Provably Safe and Scalable Multi-Vehicle Trajectory Planning

Somil Bansal*[1], Mo Chen*[2], Ken Tanabe[3], Claire J. Tomlin[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]–[30]. Methods which do combine both planning and collision avoidance typically make simplifying assumptions about the vehicle dynamics [31]–[33].

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 moving 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

This research is supported by NSF under CPS:ActionWebs (CNS-931843), under the CPS Frontiers VehiCal project (1545126), by the UC-Philippine-California Advanced Research Institute under project IIID-2016-005, and by the ONR MURI Embedded Humans (N00014-16-1-2206).

* Both authors contributed equally to this work.

[1] The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA (e-mail: {somil, tomlin}@berkeley.edu).

[2] The author is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: mochen@cs.sfu.ca).

[3] The author is with the Center for Semiconductor Research & Development, Toshiba Corporation, Minato-ku, Tokyo 105-8001, Japan (e-mail: nekebanat@gmail.com).

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) $\mathcal{V}$, a forward reach-avoid set (FRS) $\mathcal{W}$, or a sequence of BRSs and FRSs, given some target set $\mathcal{L}$, time-varying obstacle $\mathcal{G}(t)$ 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 $\mathcal{L}$ while avoiding the obstacles $\mathcal{G}(\cdot)$, and the FRS represents the sets of states that can be reached starting from the target set $\mathcal{L}$. For details on HJ reachability and its application in STP, we encourage the reader to refer to [1], [59].

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

$$\begin{aligned} \max\Big\{ \min\{D_t V(t,x) + H(t,x,\nabla V(t,x)), l(x) - V(t,x)\}, \\ - g(t,x) - V(t,x)\Big\} = 0, \quad t \le t_f \\ V(t_f, x) = \max\{l(x), -g(t_f, x)\} \end{aligned} \tag{1}$$

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

$$\begin{aligned} D_t W(t,x) + H(t,x,\nabla W(t,x)) = 0, \quad t \ge t_0 \\ W(t_0, x) = l(x) \end{aligned} \tag{2}$$

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

$$\begin{aligned} \mathcal{V}(t, t_f) = \{x : V(t, x) \le 0\} \\ \mathcal{W}(t_0, t) = \{x : W(t, x) \le 0\} \end{aligned} \tag{3}$$

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

$$\dot{x}_i = f_i(x_i, u_i, d_i), t \leq t_i^{\text{STA}} \tag{4}$$
$$u_i \in \mathcal{U}_i, d_i \in \mathcal{D}_i, i = 1 \ldots, N \tag{5}$$

where $x_i \in \mathbb{R}^{n_i}$, $u_i \in \mathcal{U}_i$ and $d_i \in \mathcal{D}_i$, 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 $\mathcal{U}_i, \mathcal{D}_i$ 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 $\mathcal{L}_i$ by some *scheduled time of arrival* t_i^{STA}. The target in general represents some set of desirable states, for example the destination of Q_i. On its way to $\mathcal{L}_i$, Q_i must avoid a set of static obstacles $\mathcal{O}_i^{\text{static}} \subset \mathbb{R}^{n_i}$, 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

$$\mathcal{Z}_{ij} = \{(x_i, x_j) : \|p_i - p_j\|_2 \leq R_c\}. \tag{6}$$

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

Given the set of STP vehicles, their targets $\mathcal{L}_i$, the static obstacles $\mathcal{O}_i^{\text{static}}$, and the vehicles' danger zones with respect to each other $\mathcal{Z}_{ij}$, the goal of STP is as follows. For each vehicle Q_i, synthesize a controller which guarantees that Q_i reaches its target $\mathcal{L}_i$ at or before the *scheduled time of arrival* t_i^{STA}, while avoiding the static obstacles $\mathcal{O}_i^{\text{static}}$, and the danger zones with respect to all other vehicles $\mathcal{Z}_{ij}, j \neq i$. In addition, we would like to obtain the *latest departure time* t_i^{LDT} such that Q_i can still arrive at $\mathcal{L}_i$ 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 $\mathcal{O}_i^k(t)$, 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 $\mathcal{L}_i$ while avoiding the static obstacles $\mathcal{O}_i^{\text{static}}$, and the danger zones of *higher-priority vehicles* $\mathcal{O}_i^k(\cdot), k < i$, 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* $\mathcal{U}_i^p \subset \mathcal{U}_i$ 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:

$$\dot{x}_{r,i} = f_i(x_{r,i}, u_{r,i}), t \leq t_i^{\text{STA}} \tag{7}$$
$$u_{r,i} \in \tilde{\mathcal{U}}_i, i = 1 \ldots, N,$$

where the particular reference trajectory $x_{r,i}(t)$ is obtained by choosing a reference control function $u_{r,i} \in \tilde{\mathcal{U}}_i$. 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 $\mathcal{L}_i$ by the same amount to obtain $\tilde{\mathcal{L}}_i$, 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_I 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^{IAT}.

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^{IAT}.

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_I^0, 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, $\bar{k}$, 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 considerations for real-world systems, since the replanning neeeds to be done during the runtime. Moreover, $\bar{k}$ 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 $\bar{k}$ vehicles within the duration of t^{IAT}. This division of the flight space guarantees that the intruder can affect the trajectory of at most $\bar{k}$ vehicles despite its best efforts, resulting in at most $\bar{k}$ 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 $t^{\text{BRD}} = \frac{t^{\text{IAT}}}{\bar{k}}$ to travel through this region. Thus, within the duration of t^{IAT}, the intruder can force at most $\bar{k}$ 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, $\mathcal{S}_i(t)$, denotes the set of states of the intruder for which the vehicle Q_i is forced to apply an avoidance maneuver. $\mathcal{S}_i(t)$ is given by the set of states from which the joint states of Q_I and Q_i can enter the danger zone $\mathcal{Z}_{iI}$ despite the best efforts of Q_i to avoid Q_I. We define the relative dynamics between Q_I and Q_i:

$$x_{Ii} = x_I - x_i, \qquad \dot{x}_{Ii} = f_r(x_{Ii}, u_i, u_I, d_i, d_I) \tag{8}$$

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

$$\begin{aligned}\mathcal{V}_i^{\text{A}}(\tau, t^{\text{IAT}}) =& \{y : \forall u_i(\cdot) \in \mathbb{U}_i, \exists u_I(\cdot) \in \mathbb{U}_I, \exists d_i(\cdot) \in \mathbb{D}_i,\\ & \exists d_I(\cdot) \in \mathbb{D}_I, x_{Ii}(\cdot) \text{ satisfies (8)},\\ & \exists s \in [\tau, t^{\text{IAT}}], x_{Ii}(s) \in \mathcal{L}_i^{\text{A}}, x_{Ii}(\tau) = y\},\\ \mathcal{L}_i^{\text{A}} =& \{x_{Ii} : \|p_{Ii}\|_2 \le R_c\}.\end{aligned} \tag{9}$$

$\mathcal{V}_i^{\text{A}}$ can be computed using the HJI-VI in (1) with Hamiltonian

$$H_i^{\text{A}}(x_{Ii}, \lambda) = \max_{u_i \in \mathcal{U}_i} \min_{\substack{u_I \in \mathcal{U}_I,\\ d_I \in \mathcal{D}_I,\\ d_i \in \mathcal{D}_i}} \lambda \cdot f_r(x_{Ii}, u_i, u_I, d_i, d_I). \tag{10}$$

The interpretation of $\mathcal{V}_i^{\text{A}}(\tau, t^{\text{IAT}})$ is that if Q_i starts inside this set, i.e., $x_{Ii}(t) \in \mathcal{V}_i^{\text{A}}(\tau, t^{\text{IAT}})$, then the intruder can force Q_i to enter the danger zone $\mathcal{Z}_{iI}$ within a duration of $(t^{\text{IAT}} - \tau)$, regardless of the control applied by the vehicle. If Q_i starts at the boundary of this set (denoted as $\partial\mathcal{V}_i^{\text{A}}(\tau, t^{\text{IAT}})$), i.e., $x_{Ii}(t) \in \partial\mathcal{V}_i^{\text{A}}(\tau, t^{\text{IAT}})$, it can *barely* avoid the intruder for a duration of $(t^{\text{IAT}} - \tau)$ using the optimal avoidance control, u_i^{A}, that maximizes the Hamiltonian

$$u_i^{\text{A}}(x_{Ii}) = \arg\max_{u_i \in \mathcal{U}_i} \min_{\substack{u_I \in \mathcal{U}_I,\\ d_I \in \mathcal{D}_I,\\ d_i \in \mathcal{D}_i}} \lambda \cdot f_r(x_{Ii}, u_i, u_I, d_i, d_I). \tag{11}$$

Finally, if Q_i starts outside $\mathcal{V}_i^{\text{A}}$, then Q_i and Q_I cannot instantaneously enter the danger zone $\mathcal{Z}_{iI}$, irrespective of the control applied by them at time t. In fact, Q_i can safely apply *any* control as long as it is outside the boundary of this set, but will have to apply the avoidance control once it reaches the boundary. Given $\mathcal{V}_i^{\text{A}}$, the separation region is given by

$$\mathcal{S}_i(t) = \mathcal{M}_i(t) + \mathcal{V}_i^{\text{A}}(0, t^{\text{IAT}}), \tag{12}$$

where the "+" in (12) denotes the Minkowski sum. Here, $\mathcal{M}_i(t)$ represents all possible states of Q_i at time t.

We now 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. This ensures that it can deviate at most $\bar{k}$ STP vehicles from their trajectories within a duration of t^{IAT}. In relative state space, the buffer region is given by the BRS, $\mathcal{V}_i^{\text{B}}(0, t^{\text{BRD}})$, corresponding to the target set $\mathcal{V}_i^{\text{A}}(t^{\text{BRD}}, t^{\text{IAT}})$ with Hamiltonian

$$H_i^{\text{B}}(x_{Ii}, \lambda) = \min_{\substack{u_i \in \mathcal{U}_i, u_I \in \mathcal{U}_I,\\ d_i \in \mathcal{D}_i, d_I \in \mathcal{D}_I}} \lambda \cdot f_r(x_{Ii}, u_i, u_I, d_i, d_I). \tag{13}$$

Intuitively, $\mathcal{V}_i^{\text{B}}(0, t^{\text{BRD}})$ represents the set of all relative states x_{Ii} from which it is possible to reach the boundary of $\mathcal{V}_i^{\text{A}}(t^{\text{BRD}}, t^{\text{IAT}})$ within a duration of t^{BRD}. Thus, as long as the initial relative state is outside $\mathcal{V}_i^{\text{B}}(0, t^{\text{BRD}})$, Q_i does not need to deviate from its path to avoid the intruder for at least a duration of t^{BRD}. Given $\mathcal{V}_i^{\text{B}}(0, t^{\text{BRD}})$, the buffer region $\mathcal{B}_{ij}(t)$ at time t between vehicle Q_i and a higher-priority vehicle Q_j is given by

$$\mathcal{B}_{ij}(t) = \partial\mathcal{S}_j(t) + \left(-\mathcal{V}_i^{\text{B}}(0, t^{\text{BRD}})\right). \tag{14}$$

C. Trajectory Planning for Intruder Avoidance

In addition to maintaining the buffer between Q_i and a higher priority vehicle, Q_j, we need to make sure that any two STP vehicles accidentally do not come too close to each other while applying the avoidance maneuver. Consequently, it is sufficient to ensure that their relative state remains outside the BRS, $\mathcal{V}_{ij}^{\text{C}}(0, t^{\text{IAT}})$, representing the set of all relative states x_{ij} from which the vehicles Q_i and Q_j can enter the set $\mathcal{Z}_{ij}$. Thus, the lower priority vehicle should avoid the set

$$\mathcal{O}_i^j(t) = \mathcal{M}_j(t) + \mathcal{V}_{ij}^{\text{C}}(0, t^{\text{IAT}}). \tag{15}$$

Finally, we compute the set of states from which Q_i can collide with any static obstacle, $\mathcal{O}_i^{\text{static}}$. This set is given by the BRS $\mathcal{V}_i^{\text{S}}(t, t + t^{\text{IAT}})$, 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:

$$\mathcal{G}_i(t) = \mathcal{V}_i^{\text{S}}(t, t + t^{\text{IAT}}) \bigcup \cup_{j=1}^{i-1} \mathcal{O}_i^j(t) \bigcup \cup_{j=1}^{i-1} \mathcal{B}_{ij}(t). \tag{16}$$

Given $\mathcal{G}_i(t)$, we compute a BRS $\mathcal{V}_i^{\text{PP}}(t, t_i^{\text{STA}})$ for trajectory planning that contains the initial state of Q_i and avoids $\mathcal{G}_i(t)$:

$$\begin{aligned}\mathcal{V}_i^{\text{PP}}(t, t_i^{\text{STA}}) =& \{y : \exists u_i(\cdot) \in \mathbb{U}_i, \forall d_i(\cdot) \in \mathbb{D}_i, \\ & x_i(\cdot) \text{ satisfies (4)}, \\ & \forall s \in [t, t_i^{\text{STA}}], x_i(s) \notin \mathcal{G}_i(s), \\ & \exists s \in [t, t_i^{\text{STA}}], x_i(s) \in \mathcal{L}_i, x_i(t) = y\},\end{aligned} \tag{17}$$

with Hamiltonian

$$H_i^{\text{PP}}(x_i, \lambda) = \min_{u_i \in \mathcal{U}_i} \max_{d_i \in \mathcal{D}_i} \lambda \cdot f_i(x_i, u_i, d_i). \tag{18}$$

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

$$u_i^{\text{PP}}(t, x_i) = \arg \min_{u_i \in \mathcal{U}_i} \max_{d_i \in \mathcal{D}_i} \lambda \cdot f_i(x_i, u_i, d_i) \tag{19}$$

When intruder is not present in the system, Q_i applies the control u_i^{PP} and we get the *nominal trajectory* of Q_i. Once intruder appears in the system, Q_i applies the avoidance control u_i^{A} 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:

$$u_i^*(t, x_i, x_I) = \begin{cases} u_i^{\text{PP}}(t, x_i) & x_{Ii}(t) \notin \mathcal{V}_i^{\text{A}}(t, t^{\text{IAT}}) \\ u_i^{\text{A}}(t, x_{Ii}) & \text{otherwise} \end{cases} \tag{20}$$

If Q_i starts within $\mathcal{V}_i^{\text{PP}}$ and uses the control u_i^*, 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 $\bar{k}$ 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 $\mathcal{N}^{\text{RP}}$ denote the set of all vehicles for whom replanning is required. $\mathcal{N}^{\text{RP}}$ can be obtained by checking if a vehicle Q_i applied any avoidance control during $[\underline{t}, \underline{t} + t^{\text{IAT}}]$, i.e.,

$$\mathcal{N}^{\text{RP}} = \{Q_i : \exists t \in [\underline{t}, \underline{t} + t^{\text{IAT}}], x_{Ii}(t) \in \mathcal{V}_i^{\text{A}}(t, t^{\text{IAT}})\}, \tag{21}$$

where $\underline{t}$ 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 $\bar{k}$ vehicles can be affected by Q_I, i.e., $|\mathcal{N}^{\text{RP}}| \leq \bar{k}$. The replanning for the vehicles in $\mathcal{N}^{\text{RP}}$ can be performed by assigning them the lowest priority and using the nominal STP algorithm presented in Section II-B. Note that $\bar{k}$ 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 specifies of STP vehicle trajectories may be incorporated; however, we defer such considerations to future work.

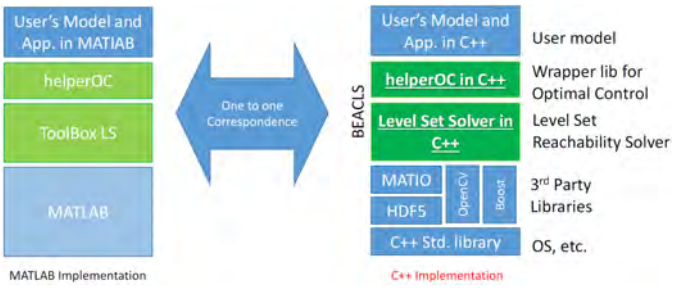


Fig. 1: Correspondence between the MATLAB implementation of Level Set Toolbox and BEACLS. In the case of this paper, the block called "User's Model and App." would be an implementation of an STP algorithm.

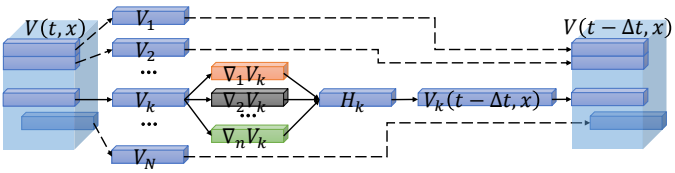


Fig. 2: BEACLS splits multi-dimensional arrays representing the value function $V(t, x)$ into appropriate overlapping chunks according to processor configuration. Numerical gradients and the Hamiltonian values of each chunk are computed in parallel to produce the updated value function at the previous 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.

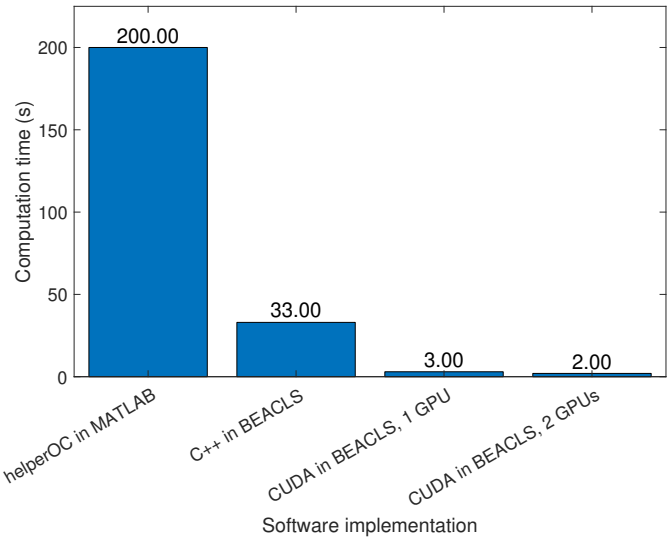


Fig. 3: Summary of computation times for the city simulation environment in Section V. A 50-UAV simulation takes less than 2 minutes with the CUDA implementation in BEACLS using two GPUs, compared to 27 minutes with a non-CUDA C++ implementation in BEACLS or 2.8 hours with helperOC and the level set toolbox in MATLAB.

- 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 [62], 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 V. 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 VI 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[1].

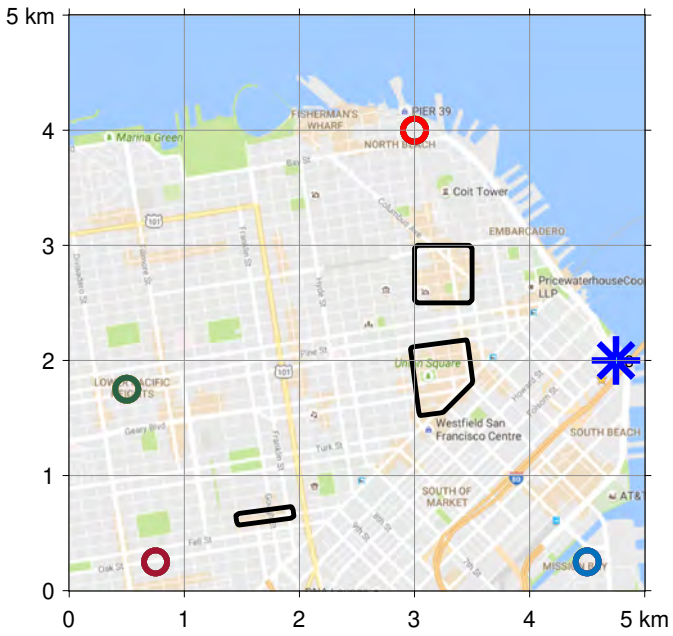


Fig. 4: City environment multiple UAV simulation setup. A 25 km^2 area in the City of San Francisco is used as the space for the 50-vehicle simulation. Vehicles originate from the blue star and go to one of the four destinations, denoted by circles. Tall buildings in the downtown area are used as static obstacles, represented by the black contours.

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 a 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[2].

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 [63]. In general there may be

[1] BEACLS website: https://github.com/HJReachability/beacls
[2] Video link: https://youtu.be/1ocaBGZqSAE

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 $\mathcal{L}_i$ 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

$$\mathcal{L}_i = \{x_i : \|p_i - c_i\|_2 \leq r\} \tag{22}$$

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 downtown SF and the city hall are used as representative static obstacles for the STP vehicles, denoted by black contours in Fig. 4.

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

$$\begin{aligned} \dot{p}_{x,i} &= v_i \cos\theta_i + d_{x,i} \\ \dot{p}_{y,i} &= v_i \sin\theta_i + d_{y,i} \\ \dot{\theta}_i &= \omega_i, \\ \underline{v} \leq v_i \leq \bar{v},\ &|\omega_i| \leq \bar{\omega}, \\ \|(d_{x,i}, d_{y,i})\|_2 &\leq d_r \end{aligned} \tag{23}$$

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, θ_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 ω_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 $\underline{v} = 0$ 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 $\mathcal{U}_j^p = \{(v_{r,j}, \omega_{r,j}) : 11 \text{ m/s} \leq v_{r,j} \leq 13 \text{ m/s}, |\omega_{r,j}| \leq 1.2 \text{ rad/s}\}$. 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_i^{STA} 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_i^{STA} 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.

B. High UAV Density with 6 m/s Wind

We start with Q_1 and sequentially compute the optimal control policy and the latest departure time t_j^{LDT} for each vehicle. In presence of moderate winds, the obtained tracking error bound is 5 meters. This means that given any trajectory (which is a sequence of states over time) of vehicle, winds can at most cause a deviation of 5 meters from this trajectory at all times. Consequently, the vehicle will be within a distance of 5 meters from the planned trajectory. Note that since all vehicles have same dynamics, the error bound is also same for all vehicles. This error bound is used to obtain the augmented obstacles and the reduced target set $\tilde{\mathcal{L}}_i$. Note that since disturbance directly impacts the computation of tracking error bound, in general the size of augmented obstacles increases as disturbance magnitude increases. We will illustrate the effect of disturbance magnitude on the trajectories of vehicles in Section V-C.

The nominal trajectory can now be obtained using a reduced control authority starting from the initial state x_j^0. The resulting trajectories of vehicles for $d_r = 6$ m/s and $t_j^{\text{STA}} = 0\ \forall j$ at different times are shown in Fig. 5. As is evident from the figures, the vehicles remain clear of all the static obstacles (the black contours) and make progress towards reaching their destinations, according to their planned trajectories. The vehicles whose destinations are relatively close need less time to travel to their destinations and thus they depart later.

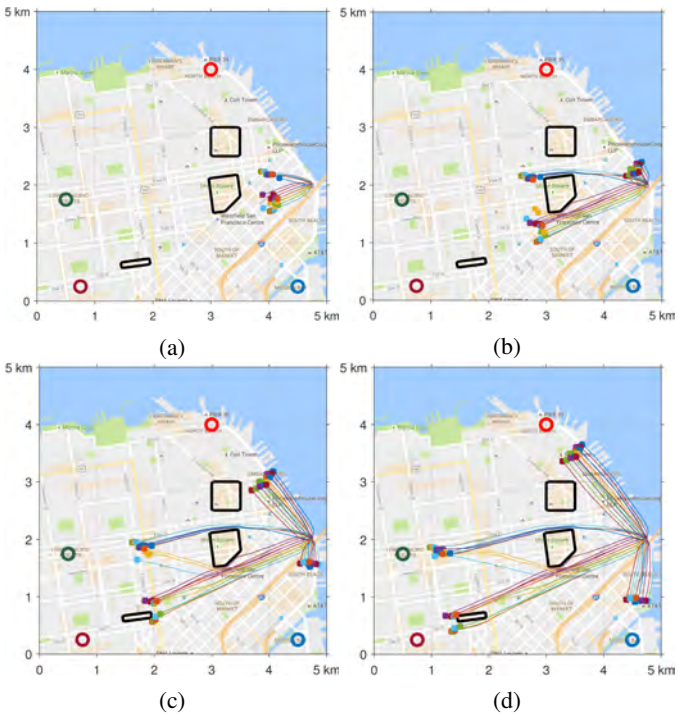
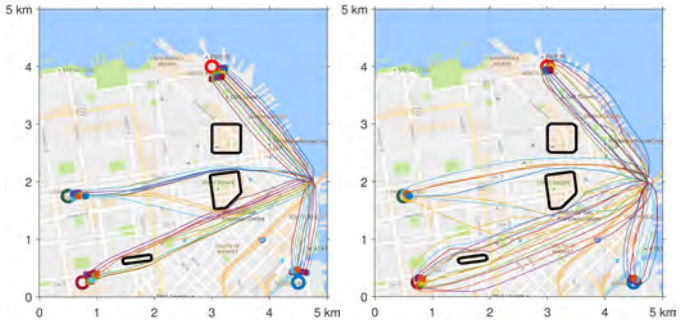
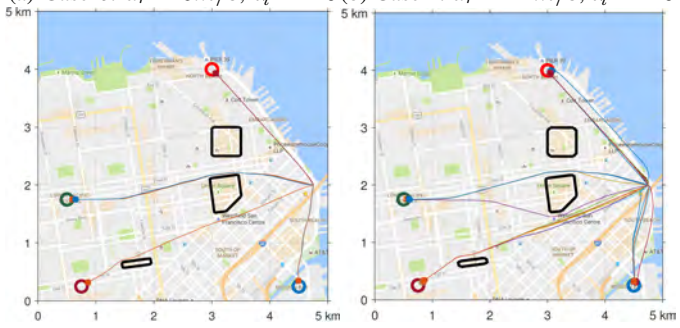


Fig. 5: Snapshots of vehicle trajectories at approximately a) 1 minute, b) 3 minutes, c) 4 minutes, and d) 5 minutes after the first vehicle departs. The wind speed is uniformly random with a bound of $d_r = 6$ m/s. The vehicles remain clear of all static obstacles and of each other despite the disturbance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



(a) Case 0: $d_r = 6m/s$, $t_i^{\text{STA}} = 0$ (b) Case 1: $d_r = 11m/s$, $t_i^{\text{STA}} = 0$



(c) Case 2: $d_r = 6m/s$, $t_i^{\text{STA}} = 5(i-1)$ (d) Case 3: $d_r = 11m/s$, $t_i^{\text{STA}} = 5(i-1)$

Fig. 6: Effect of the disturbance magnitude and the scheduled times of arrival on vehicle trajectories. All trajectories are simulated under uniformly random disturbance. The relative separation in the scheduled times of arrival of vehicles determines the number of lanes between a pair of origin and destination, and more and more trajectories become *time-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* trajectories, 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

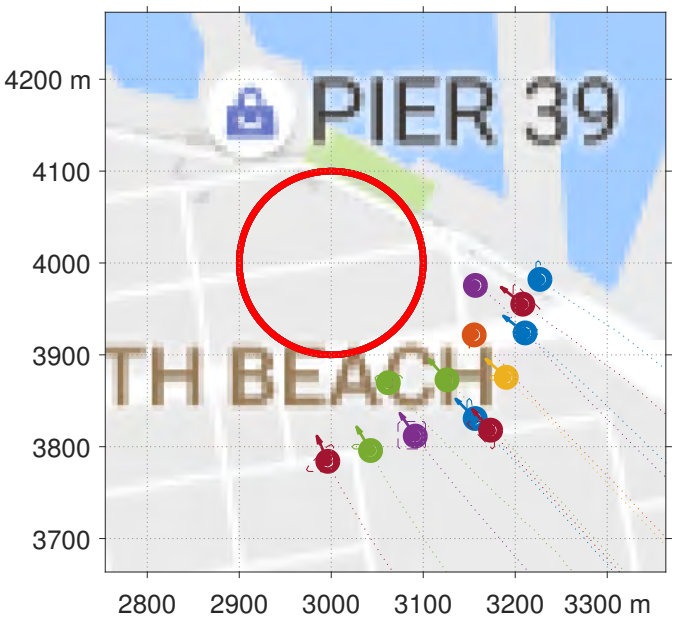


Fig. 7: Zoomed-in version of vehicle trajectories near the red target in Fig. 6a. The STP algorithm ensures that the vehicles are outside each other's danger zones; here the smallest distance between vehicles is just over 100 meters (blue and red vehicles below the letter "H").

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 3.

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

The interpretation of $t_i^{\text{STA}} = 5(i-1)$ 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 [66].

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^{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^{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 $t_j^{\text{STA}} > t_i^{\text{STA}}$, 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^{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^{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. 8: Trajectories of 50 vehicles for Case 4: $d_r = 11$ m/s, $t_i^{\text{STA}} = 10(i-1)$. Since different vehicles have different scheduled times of arrival, there is a single lane between every origin-destination pair.

VI. Large-Scale Multiple UAV Simulations: Multi-City Environment

We next use STP to design trajectories for a 200-UAV system where UAVs are flying through a multi-city region.

A. Setup

We use a part of the San Francisco Bay Area in California, USA as our position space, as shown in Fig. 9. We consider the UAVs flying to and from four cities: Richmond, Berkeley, Oakland, and San Francisco. The blue region in Fig. 9 represents bay (water). This environment is different from the city environment in Section V in that here the UAVs need to fly for longer distances and through a high-density vehicle environment with strong winds, but have no static obstacles like tall buildings. Due to the much larger number of vehicles and longer trajectory time horizons, many more reachable sets need to be computed, and, even more crucially, each computation must be done on a much larger computational domain. Therefore, the use of GPU parallelization is essential for making this simulation possible.

The vehicles are flying to and from the four cities indicated by the four circles. The origin and destination of each vehicle is chosen randomly from these four cities. The vehicle dynamics are given by (23). We choose velocity and turn-rate bounds as $\underline{v} = 0$ m/s, $\bar{v} = 25$ m/s, $\bar{\omega} = 2$ rad/s. The disturbance bound is chosen as $d_r = 11$ m/s, which corresponds to "strong breeze" on Beaufort wind force scale [66]. The scheduled time of arrival t^{STA} for vehicles are chosen as $5(i-1)$ seconds.

The goal of the vehicles is to reach their destinations while avoiding a collision with the other vehicles. The joint state space of this 200-vehicle system is 600-dimensional, making the joint trajectory planning and collision avoidance problem intractable for direct analysis. Therefore, we again use STP and assign a priority order to vehicles to solve the trajectory planning problem sequentially.

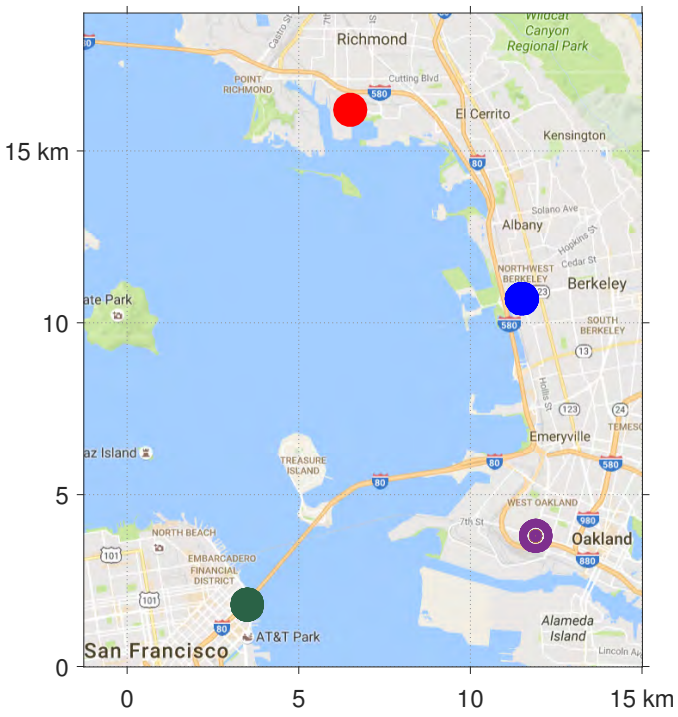


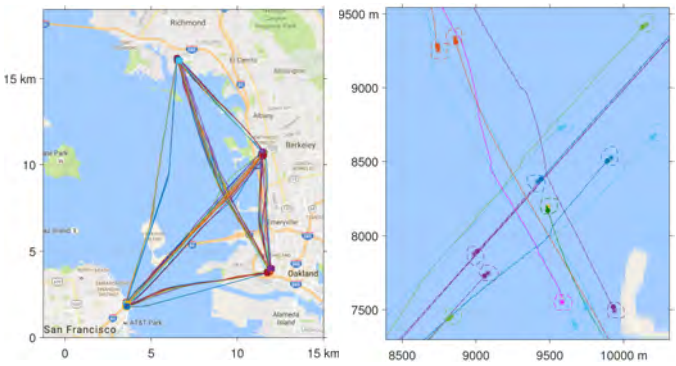
Fig. 9: Multi-city simulation setup. A 300 km^2 area of San Francisco Bay Area is used as the state-space for vehicles. STP vehicles fly to and from the four cities indicated by the four disks. The simulations are performed under the strong winds condition with $d_r = 11$ m/s.

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 $t_i^{\text{STA}} = 0 \ \forall i$. 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



(a) (b)

Fig. 10: (a) Trajectories obtained from the STP algorithm for the multi-city simulation with $d_r = 11$ m/s, $t_i^{\text{STA}} = 5(i-1)$. (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.

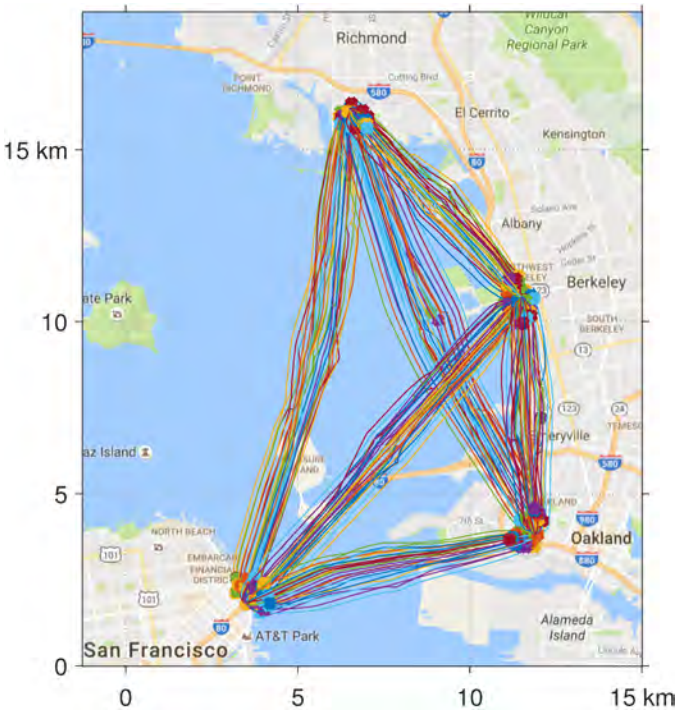


Fig. 11: Vehicle trajectories for $d_r = 11$ m/s, $t_i^{\text{STA}} = 0$. Since different vehicles have same scheduled times of arrival, a multiple-lane behavior is observed between every pair of cities.

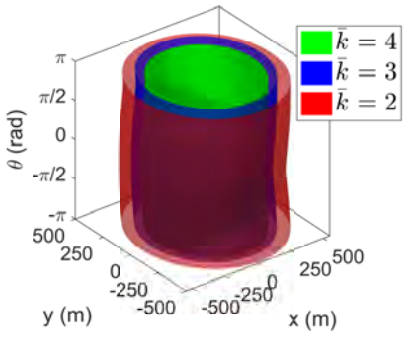


Fig. 12: Buffer regions for different $\bar{k}$ (best visualized with colors). As $\bar{k}$ decreases, a larger buffer is required between vehicles to ensure that the intruder spends more time traveling through this buffer region so that it forces fewer vehicles to apply an avoidance maneuver.

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 $t^{\text{IAT}} = 10\,\text{s}$. The intruder dynamics are given by (23).

B. Results

We present the simulation results for $\bar{k} = 3$. The resultant buffer region is shown in Blue in Figure 12. For the comparison purposes, we also compute the buffer regions for $\bar{k} = 2$ and $\bar{k} = 4$. As shown in Figure 12, a bigger buffer is required between vehicles when $\bar{k}$ is smaller. Intuitively, when $\bar{k}$ 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 $t^{\text{IAT}} = 10\,\text{s}$.

These buffer region computations along with the induced obstacle computations are performed sequentially for each vehicle to obtain $\mathcal{G}(\cdot)$ in (16). This overall obstacle set is then used during their trajectory planning and the control policy $u^{\text{PP}}(\cdot)$ is computed, as defined in (19). Finally, the corresponding nominal trajectories are obtained by executing control policy $u^{\text{PP}}(\cdot)$. 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 $u^{\text{PP}}(\cdot)$ (Red line) and u^{A} (Blue line) in the presence of the intruder. Black dashed line represents the collision radius $r = 100\,\text{m}$ between the vehicle and the intruder. If the vehicle

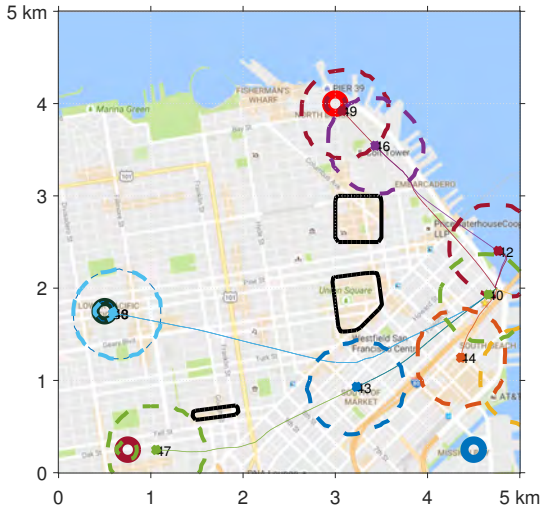


Fig. 13: Nominal trajectories and induced obstacles by different vehicles. The nominal trajectories (solid lines) are well separated from each other to ensure that the intruder cannot force more than 3 vehicles to apply an avoidance maneuver.

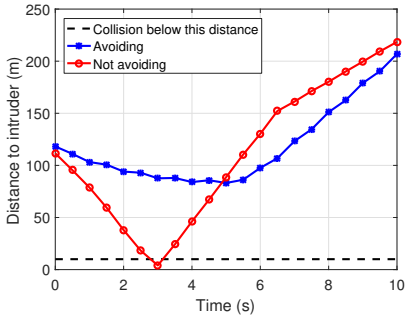


Fig. 14: The trajectory of a STP vehicle when it applies the nominal controller vs when it applies the avoidance control. The vehicle is forced to apply the avoidance maneuver in the presence of an intruder, which can cause vehicle's deviation from its nominal trajectory.

continues to apply the control policy $u^{\text{PP}}(\cdot)$ 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 $u^{\text{PP}}(\cdot)$ 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 $\bar{k}$ 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 $\mathcal{N}^{\text{RP}} = \{Q_1, Q_2\}$. 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 $\bar{k} = 3$, 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 $u^{\text{PP}}(\cdot)$, 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 $u^{\text{PP}}(\cdot)$ 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

[1] M. Chen, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Robust sequential trajectory planning under disturbances and adversarial intruder," *Transactions on Control Systems Technology*, pp. 1–17, 2018.

[2] B. Tice, "Unmanned aerial vehicles: The force multiplier of the 1990s," *Airpower Journal*, 1991.

[3] W. DeBusk, "Unmanned aerial vehicle systems for disaster relief: Tornado alley," in *Infotech@ Aerospace Conferences*, 2010.

[4] Amazon.com, Inc., "Amazon Prime Air," 2016. [Online]. Available: http://www.amazon.com/b?node=8037720011

[5] AUVSI News, "UAS aid in South Carolina tornado investigation," 2016. [Online]. Available: http://www.auvsi.org/blogs/auvsi-news/2016/01/29/tornado

[6] BBC Technology, "Google plans drone delivery service for 2017," 2016. [Online]. Available: http://www.bbc.com/news/technology-34704868

[7] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," *Transactions on Robotics*, vol. 32, no. 6, pp. 1498–1511, 2016.

[8] S. Hayat, E. Yanmaz, T. X. Brown, and C. Bettstetter, "Multi-objective UAV path planning for search and rescue," in *International Conference on Robotics and Automation*, 2017.

[9] Joint Planning and Development Office, "Unmanned Aircraft Systems (UAS) comprehensive plan," Federal Aviation Administration, Tech. Rep., 2014.

[10] T. Prevot, J. Rios, P. Kopardekar, J. Robinson III, M. Johnson, and J. Jung, "UAS Traffic Management (UTM) concept of operations to safely enable low altitude flight operations," in *AIAA Aviation Technology, Integration, and Operations Conference*, 2016.

[11] P. Chandler, S. Rasmussen, and M. Pachter, "UAV cooperative path planning," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2000, p. 4370.

[12] M. Radmanesh and M. Kumar, "Flight formation of UAVs in presence of moving obstacles using fast-dynamic mixed integer linear programming," *Aerospace Science and Technology*, vol. 50, pp. 149–160, 2016.

[13] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, July 1998.

[14] G. Chasparis and J. Shamma, "Linear-programming-based multi-vehicle path planning with adversaries," in *American Control Conference*, 2005.

[15] J. Van den Berg, L. Ming, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *International Conference on Robotics and Automation*, 2008.

[16] A. Wu and J. How, "Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles," *Autonomous Robots*, vol. 32, no. 3, pp. 227–242, 2012.

[17] R. Olfati-Saber and R. Murray, "Distributed cooperative control of multiple vehicle formations using structural potential functions," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 495–500, 2002.

[18] Y. Chuang, Y. Huang, M. D'Orsogna, and A. Bertozzi, "Multi-vehicle flocking: Scalability of cooperative control algorithms using pairwise potentials," in *International Conference on Robotics and Automation*, 2007.

[19] M. Radmanesh, P. H. Guentert, M. Kumar, and K. Cohen, "Analytical pde based trajectory planning for unmanned air vehicles in dynamic hostile environments," in *American Control Conference*, 2017.

[20] R. Beard and T. McLain, "Multiple UAV cooperative search under collision avoidance and limited range communication constraints," in *Conference on Decision and Control*, 2003.

[21] T. Schouwenaars and E. Feron, "Decentralized cooperative trajectory planning of multiple aircraft with hard safety guarantees," in *AIAA Guidance, Navigation and Control Conference*, 2004.

[22] D. Stipanovic, P. Hokayem, M. Spong, and D. Siljak, "Cooperative avoidance control for multiagent systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, p. 699, 2007.

[23] M. Massink and N. De Francesco, "Modelling free flight with collision avoidance," in *Conference on Engineering of Complex Computer Systems*, 2001.

[24] M. Althoff and J. Dolan, "Set-based computation of vehicle behaviors for the online verification of autonomous vehicles," in *Conference on Intelligent Transportation Systems*, 2011.

[25] P. Barooah, P. G. Mehta, and J. P. Hespanha, "Mistuning-based control design to improve closed-loop stability margin of vehicular platoons," *Transactions on Automatic Control*, vol. 54, no. 9, pp. 2100–2113, 2009.

[26] Y. Lin and S. Saripalli, "Collision avoidance for UAVs using reachable sets," in *Conference on Unmanned Aircraft Systems*, 2015.

[27] P. Frihauf and M. Krstic, "Leader-enabled deployment onto planar curves: A PDE-based approach," *Transactions on Automatic Control*, vol. 56, no. 8, pp. 1791–1806, 2010.

[28] E. Lalish, K. Morgansen, and T. Tsukamaki, "Decentralized reactive collision avoidance for multiple unicycle-type vehicles," in *American Control Conference*, 2008.

[29] G. Hoffmann and C. Tomlin, "Decentralized cooperative collision avoidance for acceleration constrained vehicles," in *Conference on Decision and Control*, 2008.

[30] M. Chen, J. Shih, and C. Tomlin, "Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming," in *Conference on Decision and Control*, 2016.

[31] F. Lian and R. Murray, "Real-time trajectory generation for the cooperative path planning of multi-vehicle systems," in *Conference on Decision and Control*, 2002.

[32] A. Ahmadzadeh, N. Motee, A. Jadbabaie, and G. Pappas, "Multi-vehicle path planning in dynamically changing environments," in *International Conference on Robotics and Automation*, 2009.

[33] J. Bellingham, M. Tillerson, M. Alighanbari, and J. How, "Cooperative path planning for multiple UAVs in dynamic and uncertain environments," in *Conference on Decision and Control*, 2002.

[34] D. Panagou, M. Turpin, and V. Kumar, "Decentralized goal assignment and trajectory generation in multi-robot networks: A multiple Lyapunov functions approach," in *International Conference on Robotics and Automation*, 2014.

[35] H. Min, F. Sun, and F. Niu, "Decentralized UAV formation tracking flight control using gyroscopic force," in *Conference on Computational Intelligence for Measurement Systems and Applications*, 2009.

[36] R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson, "Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1306–1324, 2006.

[37] D. Panagou, D. M. Stipanović, and P. G. Voulgaris, "Distributed coordination control for multi-robot networks using Lyapunov-like barrier functions," *Transactions on Automatic Control*, vol. 61, no. 3, pp. 617–632, 2015.

[38] X. Gu, Y. Zhang, J. Chen, and L. Shen, "Real-time decentralized cooperative robust trajectory planning for multiple UCAVs air-to-ground target attack," *Journal of Aerospace Engineering*, vol. 229, no. 4, pp. 581–600, 2015.

[39] Y. Kuwata and J. P. How, "Cooperative distributed robust trajectory optimization using receding horizon MILP," *Transactions on Control Systems Technology*, vol. 19, no. 2, pp. 423–431, 2010.

[40] M. Innocenti, L. Pollini, and A. Bracci, "Cooperative path planning and task assignment for unmanned air vehicles," *Journal of Aerospace Engineering*, vol. 224, no. 2, pp. 121–131, 2010.

[41] I. Kaminer, O. Yakimenko, V. Dobrokhodov, A. Pascoal, N. Hovakimyan, V. Patel, C. Cao, and A. Young, "Coordinated path following for time-critical missions of multiple UAVs via L1 adaptive output feedback controllers," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007, p. 6409.

[42] T. W. McLain and R. W. Beard, "Coordination variables, coordination functions, and cooperative timing missions," *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 1, pp. 150–161, 2005.

[43] E. Barron, "Differential games with maximum cost," *Nonlinear analysis: Theory, methods & applications*, vol. 14, no. 11, pp. 971–989, 1990.

[44] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[45] O. Bokanowski, N. Forcadel, and H. Zidani, "Reachability and minimal times for state constrained nonlinear problems without any controllability assumption," *Journal on Control and Optimization*, vol. 48, no. 7, pp. 4292–4316, 2010.

[46] O. Bokanowski and H. Zidani, "Minimal time problems with moving targets and obstacles," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 2589–2593, 2011.

[47] K. Margellos and J. Lygeros, "Hamilton-Jacobi formulation for reach-avoid differential games," *Transactions on Automatic Control*, vol. 56, no. 8, pp. 1849–1861, 2011.

[48] J. Fisac, M. Chen, C. Tomlin, and S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Conference on Hybrid Systems: Computation and Control*, 2015.

[49] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[50] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2006.

[51] I. Mitchell, "Application of level set methods to control and reachability problems in continuous and hybrid systems," Ph.D. dissertation, Stanford University, 2002.

[52] ——, "A toolbox of level set methods," *Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, http://www. cs. ubc. ca/˜ mitchell/ToolboxLS/toolboxLS.pdf, Tech. Rep. TR-2004-09*, 2004.

[53] A. Bayen, I. Mitchell, M. Osihi, and C. Tomlin, "Aircraft autolander safety analysis through optimal control-based reach set computation," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 1, pp. 68–77, 2007.

[54] J. Ding, J. Sprinkle, S. Sastry, and C. Tomlin, "Reachability calculations for automated aerial refueling," in *Conference on Decision and Control*, 2008.

[55] P. Bouffard, "On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments," Master's thesis, University of California, Berkeley, 2012.

[56] H. Huang, J. Ding, W. Zhang, and C. Tomlin, "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag," in *International Conference on Robotics and Automation*, 2011.

[57] M. Chen, J. Fisac, S. Sastry, and C. Tomlin, "Safe sequential path planning of multi-vehicle systems via double-obstacle Hamilton-Jacobi-Isaacs variational inequality," in *European Control Conference*, 2015.

[58] S. Bansal, M. Chen, J. Fisac, and C. Tomlin, "Safe sequential path planning of multi-vehicle systems under presence of disturbances and imperfect information," in *American Control Conference*, 2017.

[59] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi reachability: A brief overview and recent advances," in *Conference on Decision and Control*, 2017.

[60] M. G. Crandall and P.-L. Lions, "Two approximations of solutions of Hamilton-Jacobi equations," *Mathematics of Computation*, vol. 43, no. 167, pp. 1–19, Sep. 1984.

[61] S. Bansal, M. Chen, and C. J. Tomlin, "Safe and resilient multi-vehicle trajectory planning under adversarial intruder," *arXiv preprint arXiv:1711.02540*, 2017.

[62] helperOC Team, "helperOC library," https://github.com/HJReachability/helperOC, 2019.

[63] M. Chen, Q. Hu, J. Fisac, K. Akametalu, C. Mackin, and C. Tomlin, "Reachability-based safety and goal satisfaction of unmanned aerial platoons on air highways," *Journal of Guidance, Control, and Dynamics*, pp. 1–14, 2017.

[64] 3D Robotics, "Solo specs: Just the facts," 2015. [Online]. Available: https://news.3dr.com/solo-specs-just-the-facts-14480cb55722#.w7057q926

[65] New Atlas, "Amazon Prime Air." [Online]. Available: http://newatlas.com/amazon-new-delivery-drones-us-faa-approval/36957/

[66] Wikipedia, "Beaufort scale." [Online]. Available: `_scale`https://en.wikipedia.org/wiki/Beaufort\protect\unhbox\voidb@x\bgroup\def_˙scale#Modern