4: 2D and 3D workspaces with initial (blue) and goal (green) sets. The scenarios run in the two-dimensional  $\mathcal{W}$  use the car model. The scenarios run in the three dimensional  $\mathcal{W}$  use the hovercraft model. The black lines denote  $\xi_{ref}$  and the dotted violet lines denote  $\xi_g$ .

## 6.2 Synthesis performance

Table 1 presents the performance of FACTEST on several synthesis problems. Several points are worth highlighting. (a) The absolute running time is at the sub-second range, even for 6-dimensional vehicle models with 4-inputs, operating in a 3D workspace. This is encouraging for online motion-control applications with dynamic obstacles. (b) The running time is not too sensitive to dimensions of  $\mathcal{X}$  and  $\mathbf{U}$  because the waypoints are only being generated in the lower dimensional workspace  $\mathcal{W}$ . Additionally, the construction of  $\xi_{ref}$  from the waypoints does not add significant time. However, since different models have different dynamics and Lypunov functions, they would have different error bounds for position. Such different bound could influence the final result. For example, the result for the Barrier scenario differs between the car and the robot. The car required 25 partitions to find a solution over all of  $\Theta$  and the robot required 22. (c) Confirming what we have seen in Section 5.2, the runtime of the algorithm scales with the number of segments required to solve the scenario and the number of obstacles. (d) As expected and seen in Zigzag scenarios, all other things being the same, the running time and the number of partitions grow with larger initial set uncertainty.

### Comparison with other motion controller synthesis tools: A challenge.

Few controller synthesis tools for nonlinear models are available for direct comparisons. We had detailed discussions with the authors of FastTrack [11], but found it difficult to plug-in new vehicle models. RTD [44] is implemented in MatLab also for specific vehicle models. Pessoa [26] and SCOTS [38] are implemented as general purpose tools. However, they are based on construction of discrete abstractions, which requires several additional user inputs. Therefore, we were only able to compare FACTEST with SCOTS and Pessoa using the sce-

Scenario	n,m	radius of $\Theta$	# $\mathbf{O}$	Time (s)	# segments per $\xi_{\text{ref}}$	# partitions
Zigzag, car 1	3, 2	0.200	9	0.037	6.0	1.0
Zigzag, car 2	3, 2	0.400	9	0.212	4.0	6.0
Zigzag, car 3	3, 2	0.800	9	0.915	5.0 - 6.0	16.0
Zigzag, robot 1	4, 2	0.200	9	0.038	6.0	1.0
Zigzag, robot 2	4, 2	0.400	9	0.227	4.0	6.0
Zigzag, robot 3	4, 2	0.800	9	0.911	5.0 - 6.0	16.0
Barrier car	3, 2	0.707	6	0.697	2.0 - 4.0	25.0
Barrier robot	4, 2	0.707	6	0.645	2.0 - 4.0	22.0
Maze, car	3, 2	0.200	22	0.174	8.0	1.0
Maze, robot	4, 2	0.200	22	0.180	8.0	1.0
SCOTS, car	3, 2	0.070	19	1.541	26.0	1.0
SCOTS, robot	4, 2	0.070	19	1.623	26.0	1.0
L-tunnel, hovercraft	4, 3	0.173	10	0.060	5.0	1.0
L-tunnel, AUV	6, 4	1.732	10	0.063	5.0	1.0
Z-tunnel, hovercraft	4, 3	0.173	5	0.029	4.0	1.0
Z-tunnel, AUV	6, 4	1.732	10	0.029	4.0	1.0

Table 1: Synthesis performance on different scenarios (environment, vehicle). Dimension of state space  $\mathcal{X}$  ( $n$ ), input ( $m$ ), radius of initial set  $\Theta$ , number of obstacles  $\mathbf{O}$ , running time (in seconds).

nario SCOTS. This scenario was originally built in SCOTS and is using the same car model.

The results for SCOTS and Pessoa can be found in [38]. The total runtime of SCOTS consists of the abstraction time  $t_{\text{abs}}$  and the synthesis time  $t_{\text{syn}}$ . The Pessoa tool has an abstraction time of  $t_{\text{abs}} = 13509\text{s}$  and a synthesis time of  $t_{\text{syn}} = 535\text{s}$ , which gives a total time of  $t_{\text{tot}} = 14044\text{s}$ . The SCOTS tool has a has an abstraction time of  $t_{\text{abs}} = 100\text{s}$  and a synthesis time of  $t_{\text{syn}} = 413\text{s}$ , which gives a total time of  $t_{\text{tot}} = 513\text{s}$ . FACTEST clearly outperforms both SCOTS and Pessoa with a total runtime of  $t_{\text{tot}} = 1.541\text{s}$ . This could be attributed to the fact that FACTEST does not have to perform any abstractions, but even by looking sole at  $t_{\text{syn}}$ , FACTEST is significantly faster. However, we do note that the inputs of FACTEST and SCOTS are different. For example, SCOTS needs a growth bound function  $\beta$  for the dynamics but FACTEST requires Lyapunov functions for the tracking error.

### 6.3 RRT vs. SAT-Plan

To demonstrate the speed of our SAT-based reference trajectory synthesis algorithm (i.e. only the **while**-loop from Line 6 to Line 15 of Algorithm 2 which we call **SAT-Plan**), we compare it with Rapidly-exploring Random Trees (RRT) [20]. The running time, number of line segments, and number of iterations needed to find a path were compared. RRT was run using the Python Robotics library [39], which is not necessarily an optimized implementation. SAT-Plan was run using Yices 2.2. The scenarios are displayed in Figure 4 and the results are in Figure 5.

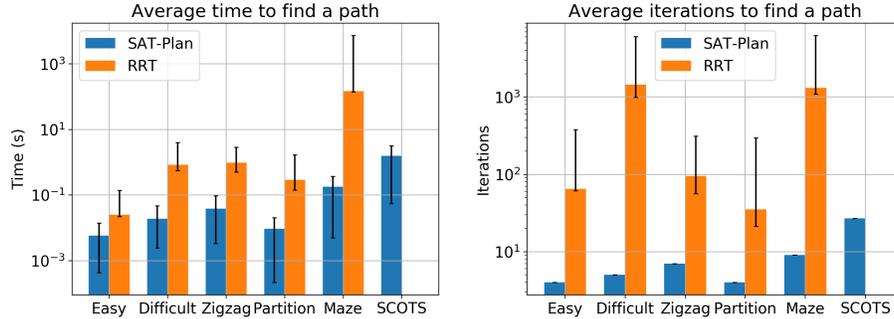


Fig. 5: Comparison of RRT and SAT-Plan. The left plot shows the runtime and the right plot shows the number of necessary iterations. Note that RRT timed out on the SCOTS scenario.

Each planner was run 100 times. The colored bars represent the average runtime and average number of iterations. The error bars represent the range of minimum and maximum. The RRT path planner was given a maximum of 5000 iterations and a path resolution of 0.01. SAT-Plan was given a maximum of 100 line segments to find a path. RRT timed out for the SCOTS scenario, unable to find a trajectory within 5000 iterations. The maze scenario timed out about 10% of the time.

Overall SAT-Plan scales in time much better as the size of the unsafe set increases. Additionally, the maximum number of iterations that RRT had to perform was far greater than the average number of line segments needed to find a safe path. This means that the maximum number of iterations that RRT must go through must be sufficiently large, or else a safe path will not be found even if one exists. SAT-Plan does not have randomness and therefore will find a reference trajectory (with  $k$  segments) in the modified space (bloated obstacles and shrunk goal) if one (with  $k$  segments) exists. Various examples of solutions found by RRT and SAT-Plan can be found on the FACTEST’s website [27].

## 7 Conclusion and discussion

We introduced a technique for synthesizing correct-by-construction controllers for a nonlinear vehicle models, including ground, underwater, and aerial vehicles, for reach-avoid requirements. Our tool FACTEST implementing this technique shows very encouraging performance on various vehicle models in different 2D and 3D scenarios.

There are several directions for future investigations. (1) One could explore a broader class of reference trajectories to reduce the tracking error bounds. (2) It would also be useful to extend the technique so the synthesized controller can satisfy the actuation constraints automatically. (3) Currently we require user to provide the tracking controller  $g_{\text{trk}}$  with the Lyapunov functions, it would be interesting to further automate this step.

## References

1. Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431. IEEE, 2019.
2. M. Mahdi Ghazaei Ardakani, Björn Olofsson, Anders Robertsson, and Rolf Johansson. Real-time trajectory generation using model predictive control. In *IEEE International Conference on Automation Science and Engineering*, pages 942–948. IEEE, 2015.
3. Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.
4. Alberto Bemporad, Francesco Borrelli, and Manfred Morari. Model predictive control based on linear programming - The explicit solution. *IEEE Transactions on Automatic Control*, 47(12):1974–1985, 2002.
5. Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.
6. Bruno Dutertre. Yices 2.2. In *International Conference on Computer Aided Verification*, pages 737–744. Springer, 2014.
7. Chuchu Fan, Umang Mathur, Sayan Mitra, and Mahesh Viswanathan. Controller synthesis made real: reach-avoid specifications and linear dynamics. In *International Conference on Computer Aided Verification*, pages 347–366. Springer, 2018.
8. Jose M.Mendes Filho, Eric Lucet, and David Filliat. Real-time distributed receding horizon motion planning and control for mobile multi-robot dynamic systems. In *International Conference on Robotics and Automation*, pages 657–663. IEEE, 2017.
9. Ioannis Filippidis, Sumanth Dathathri, Scott C. Livingston, Necmiye Ozay, and Richard M. Murray. Control design for hybrid systems with tulip: The temporal logic planning toolbox. In *IEEE Conference on Control Applications*, pages 1030–1041, 2016.
10. Antoine Girard. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5):947–953, 2012.
11. Sylvia L Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. FaSTrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522. IEEE, 2017.
12. Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research*, 34(7):883–921, 2015.
13. Yutaka Kanayama, Yoshihiko Kimura, Fumio Miyazaki, and Tetsuo Noguchi. A stable tracking control method for an autonomous mobile robot. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 384–389. IEEE, 1990.
14. Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104(2), 2010.
15. Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
16. Hassan K Khalil and Jessy W Grizzle. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.
17. Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

18. Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.
19. James J Kuffner and Steven M LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001. IEEE, 2000.
20. Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.
21. Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.
22. Chang Liu, Seungho Lee, Scott Varnhagen, and H. Eric Tseng. Path planning for autonomous vehicles using model predictive control. In *IEEE Intelligent Vehicles Symposium*, pages 174–179. IEEE, 2017.
23. 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.
24. Kaushik Mallik, Anne-Kathrin Schmuck, Sadegh Soudjani, and Rupak Majumdar. Compositional synthesis of finite-state abstractions. *IEEE Transactions on Automatic Control*, 64(6):2629–2636, 2018.
25. David Q. Mayne. Model predictive control: Recent developments and future promise. In *Automatica*, volume 50, pages 2967–2986. Pergamon, 2014.
26. Manuel Mazo, Anna Davitian, and Paulo Tabuada. Pessoa: A tool for embedded controller synthesis. In *International Conference on Computer Aided Verification*, pages 566–569. Springer, 2010.
27. Kristina Miller, Chuchu Fan, and Sayan Mitra. Factest webpage. <https://kmmille.github.io/FACTEST/index.html>, 2020 (accessed May 13, 2020).
28. Sebti Mouelhi, Antoine Girard, and Gregor Gössler. CoSyMA: A tool for controller synthesis using multi-scale abstractions. In *International Conference on Hybrid Systems: Computation and Control*, pages 83–88. ACM, 2013.
29. Yoshihiko Nakamura and Shrikant Savant. Nonlinear tracking control of autonomous underwater vehicles. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages A4–A9. IEEE, 1992.
30. Katsuhiko Ogata and Yanjuan Yang. *Modern control engineering*, volume 5. Prentice hall Upper Saddle River, NJ, 2010.
31. Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.
32. Texas A&M University Parasol MP Group, CSE Dept. Algorithms & applications group benchmarks.
33. Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model predictive control with signal temporal logic specifications. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 81–87. IEEE, 2014.
34. Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *International Conference on Hybrid Systems: Computation and Control*, pages 239–248. ACM, 2015.
35. Hadi Ravanbakhsh and Sriram Sankaranarayanan. Robust controller synthesis of switched systems using counterexample guided framework. In *2016 international conference on embedded software (EMSOFT)*, pages 1–10. IEEE, 2016.

36. Stefan Richter, Colin Neil Jones, and Manfred Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2011.
37. Pritam Roy, Paulo Tabuada, and Rupak Majumdar. Pessoa 2.0: A controller synthesis tool for cyber-physical systems. In *International Conference on Hybrid Systems: Computation and Control*, pages 315–316. ACM, 2011.
38. Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th international conference on hybrid systems: Computation and control*, pages 99–104, 2016.
39. Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. Pythonrobotics: a python code collection of robotics algorithms, 2018.
40. Sumeet Singh, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust online motion planning via contraction theory and convex optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5883–5890. IEEE, 2017.
41. Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.
42. Paulo Tabuada. *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.
43. Russ Tedrake. Lqr-trees: Feedback motion planning on sparse randomized trees. 2009.
44. Sean Vaskov, Shreyas Kousik, Hannah Larson, Fan Bu, James Ward, Stewart Worrall, Matthew Johnson-Roberson, and Ram Vasudevan. Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments. *arXiv preprint arXiv:1902.02851*, 2019.
45. Michael Vitus, Vijay Pradeep, Gabriel Hoffmann, Steven Waslander, and Claire Tomlin. Tunnel-milp: Path planning with sequential convex polytopes. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7132, 2008.
46. Kai Weng Wong, Cameron Finucane, and Hadas Kress-Gazit. Provably-correct robot control with LTLMoP, OMPL and ROS. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 2073, 2013.
47. Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830, 2012.
48. Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. Tulip: A software toolbox for receding horizon temporal logic planning. In *International Conference on Hybrid Systems: Computation and Control*, pages 313–314. ACM, 2011.
49. Melanie Nicole Zeilinger, Colin Neil Jones, and Manfred Morari. Real-time sub-optimal model predictive control using a combination of explicit MPC and online optimization. *IEEE Transactions on Automatic Control*, 56(7):1524–1534, 2011.