Planning in Dynamic and Partially Unknown Environments

Kristina Miller * **Chuchu Fan** ** **Sayan Mitra** *

* *University of Illinois Urbana-Champaign, Champaign, IL 61820 USA (e-mail: {kmmille2, mitras}@ illinois.edu)*
** *MIT, Cambridge, MA 02139 USA (e-mail: chuchu@mit.edu)*

Abstract: Motion planning in dynamic and partially unknown environments is a difficult problem requiring both perception and control components. We propose a solution to the control component while cleanly abstracting perception. We show that this clean abstraction can be used to synthesize verifiably safe reference trajectories using a combination of reachability analysis and Mixed Integer Linear Programming. Experiments with a prototype implementation of this algorithm show promise as it has subsecond synthesis performance for nonlinear vehicle models in scenarios with hundred plus obstacles on standard hardware.

Keywords: control synthesis, switched systems, cyberphysical systems, verification, autonomy

1. INTRODUCTION

Controller synthesis is the problem of generating a control function that guarantees a system meets some higher-level specification. Various synthesis algorithms have been created for restricted specification classes and implemented in tools (see [13, 21, 18, 1, 16, 20] and references). One challenge is the state space explosion problem, which limits discretization-based approaches. Another challenge arises as as we do not yet have tractable abstractions for learned models of perception [11, 12, 4].

We contribute a new synthesis algorithm to address these challenges. First, we present a perception abstraction called a *perception oracle (PO)* which can be used to explore synthesis algorithms. A PO gives an estimate of obstacles around the vehicle, and generalizes and formalizes other common abstractions [4]. It captures perception algorithms like occupancy grids [3], OctSqueeze [9], and pedestrian detection and intent tracking [10].

Second, we present a synthesis algorithm that relies on periodic invocations of the PO. We use carefully computed error bounds about a low-level *tracking controller* and generate a sequence of waypoints that are safe relative to the output of the PO and attempts to acheive the high-level goal. We show that under reasonable assumptions and using symmetries, a small number of reachable set computations can be transformed to characterize tracking errors relative to arbitrary, and *possibly piece-wise continuous*, reference trajectories defined by timed waypoints. We then show how safe reference trajectories can be computed by using Mixed Integer Linear Programming (MILP).

[1] The authors acknowledge support from the research grants NSF FMITF: 1918531 and AFOSR FA9550-17-1-0236, the DARPA Assured Autonomy under contract FA8750-19-C-0089, and from the Defense Science and Technology Agency in Singapore. 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, the U.S. Government, DSTA Singapore, or the Singapore Government.

Our algorithm is implemented in an open source tool and evaluated in various scenarios involving a nonlinear vehicle model navigating a crowded intersection. Our experiments show that safe reference trajectories are found in partially visible and dynamic environments when system parameters are reasonable according to conditions related to the soundness of the algorithm. This takes less than a second, even with over 100 pedestrians. We compare against Model Predictive Control (MPC), and show that our approach can plan over longer periods and with more obstacles.

Closely related research. Real-time motion planning is a major open problem in robotics and an active area of research with many papers. Our algorithm is comparable to those in FaSTrack [8], RTD [24], FACTEST [7], and the funnel library [14]. None of these approaches define the accuracy and timing behavior of perception modules and differ in the key step of computing tracking error bounds—which has implications on data-structures, applicability, and performance. In [8], Hamilton-Jacobi-based reachability analysis is used to produce the tracking error bounds. Convex optimization is used for computing funnels around tracking controllers in [14]. In [24] reachable sets are computed using convex optimization and it generates controllers that are provable not-at-fault. In [7], Lyapunov-type functions are used to compute error bounds and the waypoints are computed using satisfiability solvers. In contrast, our algorithm uses sampling-based reachability analysis and generates the waypoints using MILP. We implemented our algorithm within the FACTEST framework, and our approach can deal with dynamic obstacles. MPC has also been used to synthesize controllers with reach-avoid specifications. Implicit MPC [15] relies on solving an optimal control problem with explicit solutions [5, 25, 27] or with LP [5], QP [2, 19], or MILP solvers [25, 18].

2. ONLINE PLANNING IN DYNAMIC ENVIRONMENTS

Notations. The length of the line segment joining $p_1, p_2 \in \mathbb{R}^n$ is denoted by $\|p_2 - p_1\|_2$. A polytope is denoted by

$Poly(H,b) = \{x \in \mathbb{R}^n | Hx \leq b\}$ where $H \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. The i^{th} row of H is $H^{(i)} \in \mathbb{R}^n$ and the i^{th} row of b is $b^{(i)} \in \mathbb{R}$. The set $\{x : H^{(i)}x = b^{(i)}\}$ is a face of $Poly(H,b)$ and there are m faces for $H \in \mathbb{R}^{m\times n}$, and we denote this number as $\mathsf{dP}(H)$. A ball of radius r centered at p is $\mathcal{B}_r(p)$, and the bloated set of a segment S is $\mathcal{B}_r(S)$.

The overall vehicle system $\mathcal{S}$ is specified by three subsystems: (i) a *perception subsystem* $\mathcal{P}$ provides a prediction for the free space around the vehicle, (ii) a *plant-controller subsystem* $\mathcal{C}$ guides the system towards a specific waypoint, and (iii) a *planner* $\mathcal{L}$ produces a reference trajectory for $\mathcal{C}$ based on the inputs from $\mathcal{P}$. We assume that $\mathcal{P}$ and $\mathcal{C}$ are given and present an algorithm implementing $\mathcal{L}$ such that $\mathcal{S}$ satisfies given safety requirements. In this section, we spell out details about $\mathcal{P}$ and $\mathcal{C}$ and how they are composed to define $\mathcal{S}$. This can be seen in Figure 1.

2.1 Perception Subsystem: Oracles

The two or three dimensional subset of the Euclidean space occupied by the physical system and obstacles is the *workspace* $\mathcal{W}$. The part of $\mathcal{W}$ not occupied by static and dynamic obstacles is called the *freespace*. The freespace is the *ground truth* function: $\mathbf{W} : \mathbb{R}_{\geq 0} \to \mathcal{W}$ such that if the system occupies $\mathbf{W}(t)$, at any time t, then it is considered to be safe. The obstacle space is therefore $\mathbf{W}^C(t)$.

We are interested in studying planning and control problems for arbitrary perception technologies, hence, we abstract $\mathcal{P}$ in terms of a mathematical object called *a perception oracle (PO)*. A PO with a space-time horizon of (d_P, T_P), when queried at time t, returns a function underapproximating the freespace in the d_P-neighborhood of the vehicle over the time interval $[t, t+T_P]$. By underapproximating the freespace, we can take into account a set of possible trajectories for the dynamic obstacles.

Definition 1. (Perception oracle). *A (T_P, d_P)-perception oracle queried at time t and position $x \in \mathcal{X}$ returns a function $\mathbf{FS}_{t,x} : [t, t+T_P] \to 2^{\mathcal{W}}$ such that for any time $t' \in [t, t+T_P]$, the freespace in $\mathcal{B}_{d_P}(x) \subset \mathcal{W}$ contains the oracles output $\mathbf{FS}_{t,x}(t')$. That is, $\forall\ t' \in [t, t+T_P]$, $\mathbf{FS}_{t,x}(t') \subseteq \mathbf{W}(t') \cap \mathcal{B}_{d_P}(x)$. T_P is the time horizon and d_P is the sensing distance of the oracle.*

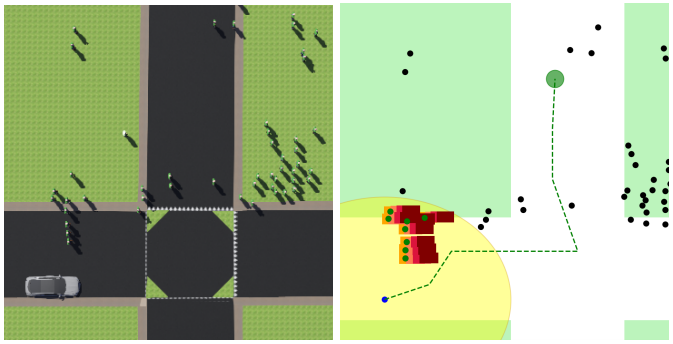

Fig. 1. Intersection scenario in Webots (left) and algorithm's internal representation (right). The vehicle (blue) must reach the green circle while avoiding pedestrians (black). The PO returns a subset of the pedestrians (green dots) within the sensed space (yellow) with predicted future occupancy (orange-red). The planner computes waypoints (dotted line).

Definition 1 may be too strong, but we posit that it is a good abstraction for studying planning and control algorithms. Perception algorithms such as occupancy grids [3], or OctSqueeze [9], essentially implement probabilistic versions of this definition. However, a probabilistic variant of Definition 1 would unnecessarily complicate and distract from our theory. Instead, we propose working with confidence bounds when using a probabilistic implementation of PO. Approaches for pedestrian detection and intent tracking [10] essentially implement Definition 1, by computing the complement of $\mathbf{FS}_{t,x}$. Even with Definition 1, planning and control remain highly nontrivial as the agent only has partial (and possibly stale) view of the world.

We represent the *complement* of $\mathbf{FS}_{t,x}$ by a set of *timed obstacles* which are polytopes in $\mathbf{W}^C \times \mathbb{R}_{\geq 0}$. A PO queried at time t_q and state x_q returns $\mathbf{FS}_{t_q,x_q}(t)$ represented by a list of timed obstacles $O_i = Poly(H_{O_i}, b_{O_i})$. The set of all PO's is $\mathbb{O}$.

2.2 Plant-Controller Subsystem

The plant-controller subsystem $\mathcal{C}$ is specified by a 4-tuple: (i) the state space $\mathcal{X} \subseteq \mathbb{R}^n$, (ii) the initial set $X_0 \subseteq \mathcal{X}$, (iii) a set of modes $\mathcal{Z}$, and (iv) a dynamic function $f : \mathcal{X} \times \mathcal{Z} \to \mathcal{X}$ that is locally Lipschitz continuous with respect to the first argument.

Given a mode $z \in \mathcal{Z}$ and an initial state $x_0 \in X_0$, a *solution* or *trajectory* of $\mathcal{C}$ up to a time bound T, is a continuous trajectory $\xi : [0,T] \mapsto \mathcal{X}$ that satisfies (i) $\xi(0) = x_0$ and (ii) for any $t \in [0,T]$, $\frac{d\xi(t)}{dt} = f(\xi(t), z)$. The initial state and mode of a trajectory ξ is $\xi(0)$ and z, respectively. In Section 3.2, we see that each mode will be a *reference trajectory* (or path) generated by $\mathcal{L}$ of the form $z : \mathbb{R}_{\geq 0} \to \mathcal{X}$. Once a mode is fixed, $\mathcal{C}$ will ideally track this path. The projection of a trajectory onto the workspace $\mathcal{W}$ is denoted by $\xi \downarrow \mathcal{W}$. We want to ensure that the vehicle always resides in the freespace, i.e., $\forall\, t, (\xi \downarrow \mathcal{W})(t) \in \mathbf{W}(t)$.

2.3 Planner and Closed-Loop System

The planner function $\mathcal{L}$ takes in the current state of the vehicle, the goal G, and a perception oracle, and produces a mode for $\mathcal{C}$. That is, $\mathcal{L} : \mathcal{X} \times \mathbb{O} \to \mathcal{Z}$, where the dependence on G is implicit.

In autonomous systems, the perception, control, and planner modules involve many threads and processes running on an operating system and communicating over channels and buses. We consider a relatively clean yet implementable mathematical model of $\mathcal{S}$, and specify the execution times in terms of the period $T_{\mathcal{S}}$ with which the PO is queried and a new prediction is obtained.

The timing of $\mathcal{S}$ is as follows. The PO is queried at each $iT_{\mathcal{S}}$ time. The computed reference trajectory z_i is the mode for ξ_i over $t \in [iT_{\mathcal{S}}, (i+1)T_{\mathcal{S}}]$. Simultaneousy, the $z_{i+1}(t)$ to be followed over $[(i+1)T_{\mathcal{S}}, (i+2)T_{\mathcal{S}}]$ is computed. Therefore, $\mathcal{S}$ is a switched system, with mode switches every $iT_{\mathcal{S}}$ time.

An *execution* of $\mathcal{S}$ is a sequence $\alpha = \xi_0, \xi_1, \ldots$, such that (i) $\xi_0(0) \in X_0$, and (ii) for all $i > 0$, ξ_i is a trajectory of duration $T_{\mathcal{S}}$ for $\mathcal{C}$ with mode z_i and starting from the state $\xi_i(0) = \xi_{i-1}(T_{\mathcal{S}})$. The mode z_i is computed using $z_{i-1}(T_{\mathcal{S}})$ and the output of the PO invoked at time $(i-1)T_{\mathcal{S}}$. That is, $z_i = \mathcal{L}(z_{i-1}(T_{\mathcal{S}}), \mathbf{FS}_{(i-1)T_{\mathcal{S}}, \xi_{i-1}(0)})$. This switching rule models the situation where the PO provides predictions based on slightly stale information, and that the computation of the planner output at time $iT_{\mathcal{S}}$ commences sometime in the previous $T_{\mathcal{S}}$ interval.

Problem statement The controller synthesis problem is the following: Given a plant-controller subsystem $\mathcal{C}$, a perception subsystem $\mathcal{P}$, a goal $G \subseteq \mathcal{W}$, and time constant $T_{\mathcal{S}}$, we want to design a planner $\mathcal{L}$ such that the along all executions of $\mathcal{S}$, (i) $\forall i, t$, $\xi_i(t) \downarrow \mathcal{W} \in \mathbf{W}(t)$ and (ii) $\xi_N(\xi_N.\mathsf{ltime}) \in G$. Here, there are N segments in the execution and $\xi_N.\mathsf{ltime}$ is the last time of ξ_N.

3. SYNTHESIS ALGORITHM FOR PLANNER

Overview In this section, we present a planner algorithm which computes reference trajectories with inputs from the perception and plant-controller subsystems. First, we bound the error between the actual vehicle trajectory and the reference *for an arbitrary reference* using reachability analysis (Section 3.1). Then, using these error bounds, we synthesize safe reference trajectories guaranteed to be in the freespace predicted by the PO. Our reference trajectories z_i will be piecewise linear (PWL) functions of time constructed from *timed waypoints* $\{(p_{i,k}, t_{i,k})\}_{k=0}^{N}$, where each waypoint $p_{i,k} \in \mathcal{W}$ and $t \in \mathbb{R}_{\geq 0}$. In Section 3.2, we present constraints needed to synthesize timed waypoints that guarantee $\xi_i(t)$ is safe. Finally, we present the full algorithm which takes the timing of the subsystems into account (Section 3.4). This algorithm performs two tasks over each time period $[iT_{\mathcal{S}}, (i+1)T_{\mathcal{S}}]$: 1) It drives the system trajectory $\xi_i(t)$ to follow $z_i(t)$. 2) It queries a (T_P, d_P)-PO, and the returned freespace over $[iT_{\mathcal{S}}, iT_{\mathcal{S}} + T_P]$ is used to compute a reference trajectory $z_{i+1}(t)$ for the time period $[(i+1)T_{\mathcal{S}}, (i+2)T_{\mathcal{S}}]$. Thus, we have the mild requirement that the prediction period (T_P) be at least twice the planning period $T_{\mathcal{S}}$ ($T_P > 2T_{\mathcal{S}}$).

3.1 Bounding Error to Arbitrary References

First, we must bound the error of the system trajectory to an arbitrary reference trajectory, so that the vehicle is safe over $[0, T_{\mathcal{S}}]$. The reference trajectory $z_i(t)$ is constructed from the $N+1$ waypoints $\{(p_{i,k}, t_{i,k})\}_{k=0}^{N}$ that define the segments that make up $z_i(t)$. The system trajectory $\xi_i(t)$ follows $z_i(t)$ over $[iT_{\mathcal{S}}, (i+1)T_{\mathcal{S}}]$. The system error over this time period is given by $\varepsilon_i(t) = z_i(t) - \xi_i(t)$. In this section, we focus on bounding the error of one such reference trajectory. Therefore, we drop the subscript i. Then, the reference trajectory is $z(t)$, the timed waypoints are $\{(p_k, t_k)\}_{k=0}^{N}$, the system trajectory is $\xi(t)$, and the system error is $\varepsilon(t) = z(t) - \xi(t)$.

The error function with respect to arbitrary reference trajectories is usually not easy to compute analytically. This problem is tackled by different methods in different tools and frameworks, as discussed in the related research section. We use recent advances in sensitivity-based reachability analysis algorithms due to their flexibility and performance [17, 6]. Reachability analysis would nominally require expensive new computations for every new ξ_i and z_i. Instead, we use symmetry properties of reachable sets from [23, 22] to perform a one-time computation that gives all the necessary error bounds for every ξ_i and z_i.

3.1.1. Error along a single segment In [23], the results for symmetry transformations of reach sets were developed for hybrid systems where each mode corresponded to a pair of waypoints. Here, we adapt the results to our notion of time-stamped waypoints. We present key definitions and theorems for completeness. The main symmetry result is Lemma 1, which allows us to transform pre-computed reach sets instead recomputing them for each reference trajectory. These reach sets are used to find the error bounds over each segment. These results are not novel, but they are crucial to proving safety of our main algorithm.

In what follows, we use $f(x, S_k)$ to denote the system dynamics following a segment $S_k \in \mathcal{W}$. Here, $S_k(t) = (p_k - p_{k-1})\frac{t-t_{k-1}}{t_k - t_{k-1}} t + p_{k-1}$ and it is valid over $[t_{k-1}, t_k]$. The trajectory starting from x_0 at time t_{k-1} and following S_k is $\xi(x_0, S_k, t)$. The *reachable set* of a system starting from a set $X_0 \subseteq \mathcal{X}$ and following the segment S_k up to time t, is $\mathsf{Reach}(X_0, S_k, t)$, and it contains all $\xi(x_0, S_k, t)$ for which $x_o \in X_0$. The size of $\mathsf{Reach}(X_0, S_k, t)$ at time t is $\mathsf{Rad}(\mathsf{Reach}(X_0, S_k, t))$.

Definition 2. *Let Γ be a group of linear operators acting on $\mathbb{R}^n$. We say $\gamma \in \Gamma$ where $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ is a symmetry of a system $\mathcal{C}$ if for any trajectory, $\xi(x_0, S_k, t)$, $\gamma \cdot \xi(x_0, S_k, t)$ is also a trajectory.*

Definition 3. *The closed-loop dynamic function $f : \mathcal{X} \times (\mathbb{R}_{\geq 0} \times \mathcal{W})^2 \to \mathcal{X}$ is Γ-equivariant if for any $\gamma \in \Gamma$, there exists $\rho : (\mathbb{R}_{\geq 0} \times \mathcal{W})^2 \to (\mathbb{R}_{\geq 0} \times \mathcal{W})^2$ such that for all $x \in \mathcal{X}$, $\frac{\partial \gamma}{\partial x} f(x, S_k) = f(\gamma(x), \rho(S_k))$.*

We can use the symmetry operation γ to construct a new trajectory without simulating the system, but by transforming the original trajectory using $\gamma(\cdot)$. Theorem 1 gives a sufficient condition under which γ is a symmetry.

Theorem 1. *If f is Γ-equivariant, then for any trajectory $\xi(x_0, S_k, t)$ and $\gamma \in \Gamma$, we have $\gamma(\xi(x_0, S_k, \cdot)) = \xi(\gamma(x_0), \rho(S_k), \cdot)$, where ρ is the transformation associated with γ in Definition 2.*

Lemma 1 (adopted from Theorem 2 in [23]) shows that for any transformation of the system dynamics following some S_k, a corresponding transformation can be applied to the reachable set. This result is useful since the reference trajectories computed in Section 3.2 are PWL, meaning that only one reachable set needs to be computed to follow some linear segment S_k, then transformations can be applied to every other segment S_j, $j \neq k$.

Lemma 1. *Let f be Γ-equivariant, then for any $\gamma \in \Gamma$ and its corresponding ρ, $\gamma(\mathsf{Reach}(X_0, S_k, t))) = \mathsf{Reach}(\gamma(X_0), \rho(S_k), t))$, where X_0 is the initial state and S_k is a segment of the reference trajectory.*

Since the system dynamics are Γ-equivariant for our symmetry, the reachable sets can be computed independently of $z(t)$ when the initial set X_0 is known. The error bounds for the system following the k^{th} waypoint is given by (1).

$$\ell_k = \max_t \mathsf{Rad}(\mathsf{Reach}(X_0, S_k, t))) \tag{1}$$

This bound is valid over $t \in [t_{k-1}, t_k]$, meaning $\xi(x_0, S_k, t)$ is always within ℓ_k of the reference $z(t)$.

Lemma 2. *Consider a system $\mathcal{C}$ following the segment S_k constructed from timed waypoints (p_{k-1}, t_{k-1}) and (p_k, t_k) and an initial position $x_0 \in X_0$. At any time $t \in [t_{k-1}, t_k]$, $\xi(x_0, S_k, t) \downarrow \mathcal{W} \in \mathcal{B}_{\ell_k}(S_k(t))$.*

Proof. Fix any segment S_k of the reference trajectory $z(t)$ and $t \in [t_{k-1}, t_k]$. Then, for any $x_0 \in X_0$, the actual trajectory $\xi(x_0, S_k, t) \in \mathsf{Reach}(X_0, S_k, t)$. The error of the system is defined as $\varepsilon(t) = z(t) - \xi(x_0, S_k, t)$. The set of all errors is $\mathcal{E}(t) = \{z(t) -$

$\xi(x_0, S_k, t)|x_0 \in X_0\}$ and since this is $\mathsf{Reach}(X_0, S_l, t)$ translated, $\mathsf{Rad}(\mathcal{E}(t)) = \mathsf{Rad}(\mathsf{Reach}(X_0, S_k, t))$. Since $\varepsilon(t) \in \mathcal{E}(t)$, then $\|\varepsilon(t)\| \leq \mathsf{Rad}(\mathsf{Reach}(X_0, S_k, t))$. Using (1), we can see that $\|\varepsilon(t)\| \leq \ell_k$. Therefore, $\xi(x_0, S_k, t) \in \mathcal{B}_{\ell_k}(z(t))\forall x_0 \in X_0$, and when projected onto the workspace, $\xi(x_0, S_k, t) \downarrow \mathcal{W}\mathcal{B}_{\ell_k}(S_k)\forall x_0 \in X_0$. □

We iteratively use (1) for $\mathsf{GetBounds}(f, X_0, N)$ to create error bounds over each segment in $z(t)$. At time t_k, the system switches to follow segment S_{k+1}. Thus, the error increases, but this change in error can be bounded. At time t_k, $\xi_{k-1}(t_k) \in \mathcal{B}_{\ell_k}(p_k)$, as shown by Lemma 2, and $\xi_k(t_k) \in \mathcal{B}_{\ell_k}(p_k)$. This only accounts for the error in $\mathcal{W}$. If the remaining states are bounded, then we can use this information to find a bound on the total change in error, allowing us to know the size of this new initial set.

3.2 Synthesizing Reference Trajectories with MILP

Given a lookahead time T_P and a sensing distance d_P, a (T_P, d_P)-PO is queried and returns a prediction of the freespace $\mathbf{FS}_{iT_\mathcal{S}, \xi_i(0)}(t)$ at each $iT_\mathcal{S}$ time. At the same time, the planner tries to find a reference trajectory $z_{i+1} : [0, t_{i+1,N}] \to \mathcal{W}$ to be followed over the time interval $[(i+1)T_\mathcal{S}, (i+2)T_\mathcal{S}]$. We require that z_{i+1} is safe at least over $[0, T_P]$. If the problem is solvable, $z_{i+1}(t)$ is returned and the vehicle follows it. In this section, we consider one such time period where such a reference trajectory is synthesized, so the subscript i is dropped.

A reference trajectory $z(t)$ is considered safe if the vehicle following it remains in $\mathbf{W}(t)$ for all $t \in [0, T_P]$. Recall that $z(t)$ is valid over $t \in [0, t_N]$. Then, to progress to a goal G, we require that $z(t_N) \in G$. Since $z(t)$ is constructed from timed waypoints $\{(p_{i,k}, t_{i,k})\}_{k=0}^N$, it suffices to synthesize these waypoints. In what follows, we show that the problem of finding the desired timed waypoints can be encoded as a mixed integer linear programming (MILP) problem.

Suppose the PO is queried at query time t_q and query state x_q. It returns $\mathbf{FS}_{t_q, x_q}(t)$ as a list of timed obstacles $\{O_i \in \mathcal{W}^C \times \mathbb{R}_{\geq 0}\}_i$. Then, each line segment S_k that connects (p_{k-1}, t_{k-1}) and (p_k, t_k) must be at least ℓ_k away from every obstacle so that the vehicle trajectory will not intersect with the obstacle. As each obstacle $O \in \{O_i\}_i$ is represented using a polytope, it suffices to have $[p_{k-1}, t_{k-1}]$ and $[p_k, t_k]$ be outside of *at least one face* of the polytope for $z(t)$ to avoid O. Note that it may be impossible for the endpoints to lie outside *every face* of the polytope, so we have a disjunction of constraints in (2).

Lemma 3. *Given a polytope $O = Poly(H_O, b_O)$ where $H_O \in \mathbb{R}^{m\times(n+1)}$, an integer $M \gg 0$, and variables $\alpha_j \in \{0,1\}$ for $j = 1, \cdots, m$. Consider the timed waypoints (p_{k-1}, t_{k-1}) and (p_k, t_k) and the segment $S_k(t)$ joining them. If the following conditions hold*

$$\bigwedge_{j=1}^{\mathsf{dP}(H_O)} \begin{pmatrix} -H_O^{(j)} \begin{bmatrix} p_{k-1} \\ t_{k-1} \end{bmatrix} + (b_O^{(j)} + \|H_O^{(j)}\|\ell_k) \leq M(1-\alpha_j) \wedge \\ -H_O^{(j)} \begin{bmatrix} p_k \\ t_k \end{bmatrix} + (b_O^{(j)} + \|H_O^{(j)}\|\ell_k) \leq M(1-\alpha_j) \end{pmatrix}$$
$$\sum_{j=1}^{\mathsf{dP}(H_O)} \alpha_j \geq 1 \tag{2}$$

then $\forall t \in [t_{k-1}, t_k]$, $\mathcal{B}_{\ell_k}(S_k(t)) \cap O = \varnothing$.

Proof. The waypoint (p_k, t_k) is outside and at least ℓ_k away from the j^{th} face of $Poly(H_O, b_O)$ if $(-H_O^{(j)} [p_k \ t_k]^T + (b_O^{(j)} + \|H_O^{(j)}\|\ell_k)) < 0$. If this is the case, then $\alpha_j = 1$ as the MILP constraint becomes

$$\left(-H_O^{(j)} [p_k \ t_k]^T + (b_O^{(j)} + \|H_O^{(j)}\|\ell_k)\right) \leq M(1-1) = 0.$$

If this is not the case, then $\alpha_j = 0$, and the MILP constraint becomes

$$\left(-H_O^{(j)} [p_k \ t_k]^T + (b_O^{(j)} + \|H_O^{(j)}\|\ell_i)\right) \leq M(1-0) = M.$$

This constraint will still hold true when $M \gg 0$.

If both (p_{k-1}, t_{k-1}) and (p_k, t_k) lie at least ℓ_k outside the j^{th} plane, then it is easy to see that $\alpha_j = 1$. If this is only true for one of the waypoints, then $\alpha_j = 0$. Therefore, if $\sum_{j=1}^{\mathsf{dP}(H_O)} \alpha_j \geq 1$, then (p_{k-1}, t_{k-1}) and (p_k, t_k) lie outside at least one plane of the polytope. This means that the line segment $S_k(t)$ is at least ℓ_k away from O. As $S_k(t)$ is at least ℓ_k away from O, then $\mathcal{B}_{\ell_k}(S_k(t)) \cap O = \varnothing$. □

$\mathtt{Avoid}(\ell_k, O, (p_{k-1}, t_{k-1}), (p_k, t_k))$ denotes the constraint in (2). Since all the system trajectories starting from X_0 are contained within $\mathcal{B}_{\ell_k}(S_k(t))$, it follows that the vehicle following S_k will not intersect with O. If these constraints hold true for every $O \in \{O_i\}_i$ then $\mathcal{B}_{\ell_k}(S_k(t)) \subset \mathbf{FS}_{t_q, x_q}(t)$. If this constraint holds true for every $S_k(t)$ for $k = 1, \cdots, N$, then $\mathcal{B}_{\ell(k)}(z(t) \downarrow \mathcal{W}) \subset \mathbf{FS}_{t_q, x_q}(t)$ for all time $t \in [0, T_P]$. Similarly, the reference trajectory must end within the goal shrunken by ℓ_N:

$$\mathtt{Goal}(G, \ell_N, (p_N, t_N)) := \wedge_{j=1}^{\mathsf{dP}(H_G)} H_G p_N \leq b_G^{(j)} - \|H_G\|\ell_N \tag{3}$$

Finally, we introduce maximum velocity constraints by simply requiring that the distance between the waypoints is upper-bounded and the time between them is lower-bounded. In (4), we use the 1-norm since it can be easily turned into linear constraints. For a maximum velocity of v_{max}, we choose $\frac{l_{max}}{\Delta t_{min}} = v_{max}$.

$$\begin{aligned} &\mathtt{RefVel}(l_{max}, \Delta t_{min}, (p_{k-1}, t_{k-1}), (p_k, t_k)) \\ &= |p_k - p_{k-1}|_1 < l_{max} \ \wedge \ t_k - t_{k-1} \geq \Delta t_{min} \end{aligned} \tag{4}$$

Putting it all together, the MILP formulation to find a reference trajectory is ϕ_{pts}, seen in (5).

$$\begin{aligned} \phi_{pts} = \exists (p_0, t_0), (p_1, t_1), \cdots, (p_N, t_N) \quad \text{s.t.} \\ p_0 = z(0) \downarrow \mathcal{W}; t_0 = 0 \\ \bigwedge_{k=1}^{N} \bigwedge_{O \in \{O_i\}_i} \mathtt{Avoid}(O, \ell_k, (p_{k-1}, t_{k-1}), (p_k, t_k)) \\ \bigwedge_{k=1}^{N} \mathtt{RefVel}(l_{max}, \Delta t_{max}, (p_{k-1}, t_{k-1}), (p_k, t_k)) \\ \mathtt{Goal}(G, \ell_N, (p_N, t_N)) \end{aligned} \tag{5}$$

The initial state of the vehicle is $z(0)$. The waypoints that construct a safe reference trajectory are computed by solving for (5), which we call GetWaypoints.

We convert the waypoints returned by GetWaypoints to a PWL reference trajectory, which is implemented in a function called WaypointsToTraj. Due to limited space we omit the details of this procedure in this paper.

3.3 Synthesis algorithm

The full algorithm to synthesize a safe reference trajectory is as follows. While N is less than the maximum number of segments N_{max}, check to see if a sequence of timed waypoints $\{(p_k, t_k)\}_{k=0}^{N}$ that satisfies the constraints ϕ_{pts} exists. If they exist, convert them to a valid reference trajectory $z(t)$ and return $z(t)$. If they do not exist, add another line segment and try again. If no waypoints can be found within N_{max}, the algorithm will return None.

Algorithm 1: Synthesize

Input: $z(0), G, \mathbf{FS}_{t_q,x_q}(t), N_{max}, \{\ell_k\}_{k=1}^{N_{max}}, l_{max}, \Delta t_{min}$

$N \leftarrow 1$
while $N \leq N_{max}$ **do**
  $\{(p_k, t_k)\}_{k=0}^{N} \leftarrow \mathsf{GetWaypoints}(\phi_{pts})$
  try:
    $z(t) \leftarrow \mathsf{WaypointsToTraj}(\{(p_k, t_k)\}_{k=0}^{N})$
    return $z(t)$
  catch $\{(p_k, t_k)\}_{k=0}^{N} = \mathit{None}$:
    $N \leftarrow N + 1$
$z(t) \leftarrow$ None
return $z(t)$

Theorem 2 states that if $T_P > 2T_S$, any $\xi(t)$ following $z(t)$ will be safe for all $t \in [0, T_P]$. The proof follows from Lemmas 2 and 3 and (3), which we omit for space.

Theorem 2. *Given an initial state $z(0)$, error bounds $\{\ell_k\}_{k=1}^{N}$ constructed using (1), and a $[T_P, d_P]$-perception oracle queried at time t_q and state x_q that returns $\mathbf{FS}_{t_q,x_q}(t)$, a goal set G, the reference trajectory $z(t)$ computed using Algorithm 1 ensures that the actual trajectory $\xi(t)$ following $z(t)$ satisfies $\forall t \in [0, T_P]$, $\xi(t) \downarrow \mathcal{W} \in \mathbf{W}(t)$ and $\xi(z.\mathit{ltime}) \downarrow \mathcal{W} \in G$.*

Remark. The assumption in this theorem gives a condition on perception distance d_P. Let the maximum velocity of any obstacle and the vehicle $v_{O,max}$ and v_{max} respectively. The maximum distance both move over T_S is $d_{O,max} = v_{O,max}T_S$ and $d_{max} = v_{max}T_S$ respectively. Thus, we require $d_P \geq d_{O,max} + d_{max}$ to ensure that if new dynamical obstacle appears, it won't cause a collision and can be captured in the next sensing time.

3.4 Full Algorithm with Timing

Given a vehicle model $\mathcal{C}$, some initial reference trajectory $z_0(t)$, the freespace $\mathbf{W}(t)$, a lookahead time T_P, the goal is to find a safe controller from the initial set to the goal set such that for any execution $\alpha = \xi_0(t) \frown \xi_1(t) \cdots$, $\frown \xi_i(t) \in \mathbf{W}(t), \forall t \in [0, T_S], \forall i = 0, 1, \cdots$. For simplicity of illustration, we abuse the notation and use $\mathbf{W}(t)$ to represent freespace at local time t which is reset whenever the system switches modes. This $z_0(t)$ is the reference trajectory that the system is following when the algorithm starts, and assumed to be safe. In our implementation in Section 4, $z_0(t)$ is set so that the vehicle is at a full stop.

The inputs to Algorithm 2 are as follows: (i) the scenario defined by $(\mathbf{W}(t), G)$, (ii) system dynamics f, (iii) time constants T_S and T_P, an (iv) sensing distance d_P, (v) some maximum number of segments N_{max}, and (vi) a safe initial reference trajectory $z_0(t)$ with error bounds $\{\ell_k\}_{k=1}^{N_{max}}$.

Algorithm 2: Main Algorithm

Input: $(\mathbf{W}(t), G)$, f, T_S, T_P, d_P, N_{max}, l_{max}, Δt_{min}, $z_0(t)$, $\{\ell_k\}_{k=1}^{N_{max}}$

1 $i \leftarrow 0$
2 $z_i(t) \leftarrow z_0(t)$
3 **while** $\xi_i(t) \notin G$ ***and*** $z_i(t) \neq \mathit{None}$ **do**
4 $\xi_i([0, T_S]) \leftarrow \mathsf{RunController}(z_i(t), f)$
5 $X_0 \leftarrow \mathsf{CreateInitial}(z_i(T_S), \ell_k)$
6 $\{\ell_k\}_{k=1}^{N_{max}} \leftarrow \mathsf{GetBounds}(f, X_0, N_{max})$
7 $z_{i+1}(t) \leftarrow \mathsf{Synthesize}(z_i(T_S), G, \mathbf{FS}_{iT_S,\xi_i(0)}(t), \{\ell_k\}_{k=1}^{N_{max}}, l_{max}, \Delta t_{min})$
8 $i \leftarrow i + 1$

Algorithm 2 exits if G is reached or if the scenario is unsolvable. From Algorithm 1, the reference trajectory $z_i(t)$ is set to None if the scenario is unsolvable. The vehicle is run according to it's system dynamics f and the reference trajectory $z_i(t)$ in the function $\mathsf{RunController}(z_i(t), f)$ (line 4). While $z_i(t)$ is being followed, $z_{i+1}(t)$ is computed.

In line 5, the initial set is created using the function $\mathsf{CreateInitial}(z_i(T_S), \ell_k)$. Here, ℓ_k is the error bound for the line segment S_k that the system is following at T_S. The initial set for the next period is $X_0 = \mathcal{B}_{\ell_k}(z_i(T_S) \downarrow \mathcal{W})$. In line 6, the error bounds are found using the symmetrical reachable sets from Section 3.1. A new reference controller is found in line 7 and the mode is updated.

Theorem 3. *Given a vehicle model $\mathcal{C}$ with system dynamics f and a scenario defined by $(\mathbf{W}(t), G)$, the sequence of reference trajectories found using Algorithm 2 will result in the execution $\alpha = \xi_0\xi_1 \cdots$, where $\xi_i(t) \downarrow \mathcal{W} \in \mathbf{W}(t), \forall t \in [0, T_S], \forall i = 0, 1, \cdots$.*

We defer the proof to the full version of this paper. This proof uses Theorem 2 and Lemma 2.

Algorithm 2 may not always be able to generate a reference trajectory. In such cases, the algorithm should exit and a safe backup controller should be used. We will not discuss it here, but this is an interesting future research direction.

4. EXPERIMENTS AND DISCUSSIONS

Implementation details. Algorithm 2 is implemented in the FACTEST framework [7] using Gurobi, and in a Python or Webots simulator. We consider scenarios where pedestrians move according to a dataset [26]. We can typically find safe trajectories in less than a second when one exists.

Path complexity is the number of segments in a path. Figure 2 (left) suggests computation time scales linearly with path complexity and increases with the number of obstacles. This reflects the increase in constraints as the number of obstacles and path complexity increases. Algorithm 2 was deployed with the Webots vehicle model in scenarios with 12 and 105 pedestrians. We varied the lookahead (T_P), synthesis period (T_S), and maximum velocity bounds. The results are summarized in Table 1.

We note the following: 1) As the maximum velocity of the vehicle decreases, the safety margin (min. dist) increases. This makes sense, as the vehicle can maintain higher separation. 2) As synthesis time increases, the safety margin decreases, since re-planning occurs less often. Issues may arise if pedestrians can suddenly enter the vicinity of the

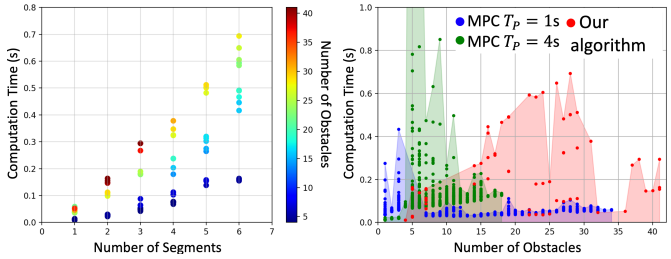

Fig. 2. Computation time comparisons. Left: effect of path complexity and number of obstacles on running time. Right: comparison of MPC ($T_P = 1s$ blue, $T_P = 4s$ green) and Algorithm 2 (red).

Ppl	T_P	T_S	d_{sense}	t_{min}	l_{max}	Min dist	Completion
12	4	1	10	5	10	1.405	17.312
12	4	1	10	2.5	10	2.256	10.912
12	4	1	10	5	5	3.501	25.184
12	4	0.5	10	5	10	1.390	15.36
12	4	2	10	5	10	1.020	19.776
105	4	1	10	5	10	1.622	15.584
105	4	2	10	5	10	1.796	15.456
105	4	2	10	6	12	0.766	18.528
105	4	2	10	4	8	1.256	17.344

Table 1. Online synthesis results. The parameters are: number of pedestrians (Ppl), T_S, T_P, t_{min}, l_{max}, the minimum safety margin (Min dist), and the completion time (Completion). The units of time and distance are seconds and meters.

vehicle, though we can add the condition that a new path should be synthesized if an unexpected obstacle appears. 3) The safety margin decreases as l_{max} increases. This is due to the MILP solver defaulting to reference trajectories that lie close to the obstacles and can be overcome by adding a cost function that rewards a larger safety margin.

Comparison with MPC. In this comparison, we specify a high-level path, and MPC computes the optimal inputs to avoid obstacles. We compare in scenarios with 4, 76, and 105 pedestrians and with two different T_P. Computation time results are seen in Figure 2 (right). While Algorithm 2 may not be the clear "winner" we note where it can be beneficial. MPC is faster when T_P is "just right". If T_P is too large, or too many obstacles are detected , MPC times out (see $T_P = 4s$). Therefore, Algorithm 2 is beneficial in scenarios where a large T_P is desirable.

Conclusions. We presented an abstraction for perception (PO) and based on that we developed a synthesis algorithm for unknown environments. Our analysis helps to identify sufficient conditions on perception, update period, and maximum velocity of the vehicle that assure safety. The open prototype implementation shows promise. In the future, we plan to compare with tools like RTD and FaSTrack. Some interesting directions are to use learned tracking controllers with complex plant models, or explore conditions under which progress can be guaranteed.

REFERENCES

[1] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *ECC*, pages 3420–3431. IEEE, 2019.

[2] M. M. G. Ardakani, B. Olofsson, A. Robertsson, and R. Johansson. Real-time trajectory generation using model predictive control. In *CASE*, pages 942–948. IEEE, 2015.

[3] H. Badino, U. Franke, and R. Mester. Free space computation using stochastic occupancy grids and dynamic programming. In *ICCV*, volume 20. Citeseer, 2007.

[4] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin. An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In *CDC*, pages 1758–1765. IEEE, 2019.

[5] A. Bemporad, F. Borrelli, and M. Morari. Model predictive control based on linear programming - The explicit solution. *IEEE TAC*, 47(12):1974–1985, 2002.

[6] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: A verification tool for stateflow models. In *TACAS*, pages 68–82. Springer, 2015.

[7] C. Fan, K. Miller, and S. Mitra. Fast and guaranteed safe controller synthesis for nonlinear vehicle models. In *CAV*, pages 629–652. Springer, 2020.

[8] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin. Fastrack: A modular framework for fast and guaranteed safe motion planning. In *CDC*, pages 1517–1522. IEEE, 2017.

[9] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. In *CVPR*, June 2020.

[10] Z. Huang, A. Hasan, K. Shin, R. Li, and K. Driggs-Campbell. Long-term pedestrian trajectory prediction using mutable intention filter and warp lstm. *IEEE RA-L*, 2020.

[11] S. Jha, V. Raman, D. Sadigh, and S. A. Seshia. Safe autonomy under perception uncertainty using chance-constrained temporal logic. *Journal of Automated Reasoning*, 60(1):43–62, 2018.

[12] Y. Kantaros, M. Malencia, and G. J. Pappas. Reactive temporal logic planning for multiple robots in unknown occupancy grid maps. *arXiv preprint arXiv:2012.07912*, 2020.

[13] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE T-RO*, 25(6):1370–1381, 2009.

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

[15] D. Q. Mayne. Model predictive control: Recent developments and future promise. In *Automatica*, volume 50, pages 2967–2986. Pergamon, 2014.

[16] S. Mouelhi, A. Girard, and G. Gössler. CoSyMA: A tool for controller synthesis using multi-scale abstractions. In *HSCC*, pages 83–88. ACM, 2013.

[17] B. Qi. *Building DryVR: A verification and controller synthesis engine for cyber-physical systems and safety-critical autonomous vehicle features.* PhD thesis, UIUC, 2018.

[18] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia. Reactive synthesis from signal temporal logic specifications. In *HSCC*, pages 239–248. ACM, 2015.

[19] S. Richter, C. N. Jones, and M. Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE TAC*, 57(6):1391–1403, 2011.

[20] P. Roy, P. Tabuada, and R. Majumdar. Pessoa 2.0: A controller synthesis tool for cyber-physical systems. In *HSCC*, pages 315–316. ACM, 2011.

[21] M. Rungger and M. Zamani. Scots: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th HSCC*, pages 99–104, 2016.

[22] G. Russo and J.-J. E. Slotine. Symmetries, stability, and control in nonlinear systems and networks. *Physical Review E*, 84(4):041929, 2011.

[23] H. Sibai, N. Mokhlesi, C. Fan, and S. Mitra. Multi-agent safety verification using symmetry transformations. In *TACAS*, pages 173–190. Springer, 2020.

[24] S. Vaskov, S. Kousik, H. Larson, F. Bu, J. Ward, S. Worrall, M. Johnson-Roberson, and R. Vasudevan. Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments. *arXiv preprint arXiv:1902.02851*, 2019.

[25] M. Vitus, V. Pradeep, G. Hoffmann, S. Waslander, and C. Tomlin. Tunnel-milp: Path planning with sequential convex polytopes. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7132, 2008.

[26] D. Yang, L. Li, K. Redmill, and Ü. Özgüner. Top-view trajectories: A pedestrian dataset of vehicle-crowd interaction from controlled experiments and crowded campus. In *2019 IEEE (IV)*, pages 899–904. IEEE, 2019.

[27] M. N. Zeilinger, C. N. Jones, and M. Morari. Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization. *IEEE TAC*, 56(7):1524–1534, 2011.