Provably Safe and Scalable Multi-Vehicle Trajectory Planning

Somil Bansal\textsuperscript{1}, Mo Chen\textsuperscript{2}, Ken Tanabe\textsuperscript{3}, Claire J. Tomlin\textsuperscript{1}

Abstract—In [1], the Sequential Trajectory Planning (STP) method was proposed, which allows multiple vehicle trajectory planning to be done with a computation complexity that scales linearly with the number of vehicles. However, the STP computation is not tractable for large-scale systems using the currently available tools. In this work, we introduce BEACLS, a C++-based reachability toolbox that can leverage GPU parallelization to improve computation speed of HJ reachability by nearly 100 times compared to existing MATLAB implementations. We then combine BEACLS with STP for safe, large-scale multiple UAV planning in a city environment and a multi-city environment. We show that intuitive multi-lane structures naturally emerge, and that the size of disturbances and vehicle density are primary factors determining the number and width of lanes. We also extend the STP method to safely account for an adversarial intruder during trajectory planning. In the proposed formulation, the number of vehicles that needs to replan is a design parameter that can be chosen based on the computational resources available during run time. The proposed formulation along with BEACLS provide both an algorithm as well as an efficient computational tool for resilient, large-scale multiple vehicle trajectory planning.

I. INTRODUCTION

Recently, there has been an immense surge of interest in the use of unmanned aerial systems (UASs) for civil applications [2]–[6], such as package delivery, precision agriculture and surveillance [7], [8]. These applications will involve unmanned aerial vehicles (UAVs) flying in urban environments, potentially in close proximity to humans, other UAVs, and other important assets. As a result, new scalable ways to organize an airspace are required in which potentially thousands of UAVs can fly together [9], [10].

For this to be successful, a safe, multiple vehicle trajectory planning algorithm needs to be available. Current solutions propose to partition and reserve regions of airspace in space and time [9], [10], yet a way to do this flexibly and dynamically which corresponds to the demand is being sought after. Previous work has addressed this problem under different assumptions. In some studies, specific control strategies for the vehicles are derived using Voronoi paths [11] or mixed-integer programming-based solutions [12]. These approaches tend to be computationally intractable as the number of agents grow. Approaches involving induced velocity obstacles [13]–[16], virtual potential fields [17], [18], and analytical PDE-based methods [19] to maintain collision avoidance are also available, yet these have not considered planning and collision avoidance simultaneously. Other related work is in the collision avoidance problem without path planning. These results include those that assume the system has a linear model [20]–[22], rely on a linearization of the system model [23]–[25], assume a simple positional or planar state space [26], [27], and many others [28]–[50]. Methods which do combine both planning and collision avoidance typically make simplifying assumptions about the vehicle dynamics [51]–[53].

Other works take a distributed or decentralized approach wherein the trajectory planning problem is divided in a series of smaller sub-problems [34]–[39]. The typical motivation for this line of work is that each of these sub-problems can be solved much faster than solving one large centralized problem. However, most of these works focus on planar motion with simplified vehicle dynamics. These approaches also typically struggle with enforcing global planning constraints [40]. Other works achieve coordination between vehicles by enforcing temporal constraints on the speed profiles or path characteristics [41], [42], effectively decoupling the planning state-space.

However, scalable methods to flexibly plan provably safe and dynamically feasible trajectories without making strong assumptions on the vehicles’ dynamics and other vehicles’ motion are still lacking. Moreover, any trajectory planning scheme that addresses collision avoidance must also guarantee both goal satisfaction and safety of UAVs despite disturbances and communication faults [10]. Furthermore, unexpected scenarios such as UAV malfunctions or even UAVs with malicious intent need to be accounted for.

Hamilton-Jacobi (HJ) reachability-based methods [43]–[48] could be particularly suitable for this problem as formal guarantees are provided without making such strong assumptions about the other vehicles’ motion. In this context, one computes the reach-avoid set, defined as the set of states from which the system can be driven to a desired configuration or trajectory while satisfying (potentially) time-varying state constraints at all times. A major practical appeal of this approach stems from the availability of modern numerical tools which can compute various definitions of reachable sets [49]–[52]. These numerical tools have been successfully used to solve a variety of differential games, trajectory planning problems, and optimal control problems [53]–[56]. However, reachable set computations involve solving a HJ partial differential equation (PDE) or variational inequality (VI) on a grid representing a discretization of the state space, resulting in an exponential scaling of computational complexity with respect to the system dimension. Therefore, reachability analysis or other dynamic programming-based methods alone are not suitable for the problem considered here.

To overcome this problem, the priority-based Sequential Trajectory Planning (STP) method has been proposed [57], [58]. In this context, higher-priority vehicles plan their trajectories without taking into account the lower-priority vehicles, and lower-priority vehicles treat higher-priority vehicles as obstacles. Under this assumption, time-varying formulations of reachability [46], [48] can be used to obtain the optimal and provably safe trajectories for each vehicle.
sequentially, starting from the highest-priority vehicle. Thus, the curse of dimensionality is overcome at the cost of a structural assumption, under which the computation complexity scales just linearly with the number of vehicles. In addition, such a structure has the potential to flexibly divide up the airspace for the use of many UAVs and allows flexible multi-vehicle trajectory planning and re-planning. Practically, different economic mechanisms can be used to establish a priority order. One example could be first-come-first-serve mechanism, as highlighted in NASA’s concept of operations for UAS traffic management [10].

Previous work [1], [58] respectively extends the original STP method [57] to scenarios in which disturbances and adversarial intruders are present in the system. However, these have not been sufficient to allow STP used for large-scale trajectory planning, involving, for example hundreds to thousands of vehicles. In this paper, we focus on these computational aspects of multi-vehicle trajectory planning. Our contributions in this paper are three-fold:

1) We present BEACLS, a C++-based toolbox that parallelizes the numerical algorithms for solving the HJ PDE using graphics processing units (GPUs). This speeds up the computation time by nearly 100 times compared to state-of-the-art MATLAB implementations. GPU parallelization allows us to tractably solve many HJ PDEs. We combine BEACLS with the STP method to obtain trajectories of 200 UAVs flying over a large multi-city region, allowing provably safe trajectory planning in the real world for large-scale systems.

2) Through large-scale multiple UAV simulations, we demonstrate how different types of space-time trajectories emerge naturally for different disturbance conditions and other problem parameters. These emerging behaviors, while being provably safe, are also intuitive and would facilitate human monitoring and airspace design.

3) We propose a new method to make the airspace more resilient to intruder vehicles. Compared to the method in [1], in which a single intruder may cause all vehicles to re-plan their trajectories, our proposed method is more practical and resilient: it allows one to set a limit on the maximum number of vehicles that may need to re-plan due to the presence of an intruder.

The rest of the paper is organized as follows. In Section II we present an overview of time-varying reachability and basic STP algorithms in [57], [58]. In Section III we present our new, more resilient intruder rejection method. Section IV introduces BEACLS, our GPU-parallelized implementation of the reachability toolbox used to run our simulations. In Section V and VI we present our large-scale multiple UAV simulations results in a city environment and a multi-city environment respectively. Finally, in Section VII we present the simulation results for the multiple UAV city environment setting in the presence of an intruder.

II. BACKGROUND

In this section, we summarize first the basics of time-varying reachability analysis [48], and then the STP algorithm in [58].

A. Time-Varying Hamilton-Jacobi Reachability

Due to its flexibility with respect to time-varying systems, and optimality of the solution it provides, Hamilton-Jacobi reachability analysis [44], [46], [48], [59] is used to analyze the STP problem. This involves computing either a backward reach-avoid set (BRS) \( V \), a forward reach-avoid set (FRS) \( W \), or a sequence of BRSs and FRSs, given some target set \( L \), time-varying obstacle \( G(t) \) which captures trajectories of higher-priority vehicles, and the Hamiltonian function \( H \) which captures the system dynamics as well as the roles of the control and disturbance. Intuitively, the BRS represents the set of states from which a vehicle can reach the target set \( L \) while avoiding the obstacles \( G(t) \), and the FRS represents the sets of states that can be reached starting from the target set \( L \). For details on HJ reachability and its application in STP, we encourage the reader to refer to [1], [59].

In this paper, the BRS \( V \) in a time interval \([t, t_f]\) (for fixed \( t_f \)) or FRS \( W \) in a time interval \([t_0, t]\) (for fixed \( t_0 \)) will be denoted by \( V(t, t_f) \) or \( W(t_0, t) \) respectively. The BRS \( V(t, t_f) \) can be computed by solving the following final value HJ variational inequality:

\[
\max \left \{ \min \{ D_t V(t, x) + H(t, x, \nabla V(t, x)), l(x) - V(t, x) \}, \quad g(t, x) - V(t, x) \right \} = 0, \quad t \leq t_f
\]

\[
V(t_f, x) = \min \{ l(x), -g(t_f, x) \}
\]

In a similar fashion, the FRS \( W(t_0, t) \) can be computed by solving the following initial value HJ PDE:

\[
D_t W(t, x) + H(t, x, \nabla W(t, x)) = 0, \quad t \geq t_0
\]

\[
W(t_0, x) = l(x)
\]

In both (1) and (2), the function \( l(x) \) is the implicit surface function representing the target set \( L = \{ x : l(x) \leq 0 \} \). Similarly, the function \( g(t, x) \) is the implicit surface function representing the time-varying obstacles \( G(t) = \{ x : g(t, x) \leq 0 \} \). The BRS \( V(t, t_f) \) and FRS \( W(t_0, t) \) are given by

\[
V(t, t_f) = \{ x : V(t, x) \leq 0 \}
\]

\[
W(t_0, t) = \{ x : W(t, x) \leq 0 \}
\]

Optimizing the Hamiltonian \( H \) gives the optimal control \( u^*(t, x) \) and optimal disturbance \( d^*(t, x) \), once \( V \) is determined.

To solve the PDEs in (1) and (2), the level set toolbox has been developed [52]. The toolbox is implemented in MATLAB and is equipped to solve any final-value HJ PDE using the Lax-Friedrichs numerical scheme [60]. Since different reachable set computations can be ultimately posed as solving a final-value HJ PDE, the level set toolbox is fully equipped to compute various types of reachable sets. This toolbox has been further augmented by the Hamilton-Jacobi optimal control toolbox (or helperOC) to facilitate reachable set computations for a variety of nonlinear systems. A quick-start guide to these tools has been presented in [59]. In this paper, we build upon these works to develop a new toolbox that can leverage the modern computational tools such as GPUs for significantly improving the computation speed of reachable sets.

B. Sequential Trajectory Planning Under Disturbances

Consider \( N \) vehicles \( Q_i, i = 1, \ldots, N \) (also denoted as STP vehicles) which participate in the STP process. We assume their dynamics are given by
\[
\dot{x}_i = f_i(x_i, u_i, d_i), \quad t \leq t_i^{\text{STA}} \quad (4) \\
u_i \in U_i, d_i \in D_i, i = 1, \ldots, N \quad (5)
\]

where \(x_i \in \mathbb{R}^{n_i}, u_i \subset U_i \) and \(d_i \subset D_i \) respectively represent the state, control, and disturbance experienced by vehicle \(Q_i \). We partition the state \(x_i \) into the position component \(p_i \in \mathbb{R}^{n_p} \) and the non-position component \(h_i \in \mathbb{R}^{n_i-n_p}: x_i = (p_i, h_i) \). We will use the sets \(U_i, D_i \) to respectively denote the set of allowable controls and disturbances at each time \(t \), and \(\mathbb{U}_i, \mathbb{D}_i \) to respectively denote the set of allowable functions from which the control and disturbance functions \(u_i(\cdot), d_i(\cdot) \) are drawn.

Each vehicle \(Q_i \) has initial state \(x_i^0 \), and aims to reach its target \(L_i \) by some scheduled time of arrival \(t_i^{\text{STA}} \). The target in general represents some set of desirable states, for example the destination of \(Q_i \). On its way to \(L_i \), \(Q_i \) must avoid a set of static obstacles \(O_{\text{static}} \subseteq \mathbb{R}^{n_i} \), which could represent any set of states, such as positions of tall buildings, that are forbidden. Each vehicle \(Q_i \) must also avoid the danger zones with respect to every other vehicle \(Q_j, j \neq i \). For simplicity, we define the danger zone of \(Q_i \) with respect to \(Q_j \) to be

\[
Z_{ij} = \{(x_i, x_j) : \|p_i - p_j\|_2 \leq R_c\}. \quad (6)
\]

Vehicles \(Q_i \) and \(Q_j \) are said to have collided if \((x_i, x_j) \in Z_{ij} \).

Given the set of STP vehicles, their targets \(L_i \), the static obstacles \(O_{\text{static}} \), and the vehicles’ danger zones with respect to each other \(Z_{ij} \), the goal of STP is as follows. For each vehicle \(Q_i \), synthesize a controller which guarantees that \(Q_i \) reaches its target \(L_i \) at or before the scheduled time of arrival \(t_i^{\text{STA}} \), while avoiding the static obstacles \(O_{\text{static}} \), and the danger zones with respect to all other vehicles \(Z_{ij}, j \neq i \). In addition, we would like to obtain the latest departure time \(t_i^{\text{LDT}} \) such that \(Q_i \) can still arrive at \(L_i \) on time.

Due to the high dimensionality of the multiple vehicle joint state-space, a direct dynamic programming-based solution is intractable. Therefore, the authors in \[1, 57] proposed to assign a priority to each vehicle, and perform STP given the assigned priorities. Without loss of generality, let \(Q_k \) have a higher-priority than \(Q_i \) if \(k < i \). Under the STP scheme, higher-priority vehicles can ignore the presence of lower-priority vehicles, and perform trajectory planning without taking into account the lower-priority vehicles. A lower-priority vehicle \(Q_i \), on the other hand, must ensure that it does not enter the danger zones of the higher-priority vehicles \(Q_k, k < i \). Each higher-priority vehicle \(Q_k \) induces a set of time-varying obstacles \(O_k(t_i) \), which represents the possible states of \(Q_i \) such that a collision between \(Q_i \) and \(Q_k \) could occur. It is straightforward to see that if each vehicle \(Q_i \) is able to plan a trajectory that takes it to \(L_i \) while avoiding the static obstacles \(O_{\text{static}} \), and the danger zones of higher-priority vehicles \(O_k(\cdot), k < i \), then the set of STP vehicles \(Q_i, i = 1, \ldots, N \) would all be able to reach their targets safely.

To obtain a trajectory of vehicle \(Q_i \) that is robust to disturbances, each vehicle uses a reduced control authority \(U_i^p \subset U_i \) for planning and goal satisfaction, reserving the remaining control authority for disturbance rejection. During planning, possible reference trajectories are computed that the vehicle can track in the absence of disturbances. The reference trajectories can be obtained using the following system dynamics:

\[
\dot{x}_{r,i} = f_i(x_{r,i}, u_{r,i}), \quad t \leq t_i^{\text{STA}} \quad (7) \\
u_{r,i} \in \mathbb{U}_i, i = 1, \ldots, N,
\]

where the particular reference trajectory \(x_{r,i}(t) \) is obtained by choosing a reference control function \(u_{r,i} \in \mathbb{U}_i \). Next, a bound on the maximum deviation from the reference trajectory due to the disturbances is computed. Thus, instead of directly accounting for disturbances when planning trajectories, one first augments the time-varying obstacle by this bound to obtain the augmented obstacles, which ensures that \(Q_i \) will not collide with the obstacles in the presence of disturbances. One then reduces the size of \(L_i \) by the same amount to obtain \(L_i \), which ensures that the vehicle safely reaches its goal despite any trajectory tracking errors resulting from disturbances. The reader is encouraged to refer to \[1 \] for details about the STP algorithm that is robust to disturbances.

III. RESPONSE TO INTRUDERS

We now propose a novel algorithm that can account for an intruder vehicle \(Q_l \) in the system. An intruder vehicle may simply be a non-participating vehicle that could accidentally collide with other vehicles, or it could be one with malicious intent. This general definition of intruder allows us to develop algorithms that can also account for vehicles who are not communicating with the STP vehicles or do not know about the STP structure. In this section, our goal is to design a control policy for each vehicle that ensures separation with the intruder and other STP vehicles, and a successful transit to the destination.

A. Problem Setup and Solution Approach for Intruder Avoidance

In general, the effect of intruders on vehicles in structured flight can be unpredictable, since the intruders in principle could be adversarial in nature, and the number of intruders could be arbitrary. Therefore, to make our analysis tractable, we make the following two assumptions.

Assumption 1: At most one intruder affects the STP vehicles at any given time. The intruder is removed after a duration of \(t_i^{\text{LNT}} \).

This assumption can be valid in situations where intruders are rare, and that some fail-safe or enforcement mechanism exists to force the intruder out of the planning space. Practically, over a large region of the unmanned airspace, this assumption implies that there would be one intruder vehicle per “planning region”. Each planning region would perform STP independently from the others. The entire large region would be composed of several planning regions, and possibly several intruder vehicles. Note that we do not pose any restriction on the time at which intruder appears in the system; we only assume that once the intruder appears, it stays for a maximum duration of \(t_i^{\text{LNT}} \).

Assumption 2: The dynamics of the intruder are known and given by \(\dot{x}_I = f_I(x_I, u_I, d_I) \).

Assumption 2 is required for HJ reachability analysis. In situations where the dynamics of the intruder are not known exactly, a conservative model of the intruder may be used instead. We also denote the initial state of the intruder as \(x_I^0 \). Note that we only assume that the dynamics of the intruder
are known, but its initial state $x^0_i$, control $u_I$ and disturbance $d_I$ it experiences are unknown.

Under the above assumptions, we present an intruder avoidance algorithm that ensures that only a small and fixed number of vehicles, $\hat{k}$, needs to replan their trajectories due to the intruder, regardless of the total number of vehicles, resulting in a constant replanning time. This is often an important consideration for real-world systems, since the replanning needs to be done during the runtime. Moreover, $\hat{k}$ is a design parameter, which can be chosen based on the resources available during run time.

Our algorithm consists of two phases: the planning phase and the replanning phase. In the planning phase, it is ensured that any two vehicles are sufficiently far enough from each other such that an intruder can be in the vicinity of at most $\hat{k}$ vehicles within the duration of $t_{LAT}$. This division of the flight space guarantees that the intruder can affect the trajectory of at most $\hat{k}$ vehicles despite its best efforts, resulting in at most $\hat{k}$ replanning problems. In the replanning phase, we replan the trajectories of the affected vehicles by assigning the affected vehicles the lowest priority and using the STP algorithm presented in Section II-B.

To design the flight space during the planning phase, we compute a separation region for each vehicle such that the vehicle needs to react to the intruder if and only if the intruder is inside this separation region. We then compute a buffer region between the separation regions of any two vehicles such that the intruder requires at least a duration of $t_{BRD}$ to travel through this region. Thus, within the duration of $t_{LAT}$, the intruder can force at most $\hat{k}$ STP vehicles to deviate from their trajectories.

Remark 1: For brevity, we present all our analyses and results in this section assuming that all STP vehicles have same dynamics and control constraints, and the intruder has the same state space as STP vehicles. However, the analysis to follow is more general and can easily be extended to the scenarios where the above assumptions do not hold. We refer the interested readers to the extended version of this section [61] for more details.

B. Computation of Separation and Buffer Regions

The separation region, $S_i(t)$, denotes the set of states of the intruder for which the vehicle $Q_i$ is forced to apply an avoidance maneuver. $S_i(t)$ is given by the set of states from which the joint states of $Q_I$ and $Q_i$ can enter the danger zone $Z_{ij}$ despite the best efforts of $Q_i$ to avoid $Q_I$. We define the relative dynamics between $Q_I$ and $Q_i$:

$$x_{Ii} = x_I - x_i, \quad \dot{x}_{Ii} = f_I(x_{Ii}, u_I, u_I, d_i, d_I)$$

In relative state space, the set of potentially unsafe states is given by the backward reachable set $\cal{V}_i^A(\tau, t_{LAT})$, $\tau \in [0, t_{LAT}]$:

$$\cal{V}_i^A(\tau, t_{LAT}) = \{ y : \forall u_i(\cdot) \in U_I, \exists u_I(\cdot) \in U_I, \exists d_i(\cdot) \in D_i, \exists d_I(\cdot) \in D_I, x_{Ii}(\cdot) \text{ satisfies } (8) \},$$

$$s \in [\tau, t_{LAT}], x_{Ii}(s) \in L_i^\tau, x_{Ii}(\tau) = y,$$

$$L_i^\tau = \{ x_{Ii} : \| p_{Ii} \| \leq R_c \}.$$  

$\cal{V}_i^A$ can be computed using the HJI-VI in (11) with Hamiltonian

$$H_i^A(x_{Ii}, \lambda) = \max_{u_i \in U_i, u_I \in U_I, d_i \in D_i} \min_{d_I \in D_I} \lambda \cdot f_I(x_{Ii}, u_I, u_I, d_i, d_I).$$

Finally, we compute the set of states from which $Q_i$ can collide with any static obstacle, $\cal{O}_i^{static}$, this set is given by the BRS $\cal{V}_i^B(t, t + t_{LAT})$, representing the set of all states of $Q_i$ at time $t$ that can lead to a collision with a static obstacle for some time $\tau > t$ for some control strategy of $Q_I$. Thus, the overall set of states that $Q_i$ needs to avoid is:

$$\cal{G}_i(t) = \cal{V}_i^S(t, t + t_{LAT}) \cup \bigcup_{j=1}^{j-1} \cal{O}_j^I(t) \cup \bigcup_{j=1}^{j-1} B_{ij}(t).$$
Given $G_i(t)$, we compute a BRS $V^\text{PP}(t, t^\text{STA}_i)$ for trajectory planning that contains the initial state of $Q_i$ and avoids $G_i(t)$:

$$V^\text{PP}(t, t^\text{STA}_i) = \{ y : \exists u_i(\cdot) \in U_i, \forall d_i(\cdot) \in D_i, \ x_i(\cdot) \text{ satisfies (4)}, \ \forall s \in [t, t^\text{STA}_i], x_i(s) \notin G_i(s), \ \exists s \in [t, t^\text{STA}_i], x_i(s) \in L_i, x_i(t) = y\},$$

with Hamiltonian

$$H^\text{PP}(x_i, \lambda) = \min_{u_i \in U_i, d_i \in D_i} \max \lambda \cdot f_i(x_i, u_i, d_i).$$

$V^\text{PP}(\cdot)$ ensures goal satisfaction for $Q_i$ in the absence of intruder. The goal satisfaction controller is given by:

$$u^\text{PP}_i(t, x_i) = \arg \min_{u_i \in U_i, d_i \in D_i} \max \lambda \cdot f_i(x_i, u_i, d_i)$$

When intruder is not present in the system, $Q_i$ applies the control $u^\text{PP}$ and we get the nominal trajectory of $Q_i$. Once intruder appears in the system, $Q_i$ applies the avoidance control $u^\lambda_i$ and hence might deviate from its nominal trajectory. The overall control policy for avoiding the intruder and collision with other vehicles is thus given by:

$$u^*_{\lambda}(t, x_i, I_i) = \begin{cases} u^\text{PP}_i(t, x_i) & x_i(t) \notin V^\lambda_i(t, t^\text{LAT}) \\ u^\lambda_i(t, I_i) & \text{otherwise} \end{cases}$$

If $Q_i$ starts within $V^\text{PP}$ and uses the control $u^*_{\lambda}$, it is guaranteed to avoid collision with the intruder and other STP vehicles, regardless of the control strategy of $Q_i$. Finally, since we use separation and buffer regions as obstacles during the trajectory planning of $Q_i$, it is guaranteed that atmost $k$ vehicles are forced to deviate from their path due to the intruder.

**D. Replanning after Intruder Avoidance**

After the intruder disappears, we have to replan the trajectories of the vehicles that were affected by $Q_i$. Let $N^\text{RP}$ denote the set of all vehicles for whom replanning is required. $N^\text{RP}$ can be obtained by checking if a vehicle $Q_i$ applied any avoidance control during $[t, t + t^\text{LAT}]$, i.e.,

$$N^\text{RP} = \{ Q_i : \exists \ell \in [t, t + t^\text{LAT}], x_i(\ell) \in V^\lambda_i(t, t^\text{LAT}) \},$$

where $\ell$ denote the time at which the intruder was first detected in the system. Recall that due to the presence of separation and buffer regions, at most $k$ vehicles can be affected by $Q_i$, i.e., $|N^\text{RP}| \leq k$. The replanning for the vehicles in $N^\text{RP}$ can be performed by assigning them the lowest priority and using the nominal STP algorithm presented in Section II-B. Note that $k$ can be picked beforehand based on the available computation resources during run-time so that this replanning can be done in real time.

**Remark 2:** In this work, we assume worst-case scenarios in terms of the behavior of the intruder, the effect of disturbances, and the planned trajectories of each STP vehicle. Consequently, we are able to guarantee safety and goal satisfaction of all vehicles in all possible scenarios given the bounds on intruder dynamics and disturbances. To achieve denser operation of STP vehicles, known information about the intruder, disturbances, and specifics of STP vehicle trajectories may be incorporated; however, we defer such considerations to future work.

---

**Fig. 1:** Correspondence between the MATLAB implementation of Level Set Toolbox and BEACLS. In the case of this paper, the block called “User’s Model and App.” would be an implementation of an STP algorithm.

**Fig. 2:** BEACLS splits multi-dimensional arrays representing the value function $V(t, x)$ into appropriate overlapping chunks according to processor configuration. Numerical gradients and the Hamiltonian values of each chunk are computed in parallel to produce the updated value function at the previous time step (in the case of backward reachability). Finally, the chunks are combined together to form the updated value function over the entire computational domain.

**IV. THE BERKELEY EFFICIENT API IN C++ FOR LEVEL SET METHODS (BEACLS)**

The STP algorithm requires an efficient computation of reachable sets. Even though Level set toolbox [52] and helperOC library [62] together provide a computational tool to solve a HJ PDE (see Section II-A for a brief description of these libraries), these have not been sufficient to allow STP used for large-scale trajectory planning, involving, for example hundreds to thousands of vehicles. We propose BEACLS, the Berkeley Efficient API in C++ for Level Set Methods, which is an efficient implementation of the level set toolbox and helperOC library that allows much faster computation than the existing MATLAB implementation. In particular, it is able to parallelize using GPUs to speed up computations by nearly 100 times.

In place of the native MATLAB functionalities and toolboxes used by the level set toolbox and helperOC library, BEACLS uses several open source C++ libraries. These libraries include MATIO for loading and saving .mat files which store value functions and trajectories, OpenCV for visualizations, and several others for computation, as depicted in Fig. 1.

The key feature that allows parallelized computation is the splitting of multi-dimensional grids into smaller chunks. These smaller chunks can appropriately fit into CPU cache and GPU memory. Fig. 2 depicts the general procedure of splitting up a grid, computing partial derivatives, and recombining the chunks. A few other notable features are as follows:

- To allow parallel computation, the chunks have sufficient overlap to allow correct gradient computations at the boundaries of each chunk.

---

**Diagram:**

[Diagram showing the correspondence between MATLAB implementation and BEACLS, with arrows and labels indicating flow and communication.]
Unlike the MATLAB libraries, the computational grid is not explicitly stored in a multi-dimensional array, an implementation that is necessary to improve computation time through vectorized computations in MATLAB. Instead, BEACLS stores only the value function (and not the grid) in multi-dimensional arrays, which greatly reduces memory usage.

As with the level set toolbox [52], the Hamiltonian is approximated using the Lax-Friedrichs scheme, which requires estimates of dissipation terms required for numerical stability. The dissipation term is based on the system dynamics over the entire computational grid. Since BEACLS does not explicitly store the computational grid, the dissipation term is computed approximately. This implementation works well in our numerical simulations.

Figure 3 shows at a glance the computation times for implementations of the level set methods used to solve the HJI PDE in (1), which provides the BRS and optimal trajectories for each vehicle. The software implementations are the MATLAB level set toolbox in [52], a C++ implementation of [52] in BEACLS, and a Compute Unified Device Architecture (CUDA) implementation of [52] and [62] in BEACLS with the C++ interface. The MATLAB and C++ implementations are run on a Core i7-5820K CPU, and the CUDA implementation was run on one or two Geforce GTX Titan X GPUs. When run on two GPUs, BEACLS is approximately 100 times faster compared to the MATLAB implementation.

This improvement in computation time greatly facilitates case studies such as city-level airspace design, which may involve testing different initial and goal positions at different vehicle densities and wind speeds, as we demonstrate in Section VI. Trajectory planning with 50 vehicles takes less than 2 minutes, compared to 27 minutes for a non-CUDA C++ implementation or 2.8 hours for the MATLAB implementation.

For even larger-scale case studies, a GPU-parallelized implementation such as BEACLS may be necessary to keep the computation tractable. For example, in the study presented in Section VII involving multiple cities and 200 vehicles, computation for each vehicle was approximately 4 minutes on average using two GPUs. This is because a much finer grid was needed to maintain positional accuracy over a much larger geographical area. The entire simulation took approximately 13.3 hours. Extrapolating the computation time comparison in Fig. 3 even a non-CUDA C++ implementation would take prohibitively long – approximately 10 days.

BEACLS, along with more detailed documentation, is available at the BEACLS website.

V. LARGE-SCALE MULTIPLE UAV SIMULATIONS: CITY ENVIRONMENT

We now combine BEACLS with the STP algorithm for the safe trajectory planning for a 50-UAV system in which the vehicles are flying over the city of San Francisco. This setup can be representative of many UAV applications, such as package delivery, aerial surveillance, etc. Using this simulation, we investigate the resulting trajectories of vehicles as a function of the amount of traffic and wind speed. Videos of the city environment simulation with various vehicle densities and wind speeds can be found on YouTube.

A. Setup

We grid the City of San Francisco (SF) in California, USA, and use it as our position space, as shown in Fig. 4. The origin point for the vehicles is denoted by the blue star. This origin point may represent an exit of an air highway connecting SF to other cities in the Bay Area. In general there may be
multiple origin points; we will demonstrate this other case in Section VI.

Four different areas in the city are chosen as the destinations for the vehicles. Mathematically, the target sets $L_i$ of the vehicles are circles of radius $r$ in the position space, i.e., each vehicle is trying to reach some desired set of positions. In terms of the state space $x_i$, the target sets are defined as

$$L_i = \{ x_i : \|p_i - c_i\|_2 \leq r \}$$

where $c_i$ are centers of the target circles. In this simulation, we use $r = 100$ meters. The four targets are represented by four circles in Fig. 4. The destination of each vehicle is chosen randomly from these four destinations. Finally, some areas in the position space are blocked out by black contours in Fig. 4.

For this simulation, we use the following dynamics for each vehicle:

$$\begin{align*}
\dot{p}_{x,i} &= v_i \cos \theta_i + d_{x,i} \\
\dot{p}_{y,i} &= v_i \sin \theta_i + d_{y,i} \\
\dot{\theta}_i &= \omega_i, \\
v &\leq v_i \leq \bar{v}, \; |\omega_i| \leq \bar{\omega}, \\
\| (d_{x,i}, d_{y,i}) \|_2 &\leq d_r
\end{align*}$$

where $x_i = (p_{x,i}, p_{y,i}, \theta_i)$ is the state of vehicle $Q_i$, $p_i = (p_{x,i}, p_{y,i})$ is the position, $\theta_i$ is the heading, and $d = (d_{x,i}, d_{y,i})$ represents $Q_i$'s disturbances, for example wind, that affect its position evolution. The control of $Q_i$ is $u_i = (v_i, \omega_i)$, where $v_i$ is the speed of $Q_i$ and $\omega_i$ is the turn rate; both controls have a lower and upper bound. To make our simulations as close as possible to real scenarios, we choose velocity and turn-rate bounds as $v = 0$ m/s, $\bar{v} = 25$ m/s, $\bar{\omega} = 2$ rad/s, aligned with the modern UAV specifications [64], [65]. For planning, we choose the reduced control authority to be $U^0_r = \{ (v_{r,j}, \omega_{r,j}) : 11 \text{ m/s} \leq v_{r,j} \leq 13 \text{ m/s}, |\omega_{r,j}| \leq 1.2 \text{ rad/s} \}$. Given this reduced control authority, we obtain a tracking error bound as well as a disturbance rejection controller (see Section II-B).

The disturbance bounds are chosen to be either $d_r = 6$ m/s or $d_r = 11$ m/s. These conditions correspond to moderate breeze and strong breeze respectively on the Beaufort scale [66]. In our simulations, we draw disturbance values uniformly randomly given these bounds.

The scheduled times of arrival for all vehicles are chosen to be same for all vehicles (0 without loss of generality) for a high UAV density condition. For medium and low density conditions, we separated the arrival times by 5 seconds and 10 seconds respectively, with the latest $t^{\text{STA}}_j$ being 0. Note that we have used same dynamics and input bounds across all vehicles for clarity of illustration; however, STP can easily handle more general systems of the form in which the vehicles have different control bounds, $t^{\text{STA}}_j$ and dynamics.

The goal of the vehicles is to reach their destinations while avoiding a collision with the other vehicles or the static obstacles. The joint state space of this 50-vehicle system is 150-dimensional (150D), making the joint trajectory planning and collision avoidance problem intractable for direct analysis. Therefore, we assign a priority ordering to vehicles and solve the trajectory planning problem sequentially. For this simulation, we assign a random priority order to fifty vehicles.

Fig. 5: Snapshots of vehicle trajectories at approximately a) 1 minute, b) 3 minutes, c) 4 minutes, and d) 5 minutes after the first vehicle departs. The wind speed is uniformly random with a bound of $d_r = 6$ m/s. The vehicles remain clear of all static obstacles and of each other despite the disturbance in the dynamics.

The full trajectories of vehicles from their departure to arrival are shown in Fig. 6a. All vehicles reach their respective destinations. A zoomed-in version of Fig. 6a near the red target...
lanes, and more and more trajectories become state-separated as this relative separation increases. The disturbance magnitude determines the relative separation between different lanes, and more and more trajectories become state-separated as the disturbance increases.

(Fig. 7) illustrates that vehicles are also outside each other’s danger zones (circles around the vehicles) as required.

It is interesting to note that the vehicles going to the same destination take different trajectories. This is because all vehicles have the same scheduled time of arrival, and hence the lower-priority vehicles do not have the flexibility to wait for the higher-priority vehicles. In order to ensure that they reach their destinations on time, they must depart earlier and take alternative trajectories to their destinations, forming different “traffic lanes”. Thus, the vehicles’ trajectories are state-separated, i.e., they follow different state trajectories but at the same time.

The average trajectory computation time per vehicle is 2 seconds using the CUDA implementation of the level set method in BEACLS. Computations were run on a desktop computer with a Core i7 5820K processor and two GeForce GTX Titan X graphics processing units. Recall that all of the computation is done offline and the resulting BRS and corresponding control policy are obtained as lookup tables. In real time, computation and communication between vehicles is not required. Only a lookup table query is required, and this can be performed very quickly in real time. This illustrates the capability of STP as a provably safe trajectory planning algorithm for large multi-vehicle systems.

Without the CUDA implementation in BEACLS, the approximate computation time per vehicle is 33 seconds using the C++ implementation of the level set toolbox without CUDA, and 200 seconds using the level set toolbox and helperOC in MATLAB. A comparison of the total computation time for all 50 vehicles is shown in Figure 5.

C. Effects of Disturbance and Scheduled Time of Arrival

In this section, we illustrate how the disturbance bound $d_r$ in (23) and the relative $t_{STA}^{ij}$’s of vehicles affect the vehicle trajectories. For this purpose, we simulate the STP algorithm in five scenarios:

- Case 0: $d_r = 6$ m/s, $t_{STA}^{ij} = 0 \forall i$ (moderate breeze, high UAV density; the simulation also shown in Figure 5).
- Case 1: $d_r = 11$ m/s, $t_{STA}^{ij} = 0 \forall i$ (strong breeze, high UAV density).
- Case 2: $d_r = 6$ m/s, $t_{STA}^{ij} = 5(i-1)$ (moderate breeze, medium UAV density).
- Case 3: $d_r = 11$ m/s, $t_{STA}^{ij} = 5(i-1) \forall i$ (strong breeze, medium UAV density).
- Case 4: $d_r = 11$ m/s, $t_{STA}^{ij} = 10(i-1) \forall i$ (strong breeze, low UAV density).

The interpretation of $t_{STA}^{ij} = 5(i-1)$ is that the scheduled time of arrival of any two consecutive vehicles is separated by 5 seconds, which represents a medium vehicle density scenario; a separation of 10 seconds represents a low vehicle density scenario. $d_r = 6$ m/s and $d_r = 11$ m/s correspond to the moderate breeze and strong breeze respectively on Beaufort wind force scale.

Intuitively, as $d_r$ increases, it is harder for a vehicle to closely track a particular nominal trajectory, which results in a higher tracking error bound. As mentioned previously, with a 6 m/s wind speed, the tracking error bound is 5 meters; however, with an 11 m/s wind speed, the tracking error bound becomes 35 meters. Thus, the vehicles need to be separated more from each other in space, compared to with a 6 m/s wind speed, to ensure that they do not enter each other’s danger zones. This is
also evident from comparing the results corresponding to Case 0 (Fig. 6a) and Case 1 (Fig. 6b). As the disturbance magnitude increases from \(d_r = 6\) m/s (moderate breeze) to \(d_r = 11\) m/s (strong breeze), the vehicles’ trajectories get farther apart from each other. Since \(t^{\text{STA}}\) is same for all vehicles, the vehicles trajectories are still predominately state-separated.

We next compare Case 0 (Fig. 6a) and Case 2 (Fig. 6c). The difference between these two cases is that vehicles have a 5-second separation in their schedule times of arrival in Case 2. When vehicles \(Q_i\) and \(Q_j\) \((j > i)\) have same scheduled time of arrival as in Case 0, and are going to the same destination, they are constrained to travel at the same time to make sure they reach the destination by the designated \(t^{\text{STA}}\). However, since \(Q_i\) is high-priority, it is able to take an optimal trajectory (in terms of the total time of travel to destination) and \(Q_j\) has to settle for a relatively sub-optimal trajectory. Thus, all vehicles going to a particular destination take different trajectories, creating a “band” of trajectories between the origin and the destination, as shown in Fig. 6a. The high-priority vehicles take a relatively straight trajectory between the origin and the destination whereas the low-priority vehicles take a (relatively sub-optimal) curved trajectory. If we think of an air highway between the origin and the destination, then vehicles take different lanes of that highway to reach the destination in Case 0. Thus, the trajectories of vehicles in this case are state-separated. However, when \(t^{\text{STA}}_i > t^{\text{STA}}_j\), then \(Q_j\) is not bound to travel at the same time as \(Q_i\); it can wait for \(Q_i\) to depart and take a shorter trajectory later on. Thus, vehicles travel in a single lane in this case, as shown in Fig. 6c. In other words, they take the same trajectory to the destination, but at different times. Thus, the trajectories of vehicles in this case are time-separated.

Note that the exact number of lanes depends on both the disturbance (wind speed) and separation of scheduled times of arrival (vehicle density). As the disturbance increases, vehicles need to be separated more from each other to ensure safety. A larger arrival time difference between vehicles is also able to ensure this separation even if the vehicles were to take the same lane. As shown in Fig. 6d, a difference of 5 seconds in the \(t^{\text{STA}}\)’s is not sufficient to achieve a single lane behavior for stronger 11 m/s wind conditions. However, the number of lanes is significantly fewer than that in Case 1 (Fig. 6b).

Finally, a separation of 10 seconds in \(t^{\text{STA}}\)’s ensure that we get the single lane behavior even in the presence of 11 m/s winds, leading to time-separated trajectories, as shown in Fig. 8. Videos of the simulations can be found on YouTube.3

Given our observations about the simulations presented, one can conclude, more generally, that the relative magnitude of disturbance and scheduled times of arrival separation determines the number of lanes and type of trajectories that emerge out of the STP algorithm. For a fixed disturbance magnitude, as the separation in the scheduled times of arrival of vehicles increases, the number of lanes between a pair of origin and destination decreases, and more and more trajectories become time-separated. On the other hand, for a fixed separation in the scheduled times of arrival of vehicles, as the disturbance magnitude increases, the number of lanes between a pair of origin and destination increases, and more and more trajectories become state-separated.

---

3 Video link: https://youtu.be/1ocaBGZqSAE
Fig. 9: Multi-city simulation setup. A 300 km² area of San Francisco Bay Area is used as the state-space for vehicles. STP vehicles fly to and from the four cities indicated by the four disks. The simulations are performed under the strong winds condition with $d_r = 11$ m/s.

B. Results

The resulting trajectories of vehicles are shown in Fig. 10a. Once again, the vehicles remain clear of all other vehicles and reach their respective destinations. Since the vehicles' scheduled times of arrival are separated by 5 seconds, the trajectories are predominately time-separated, with roughly two lanes for each pair of cities (one for going from city A to city B and another for from city B to city A). A high density of vehicles is achieved in the center since the 4 trajectories are intersecting in the center (Richmond-Oakland, Oakland-Richmond, Berkeley-San Francisco, San Francisco-Berkeley), but the STP algorithm ensures safety despite this high-density, as shown in the zoomed-in version of the center at an intermediate time when a large number of vehicles are passing through the central region (Fig. 10b).

We also simulated the system for the case in which $t_{STA}^i = 0 \forall i$. As is evident from Fig. 11 we get multiple lanes between each pair of cities in this case and trajectories become predominately state-separated, as we expect based on the discussion in Section V.C.

The average computation time per vehicle is 4 minutes using BEACLS on a desktop computer with a Core i7 5820K processor and two GeForce GTX Titan X graphics processing units. The computation time is much longer than in the previous simulation in SF because of the larger space over which planning is done. Once again all the computation is done offline and only a look-up table query is required in real-time, which can be performed very efficiently. Extrapolating the computation time comparison in Fig. 3 the MATLAB implementation would take prohibitively long – approximately 90 days for the entire simulation. This simulation illustrates the scalability and the potential of deploying the STP algorithm.

Fig. 10: (a) Trajectories obtained from the STP algorithm for the multi-city simulation with $d_r = 11$ m/s, $t_{STA}^i = 5(i - 1)$. (b) Zoomed-in version of the central area. A high density of vehicles is achieved at the center because of the intersection of several trajectories; however, the STP algorithm still ensures that vehicles do not enter each other’s danger zones and reach their destinations.

Fig. 11: Vehicle trajectories for $d_r = 11$ m/s, $t_{STA}^i = 0$. Since different vehicles have same scheduled times of arrival, a multiple-lane behavior is observed between every pair of cities.
with BEACLS for provably safe trajectory planning for large multi-vehicle systems.

VII. CITY ENVIRONMENT SIMULATIONS IN THE PRESENCE OF AN INTRUDER

A. Setup

In addition to the setup in Section V-A, the vehicles now also need to account for the possibility of the presence of an intruder for a maximum duration of $t_{IAT} = 10$ s. The intruder dynamics are given by (23).

B. Results

We present the simulation results for $k = 3$. The resultant buffer region is shown in Blue in Figure 12. For the comparison purposes, we also compute the buffer regions for $k = 2$ and $k = 4$. As shown in Figure 12, a bigger buffer is required between vehicles when $k$ is smaller. Intuitively, when $k$ is smaller, a larger buffer is required to ensure that the intruder spends more time “traveling” through this buffer region so that it can affect fewer vehicles within the same duration of $t_{IAT} = 10$ s.

These buffer region computations along with the induced obstacle computations are performed sequentially for each vehicle to obtain $\mathcal{G}(\cdot)$ in (19). This overall obstacle set is then used during their trajectory planning and the control policy $u_{PP}(\cdot)$ is computed, as defined in (19). Finally, the corresponding nominal trajectories are obtained by executing control policy $u_{PP}(\cdot)$. The nominal trajectories and the overall obstacles for different vehicles are shown in Figure 13. The numbers in the figure represent the vehicle numbers. The nominal trajectories (solid lines) are well separated from each other to ensure collision avoidance even during a worst-case intruder “attack”. At any given time, the vehicle density is low to ensure that the intruder cannot force more than three vehicles to apply an avoidance maneuver. This is also evident from large obstacles induced by vehicles for the lower priority vehicles (dashed circles). This lower density of vehicles is the price that we pay for ensuring that the replanning can be done efficiently in real-time.

In Figure 14 we plot the distance between an STP vehicle and the intruder when the vehicle applies the control policies $u_{PP}(\cdot)$ (Red line) and $u_{A}$ (Blue line) in the presence of the intruder. Black dashed line represents the collision radius $r = 100$ m between the vehicle and the intruder. If the vehicle continues to apply the control policy $u_{PP}(\cdot)$ in the presence of an intruder, the intruder enters in its danger zone. Thus, it is forced to apply the avoidance control, which can cause a deviation from the nominal trajectory, but will successfully avoid the intruder, as indicated by the Blue curve.

The relative buffer region between vehicles is computed under the assumption that both the STP vehicle and the intruder are trying to collide with each other; this is to ensure that the intruder will need at least a duration of $t_{BRD}$ to reach the boundary of the avoid region of the next vehicle, irrespective of the control applied by the vehicle. However, a vehicle will be applying the control policy $u_{PP}(\cdot)$ unless the intruder forces it to apply an avoidance maneuver, which may not necessarily correspond to the policy that the vehicle will use to deliberately collide with the intruder. Therefore, it may not be able to affect $k$ vehicles even with its best strategy to affect maximum vehicles. In this simulation, the intruder is able to force only two vehicles to apply an avoidance maneuver. The set of vehicles that will need to replan their trajectories is given by $N_{RD} = \{ Q_{1}, Q_{2} \}$. This conservatism in our method is discussed further in Section VII-C.

The time for planning and replanning for each vehicle is approximately 15 minutes on a MATLAB implementation on...
a desktop computer with a Core i7 5820K processor. With BEACLS using two GeForce GTX Titan X graphics processing units, this computation time is reduced to approximately 9 seconds per vehicle. So for \( k = 3 \), replanning would take less than 30 seconds. Since reachability computations in BEACLS are highly parallelizable, replanning should be possible to do within a fraction of seconds with more computational resources.

C. Discussion

The simulations illustrate the effectiveness of reachability in ensuring that the STP vehicles safely reach their respective destinations even in the presence of an intruder. However, they also highlight some of the conservatism in the worst-case reachability analysis. For example, in the proposed algorithm, we assume the worst-case disturbances and intruder behavior while computing the buffer region and the induced obstacles, which results in a large separation between vehicles and hence a lower vehicle density overall, as evident from Figure 13. Similarly, while computing the buffer region, we assumed that a vehicle is deliberately trying to collide with the intruder so we once again consider the worst-case scenario, even though the vehicle will only be applying the nominal control strategy \( u^P(\cdot) \), which is usually not be same as the worst-case control strategy. This worst-case analysis is essential to guarantee safety regardless of the actions of STP vehicles, the intruder, and disturbances, given no other information about the intruder’s intentions and no model of disturbances except for the bounds. However, the conservatism of our results illustrates the need and the utility of acquiring more information about the intruder and disturbances, and of incorporating knowledge of the nominal strategy \( u^P(\cdot) \) in future work.

VIII. Conclusion and Future Work

Provably safe multi-vehicle trajectory planning in an important problem that needs to be addressed to ensure that vehicles can fly in close proximity of each other. Recently, the STP algorithm was proposed for multi-vehicle trajectory planning problem that scales linearly with the number of vehicles. We propose BEACLS which can leverage the computation power of GPUs along with the linear scaling of the STP framework for efficient and provably safe large-scale multi-vehicle trajectory planning. We demonstrate how different types of space-time trajectories emerge naturally out of the STP algorithm for different disturbance conditions and other problem parameters.

We also propose an algorithm to account for an adversarial intruder in sequential trajectory planning. The proposed method ensures that only a fixed number of vehicles need to replan their trajectories once the intruder disappears, irrespective of the total number of vehicles. Thus, the replanning process is feasible in real-time. Future work includes exploring methods that can account for multiple simultaneous intruders, reducing conservatism in the current reachability analysis, and deploying the proposed framework on a multi-vehicle hardware testbed.

REFERENCES


