Decentralized Safe Control for Distributed Cyber-Physical Systems using Real-time Reachability Analysis

Luan Viet Nguyen, *Member, IEEE,* Hoang-Dung Tran, *Member, IEEE,* Taylor Johnson, *Member, IEEE,* and Vijay Gupta, *Fellow, IEEE*

***Abstract*— In this paper, we present a decentralized safe control (DSC) approach for distributed cyber-physical systems based on conducting reachability analysis in real-time. Each agent can periodically compute the local reachable set from its current local time to some time instant in the near future, and then broadcast a message containing the computed reachable set to the other agents via a shared communication channel. By comparing its own reachable set to unsafe regions such as obstacles, and received reachable sets in a peer-to-peer manner, the agent can predict if any collision may exist shortly. In this circumstance, the agent can utilize the unsafe intersection of its computed reachable set, the unsafe regions and the received reachable sets to recalculate a new set of waypoints and update its corresponding control strategies. As a result, our DSC approach can ensure a distributed system achieving a mission goal with a collision-free guarantee. For evaluation, we applied the proposed DSC method to perform a decentralized control, in real-time, a group of quadcopters conducting a distributed patrol mission with safety guarantees.**

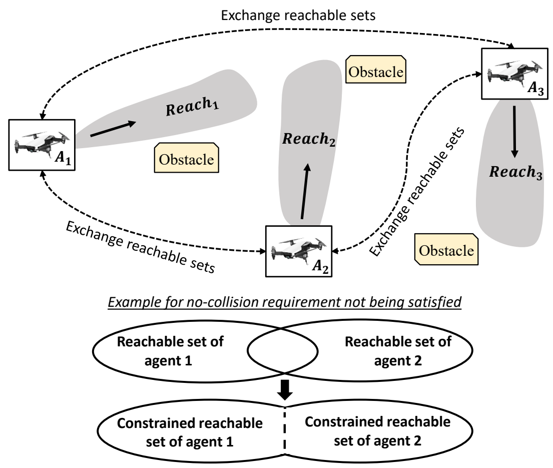


Fig. 1. An overview of our DSC approach based on conducting reachability analysis in real-time. Agents exchange local reachable sets. Any potential safety violations can be identified and local controllers constrained to avoid them in a peer-to-peer manner.

I. Introduction

A cyber-physical system (CPS) consists of computing devices communicating with one another and interacting with the physical world via sensors and actuators. Increasingly, such systems are everywhere, from smart buildings to autonomous vehicles to mission-critical military systems. In these applications, CPSs often operate distributively where each agent can communicate and cooperate with each other via a high-speed communication network. To ensure that the implementation always meets safety-critical requirements, distributed CPSs require an ability to predict, in real-time, maneuvers that cause dangerous circumstances and then update control strategies to avoid that. For instance, if an agent $\mathcal{A}_i$ can predict based on its local clock that it will collide with either an obstacle or an agent $\mathcal{A}_j$ in the next few seconds, it must perform some intelligent control action such as immediately stopping and quickly finding a safe path to avoid an accident. To provide safety guarantees such as collision avoidance for a distributed CPS in real-time, we need to constantly compute a reachable set of the system and check it against correctness requirements. Although several recent works on safety control and path planning [1]–[7], reachability analysis for CPSs [8]–[14], safety monitoring and verification for distributed CPSs [15]–[23] have been introduced, there is a shortage of methodologies that can efficiently control a distributed CPS to achieve its mission objectives with safety guarantees in real-time, especially in a decentralized manner.

In this paper, we propose a DSC approach for distributed CPSs based on conducting reachability analysis in real-time. Our approach considers both local safety constraints (e.g., ranges of velocity and operating boundaries) and global safety requirements (e.g., an agent can avoid unsafe regions and colliding with other agents). The overview of our approach shown in Figure 1. Utilizing the face-lifting method [24], each agent can periodically perform a reachability analysis to compute a local reachable set based on its current local clock to some future time instant; and then check against its local safety properties in real-time. We emphasize that our method considers communication in real-time verification. It allows the computed reachable set of an agent to be encoded in a message and then transferred to other agents via a shared communication channel, with an assumption that all agents are time-synchronized to some accuracy level. This assumption is reasonable as it can be achieved using existing time synchronization protocols such as the Network Time Protocol [25]. When the agent receives the reachable set messages, it immediately decodes them to obtain the reachable sets of the senders. By comparing the agent's own reachable set to the unsafe regions (e.g., obstacles) and the reachable

L. Nguyen is with the Department of Computer Science, University of Dayton, Dayton, OH, 45469, USA (e-mail:lnguyen1@udayton.edu)

H-D. Tran is with the School of Computing, University of Nebraska at Lincoln, Lincoln, NE, 68588 USA (e-mail: dtran30@unl.edu)

T. Johnson is with the Department of Computer Science, Vanderbilt University, Nashville, TN, 37212 USA (e-mail: taylor.johnson@vanderbilt.edu)

V. Gupta is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN, 46556, USA (e-mail: vgupta2@nd.edu)

sets received from other agents in a peer-to-peer manner, the agent can predict if there will be any forthcoming collision. In this circumstance, the agent can utilize the reachable set constraints, i.e., the intersection of its computed reachable set, the unsafe regions and the received reachable sets to recalculate a new set of waypoints and update its corresponding control strategy to safely accomplish a mission. Moreover, we also prove the soundness of the proposed DSC approach.

We implemented our DSC approach as a Java package in StarL [26], which is a novel platform-independent framework for programming and simulating distributed robotic systems on Java/Android. For evaluation, we applied the proposed method to perform a decentralized control, in real-time, a group of three quadcopters conducting a distributed patrol mission with safety guarantees. The experimental results demonstrate that our proposed DSC approach based on reachability analysis can ensure a distributed system achieving a mission goal with a collision-free guarantee in real-time. We also discuss in details a scalability of our approach.

The remainder of the paper is organized as follows. Section II introduces the system modeling of a distributed CPS. Section III formalizes the problem that we address in this paper. Section IV explains how we compute a reachable set for each agent in a real-time manner. Section V presents our reachability analysis-based DSC approach to ensure a distributed CPS achieving a mission goal with safety guarantees in real-time. Section VI presents the implementation and demonstrates the capability of our approach through a case study of a distributed patrol mission. Section VII reviews the related works and Section VIII concludes the paper with some future directions.

II. System Modeling

In this paper, we model a distributed CPS using *hybrid automata* formalism [27], [28]. A distributed CPS with N agents is a network of N hybrid automata where each agent is essentially modeled as a finite state machine extended with a set of real-valued variables evolving continuously over time.

Definition 1 (Hybrid automaton): A hybrid automaton of an i^{th} agent is a tuple $\langle \mathcal{A}_i = Var_i, Act_i, \mathcal{D}_i, \mathcal{T}_i, Init_i \rangle$, which includes the following components:

1) Var_i is the set of variables, partitioned as $I_i \cup X_i \cup Y_i$, where i) I_i is the finite set of input variables ii) X_i is the set of continuous variables including the special variable clk_i which records the agent's *local time*, and iii) Y_i is the set of discrete variables Y_i including the special variable msg_i that records all sent and received messages. We denote $\mathbf{v}_i$ as the valuations (i.e., a function that maps each $v_i \in Var_i$ to a value in its type) of all variables. We write $val(Var_i)$ for the set of all possible valuations of Var_i. We abuse a notion of $\mathbf{v}_i$ to denote a state of $\mathcal{A}_i$, and the set $Q_i \triangleq val(Var_i)$ is the state-space of $\mathcal{A}_i$.
2) Act_i is a set of *actions* consisting of the following subsets: i) a set $\{send_i(m) \mid m \in M_{i,*}\}$ of send actions (i.e., output actions), ii) a set $\{receive_i(m) \mid m \in M_{*,i}\}$ of receive actions (i.e., input actions), where $M_{i,*}$ and $M_{*,i}$ denote the sending and receiving messages by agent i, respectively.
3) $\mathcal{D}_i \subseteq val(Var_i) \times Act_i \times val(Var_i)$ is called the set of *transitions*. For a transition $(\mathbf{v}_i, a_i, \mathbf{v}'_i) \in \mathcal{D}_i$, we write $\mathbf{v}_i \overset{a_i}{\rightarrow} \mathbf{v}'_i$ in short. If $a_i = send_i(m)$ or $receive_i(m)$, then all the components of $\mathbf{v}_i$ and $\mathbf{v}'_i$ are identical except that m is added to msg in $\mathbf{v}'_i$. That is, the agent's other states remain the same on message sends and receives. Furthermore, for every state $\mathbf{v}_i$ and every receive action a_i, there must exist a state $\mathbf{v}'_i$ such that $\mathbf{v}_i \overset{a_i}{\rightarrow} \mathbf{v}'_i$, i.e., the automaton must have well-defined behavior for receiving any message in any state.
4) $\mathcal{T}_i$ is a collection of trajectories for X_i. Each trajectory of X_i is a function mapping an interval of time $[0, t], t \geq 0$ to $val(Var_i)$, following a flow rate (i.e., a differential equation) that specifies how a real variable $x_i \in X_i$ evolving over time with respect to a control input.
5) $Init_i \in Q_i$ is the set of initial states.

The *behavior* of each agent can be defined in terms of *executions*, which are alternating sequences of continuous trajectories and discrete transitions starting from an initial state. Given an initial state $\mathbf{v}_i^0 \in Init_i$, an *execution* α_i of an agent $\mathcal{A}_i$ is a sequence of states starting from $\mathbf{v}_i^0$, defined as $\alpha_i = \mathbf{v}_i^0, \mathbf{v}_i^1, \mathbf{v}_i^2, \ldots$, and for each index k in the sequence, the state update from $\mathbf{v}_i^k$ to $\mathbf{v}_i^{k+1}$ is either a transition or trajectory. A state $\mathbf{v}_i^k$ is *reachable* if there exists a finite execution that ends in $\mathbf{v}_i^k$. We denote Reach_i as the whole reachable set of agent $\mathcal{A}_i$ evolved from the initial state, and $\mathsf{Reach}_i[t_1^i, t_2^i]$ denotes the partial reachable set reached from the state at a local clock t_1^i to the state at t_2^i.

The communication between agents is implemented by the *actions* of sending and receiving messages over an asynchronous communication channel. We formally model this communication model as a single hybrid automaton, $\mathcal{A}_c$, which stores the set of in-flight messages that have been sent, but are yet to be delivered. When an agent sends a message m, it invokes a *send(m)* action. This action adds m to the *in-flight* set. At any arbitrary time, $\mathcal{A}_c$ chooses a message in the in-flight set to either deliver it to its recipient or remove it from the set. All messages are assumed to be unique and each message contains its sender and recipient identities. We denote M as the set of all possible messages used in communication between agents.

The formal model of the complete system, denoted as Σ, is a network of hybrid automata that is obtained by parallel composing the agent's models and the communication channel. Formally, we can write, $\Sigma \triangleq \mathcal{A}_1 \| \ldots \mathcal{A}_N \| \mathcal{A}_c$. The agent $\mathcal{A}_i$ and the communication channel $\mathcal{A}_c$ are synchronized through sending and receiving actions. When the agent A_i sends a message $m \in M_{i,j}$ to the agent A_j, it triggers the $send_i(m)$ action. At the same time, this action is synchronized in the $\mathcal{A}_c$ automaton by putting the message m in the *in-flight* set. After that, the $\mathcal{A}_c$ will trigger (non-deterministically) the $receive_j(m)$ action. This action is synchronized in the agent $\mathcal{A}_j$ by putting the message m into the msg_j.

III. Problem Formulation

The main goal of this work is to provide a DSC for distributed CPSs to achieving mission objectives while ensuring local and global safety properties in real-time. A local safety

$\hat{f}_{e_2}$

property specifies t

an agent, such as ra

Definition 2 (Lo

$\mathcal{A}_i$ satisfies a local

only if $\mathsf{Reach}_i \subseteq \varphi$

On the other h

avoidance requiren

regions (e.g., obsta

an agent) and a pa

Definition 3 (Gl

region, t^i and t^j be

respectively. In the

i) $\mathcal{A}_i$ can avoid the

$\emptyset$, and ii) $\mathcal{A}_i$ will n

$\mathsf{Reach}_j[t^j, t^j + T]$

Without loss of

polytope and does

region $\Upsilon_i[t^i, t^i+T]$

to the received rea

each agent has a de

the global property

mismatches, and ii) the *exchanging reachable set messages* between agents. Each agent can locally and periodically compute the local reachable set from the current local time to the next T seconds, and broadcasts this information to the other agents. When an agent receives such a reachable set message, it can perform a peer-to-peer coupled constraint verification based on its own current state and the reachable set of the sender. For instance, if the reachable sets of the agents do not intersect with each other, the system will satisfy a collision avoidance property. Note that the local safety property of the agent is verified simultaneously with the reachable set computation process at runtime. When safety cannot be guaranteed, an exception is generated and the control must be updated.

In our approach, we consider that the dynamic of each agent has a form $\dot{\mathbf{x}}_i = f(\mathbf{x}_i, \mathbf{u}_i)$, where $\mathbf{x}_i \in \mathbb{R}^n$ is the state vector, $\mathbf{u}_i \in \mathbb{R}^m$ is the control input vector and $f : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is a Lipschitz mapping function. The states $\mathbf{x}_i$ of the agent are measured by the local sensors and the provided GPS. The control signal $\mathbf{u}_i$ is computed based on the state $\mathbf{x}_i$ and a reference signal, e.g., a set of waypoints that the agent needs to follow to accomplish a mission, and then applied to the actuator to control the motion of the agent. We note that each agent can switch between different operation modes via discrete transitions, and in each mode, the control law may be different. When the agent computes its reachable set, the only information it needs are its current set of states $\mathbf{x}_i(t^i)$ and the current control input $\mathbf{u}_i(t^i)$. It should be clarified that although the control law may be changed for different operation modes, the control signal $\mathbf{u}_i$ is updated with the same control period T_c^i. Consequently, $\mathbf{u}_i$ is a constant vector in each control period.

Problem 1: Design a DSC mechanism using real-time reachability analysis such that any agent $\mathcal{A}_i$ of a distributed CPS both satisfies its local safety property and global safety property (i.e., $\mathcal{A}_i$ avoids the unsafe region and does not collide with other agent) in the next (finite number of) T seconds from the current local clock of $\mathcal{A}_i$.

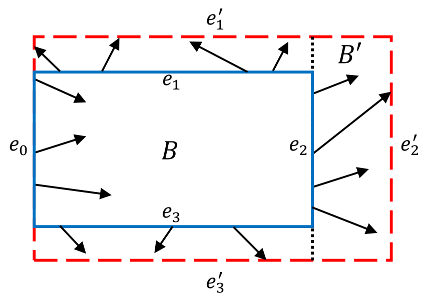


Fig. 2. An illustration of a real-time reachability analysis using face-lifting method. The reachable set evolved from the blue box $\mathcal{B}$ to the red dash box $\mathcal{B}'$ by lifting each face of $\mathcal{B}$ by its maximum outward components of f.

IV. Real-time Reachable Set Computation

In this section, we present our algorithm to compute reachable set for each agent in a real-time manner. Assume that the agent's current time is $t^i = k \times T_c^i$, and from its local sensors and provided GPS, we have the current state of the agent as $\mathbf{x}_i$. The actual state of the agent is in a set $\mathbf{x}_i \in \mathbf{X}_i$. From the current set of states $\mathbf{X}_i$ and the control signal $\mathbf{u}_i$, we can compute the forward reachable set of the agent to the time instance $t^i + T$. This reachable set computation needs to be completed after an allowable runtime $T_{run}^i < T_c^i$ because if $T_{run}^i \geq T_c^i$, a new $\mathbf{u}_i$ will be updated. The control period T_c^i is chosen based on the agent's motion dynamics. Thus, to control an agent with fast dynamics, the control period T_c^i needs to be sufficiently small. As a result, the allowable runtime for reachable set computation is also tiny. We note that the local sensors and the provided GPS can only measured the information of interest to some accuracy.

A. The Face-Lifting Approach

We adapt the well-known face-lifting method [24] to compute the reachable set of an agent in real-time. The core idea is to over-approximately compute a reachable set based on the maximum derivative along each face. Given a box $\mathcal{B}$ represents a reachable set at time t, assume that $\hat{f}_e$ is the positive, maximum outward component of a Lipschitz flow rate f relative to a face e of $\mathcal{B}$, then the new box $\mathcal{B}'$ represents a reachable set at time $t+\Delta$ is constructed by lifting every face e by an amount of $\Delta.\hat{f}_e + \epsilon$, where ϵ is some small amount number that can be tuned. The overview of the face-lifting method shown in Figure 2 illustrates how the reachable set evolved from the blue box $\mathcal{B}$ to the red dash box $\mathcal{B}'$. In the figure, the black arrows illustrate the directions and values of f on each face of the blue box $\mathcal{B}$. Here, the faces e_1, e_2 and e_3 have positive outward components of f and then being lifted to e_1', e_2' and e_3', respectively. Despite the derivative component of f along e_0 moves into the box $\mathcal{B}$, i.e., it does not add new reachable states, the face must be extended vertically to construct a new box together with its neighborhoods. We use a box as our presentation for the set of states as we want to estimate a reachable set in a very short reach time where the over-approximation error is sufficient small enough. But one can apply the face-lifting with a general polyhedron to approximate a reachable set with higher accuracy but a longer reach time. We want to emphasize that the novelty of this paper is to provide a decentralized control mechanism for distributed systems based on conducting real-time reachability analysis.

Although the face-lifting method is conservative, we chose it to estimate a reachable set because of its fast computational advantage for real-time applications. One can extends our approach by using other methods such as those presented in [29]–[33] to compute reachable sets online. In what follows, we will show our method to reduce the conservativeness by attractively improve the accuracy of a computed reachable set if time remains.

B. *Real-time Reachable Set Computation*

Next, we will explain our Algorithm 1 to compute a reachable set and check a local safety property for each agent in real-time. We first divide the time period $[t^i, t^i + T]$ by M steps, where $\Delta_i = T/M$ is reach time step. The core step of face-lifting method is the *single-face-lifting* operation illustrated in Figure 2. Using the reach time step, the current state set $\mathbf{X}_i$ and control input, the face-lifting method performs a single-face-lifting operation to compute a new reachable set $\mathbf{\Omega}_i$ and update the *remaining reach time* $T^i_\alpha < T$. This step is iteratively called until the reachable set for the whole time period of interest $[t^i, t^i + T]$ is constructed completely, i.e., $T^i_\alpha = 0$. Note that with the reach time step Δ_i defined above, the face-lifting algorithm may be finished quickly after an amount of time which is smaller than the allowable runtime T^i_{run} specified by a user. In this case, there is still an amount of time called the *remaining run time* $T^i_\beta < T^i_{run}$ that is available to recall the face-lifting algorithm with a smaller reach time step. It is important to note that if T^i_β is still greater than zero, Algorithm 1 will keep calling the face-lifting mechanism with a new reach time step $\Delta_i/2$. As a result, the conservativeness of the reachable set can be iteratively improved as the over-approximate errors will be significantly reduced. Note that the reachable set computation needs to be completed after an allowable runtime T^i_{run}. If this constraint is not met, the algorithm returns the current reachable set, i.e., there is nothing change in the agent's reachable set.

As mentioned earlier, the local safety property of each agent can be verified at runtime simultaneously with reachable set computation process. Precisely, let φ_i be the local safey property of the i^{th} agent, the agent is said to be safe from t^i to $t^i + t \leq t^i + T$ if $\mathsf{Reach}_i[t^i, t^i + t] \subseteq \varphi_i$. Since the reachable set $\mathsf{Reach}_i[t^i, t^i + t]$ is provided by face-lifting method at runtime, the local safety verification of each agent can be solved at runtime. Since the Algorithm 1 computes an over-approximate reachable set of each agent in a short time interval, it guarantees the soundness of the result, i.e., the computed reachable set contains all possible trajectories of agent $\mathcal{A}_i$ from t^i to $t^i + T$.

V. Decentralized Reachability Analysis-based Control

The working principle of our DSC approach is based on the exchanged reachable set messages between agents. Each agent $\mathcal{A}_i$ will perform the following activities.

1) Each agent computes its reachable set for the next (finite number of) T seconds corresponding to the current control input, and check against its local safety properties.

Algorithm 1 Real-time reachable computation for agent $\mathcal{A}_i$.

Input: $\mathbf{X}_i$, $\mathbf{u}_i$, t^i, T, Δ_i, T^i_{run}, φ_i
Output: $\mathsf{Reach}_i[t^i, t^i + T]$, safe

```
1:  procedure Real-time Reachability Analysis
2:      $T^i_\beta = T^i_{run}$            % remaining allowable runtime
3:      while ($T^i_\beta > 0$) do
4:          safe = $true$       % initially suppose $\mathcal{A}_i \models \varphi_i$
5:          $\mathbf{\Omega}_i = \mathbf{X}_i$          % current reachable set
6:          $T^i_\alpha = T$            % remaining reach time
7:          while $T^i_\alpha > 0$ do
8:              % do single-face-lifting to compute reach set,
                  and update remaining reach time
9:              $\mathbf{\Omega}_i, T^i_\alpha = \mathsf{SingleFaceLifting}(\mathbf{\Omega}_i, \Delta_i, T^i_\alpha, \mathbf{u}_i)$
10:             if ($\mathbf{\Omega}_i \not\subseteq \varphi_i$) then: safe = $false$
11:         % update remaining runtime wrt. $\mathcal{A}_i$'s current time
12:         $T^i_\beta = T^i_\beta - (\mathsf{Clock}(\mathcal{A}_i) - t^i)$
13:         if $T^i_\beta \leq 0$ then break
14:         else
15:             $\Delta_i = \Delta_i/2$       % reduce reach time step
16:     return $\mathsf{Reach}_i[t^i, t^i + T] = \mathbf{\Omega}_i$, safe
```

2) Next, the agent encodes the computed reachable set as a message, then broadcasts it to other agents and listens to the reachable set messages sent from them.
3) When the agent receives the messages, it immediately decodes the message to obtain the current reachable set of the senders.
4) By comparing its own reachable set to the unsafe regions and the received reachable sets in a peer-to-peer manner, the agent can predict if any collision may exist in the near future. In this case, the agent will recalculate a set of waypoints and update its control strategy in the next control period to avoid a collision.

We note that the agent must accomplish all of the above activities within one control period of T^i_c. It guarantees that the agent can always update a new control signal in the next control period if necessary.

A. *Task Scheduling Architecture*

Next, we will explain our real-time task scheduling architecture for an agent, shown in Figure 3. There are six specific tasks that the agent $\mathcal{A}_i$ needs to accomplish in order as: i) computing reachable set starts at the time instant t^i_r ii) encoding reachable set starts at the time instant t^i_e iii) transferring reachable set starts at the time instant t^i_t iv) decoding reachable set from other agents starts at the time instant t^i_d v) checking a collision avoidance property starts at the time instant t^i_c vi) recalculating waypoints and updating a control signal at the time instant t^i_w. All time instants associated with each activity are specified based on the agent $\mathcal{A}_i$'s local clock t^i. Here, we assume $t^i_r = t^i = k \times T^i_c$ and each activity must be accomplished in order. Then actual real-time intervals for each activity are specified as follows i) $\delta^i_r = t^i_e - t^i_r$ is the reachable set computation time ii) $\delta^i_e = t^i_t - t^i_e$ is the encoding time iii) $\delta^i_t \approx t^j_d - t^i_t$ is the transferring time iv) $\delta^i_d = t^i_c - t^i_d$

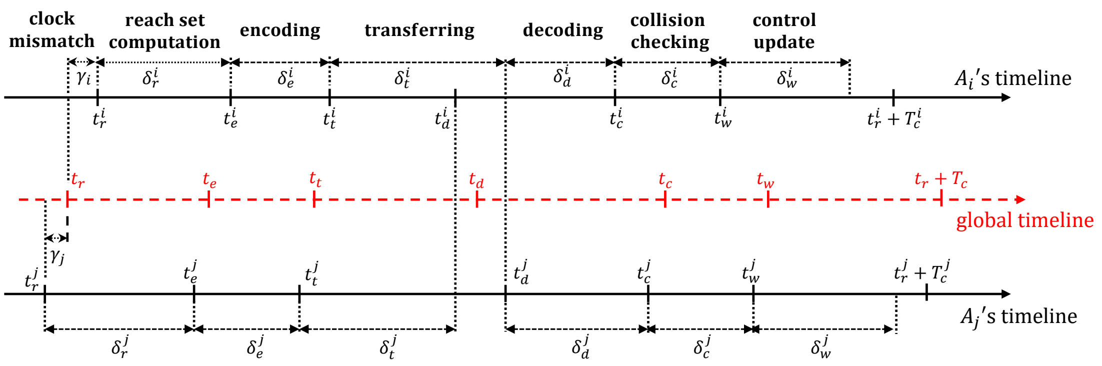


Fig. 3. A real-time task scheduling architecture for each agent of a distributed CPS with respect to the agent's local clock and the system's global clock.

is the decoding time v) $\delta_c^i = t_w^i - t_c^i$ is the collision checking time vi) $\delta_w^i \leq t^i + T_c^i - t_w^i$ is the time required for updating waypoints and control signals.

Note that we do not know the exact transferring time δ_t^i since it depends on two different local clocks. The transferring time formula describes its approximation value when neglecting the mismatch between two local clocks. The actual reachable set computation time is close to the allowable runtime chosen by a user, i.e., $\delta_r^i \approx T_{run}^i$. The encoding time and decoding time are fairly small in comparison with the transferring time, i.e., $\delta_e^i \approx \delta_d^i \ll \delta_t^i$. All of these runtimes are useful information for selecting an appropriate control period T_c^i for an agent. However, for the goal of ensuring a distributed CPS achieving a mission goal with a guarantee of the collision avoidance property, we only need to consider the time instants t_r^i, t_c^i, and t_w^i where an agent starts computing reachable set, checking collision, and updating a control signal, respectively.

B. Reachable Set Message

A reachable set message contains three important information including i) the reachable set which is a collection of boxes ii) the time period (based on the local clock) in which this reachable set is valid, i.e., the start time t_r^i and the end time $t_r^i + T$ of the computed reachable set, and iii) the time instant t_t^i that this message is transferred. Based on the timing information of the reachable set and the time-synchronization errors, an agent can examine whether or not a received reachable set contains information about the future behavior of the sent agent, which is useful for checking collision and updating a control signal. The usefulness of the received reachable sets is defined as follows.

Definition 4 (Useful reachable sets): Let γ_i and γ_j respectively be the time-synchronization errors of agent $\mathcal{A}_i$ and $\mathcal{A}_j$ in comparison with the *virtual global time* t, i.e, $t - \gamma_i \leq t^i \leq t + \gamma_i$ and $t - \gamma_j \leq t^j \leq t + \gamma_j$, where t^i and t^j are current local times of $\mathcal{A}_i$ and $\mathcal{A}_j$, respectively. Given the reachable set $\mathsf{Reach}_i[t_r^i, t_r^i + T]$ of the agent $\mathcal{A}_i$, the reachable set $\mathsf{Reach}_j[t_r^j, t_r^j + T]$ of the agent $\mathcal{A}_j$ that is available at the agent $\mathcal{A}_i$ at time t_c^i is *useful* for checking collision between $\mathcal{A}_i$ and $\mathcal{A}_j$ if: i) $t_c^i < t_r^j + T - \gamma_i - \gamma_j$ and ii) $t_c^i < t_r^i + T$.

Assume that we are at the time instant when the agent $\mathcal{A}_i$ checks if a collision occurs which means, the current local time is t_c^i. Note that agent $\mathcal{A}_i$ and $\mathcal{A}_j$ are synchronized to the global time within some errors γ_i and γ_j, respectively. The reachable set $\mathsf{Reach}_j[t_r^j, t_r^j + T]$ is useful if it contains some information about the *future behavior* of agent $\mathcal{A}_j$ under the view of the agent $\mathcal{A}_i$ based on its local clock. This can be guaranteed if we have: $t_r^j + T \geq t_r^i - \gamma_j + T > t_c^i + \gamma_i$. Additionally, the current reachable set of agent $\mathcal{A}_i$ contains information about its future behavior if $t_c^i < t_r^i + T$. It is apparent that if $t_c^i > t_r^j + T + \gamma_i + \gamma_j$, then the reachable set of $\mathcal{A}_j$ contains a past information, and thus it is useless for comparing their reachable sets.

Remark 5: In our DSC approach, we allow each activity to be accomplished with runtime flexibility without considering either fixed deadlines or worst-case execution time, as long as all activities can be finished within a control period. Consequently, the proposed approach does not rely on the concept of Lamport happens-before relation [34] to compute the local reachable set of each agent. If the agent could not receive reachable messages from others until a requested time-stamp expires, it still calculates the local reachable set based on its current state and the state information of other agents in the messages received previously. In other words, our method does not require the reachable set of each agent to be computed corresponding to the order of the events (sending or receiving a message) in the system, but only relies on the local clock period and the time-synchronization errors between agents. Such implementation ensures that the computation process can be accomplished in real-time, and is not affected by the message transmission delay.

C. Decentralized Reachability Analysis-based Control

The overview of our DSC approach shown in Algorithm 2, and works as follows. First, the agent $\mathcal{A}_i$ call Algorithm 1 to compute its reachable set in the next T seconds. The intersection Ψ^i between $\mathsf{Reach}_i[t_r^i, t_r^i + T]$ and the unsafe region Υ_i is then determined. When the agent $\mathcal{A}_i$ receives a new message from the agent $\mathcal{A}_j$, $\mathcal{A}_i$ decodes the message to obtain reachable set information of $\mathcal{A}_j$. If the obtained reachable set of $\mathcal{A}_j$ is useful, i.e., $t_c^i < t_r^j + T - \gamma_i - \gamma_j$ and $t_c^i < t_r^i + T$, $\mathcal{A}_i$ will then determine the intersection $\Theta^{ij}[t_r^i, t_r^i + T]$ between its current reachable and the received one. If the union of Ψ^i and $\Theta^{ij}[t_r^i, t_r^i + T]$ is empty, then there will be no collision in the next T seconds. Otherwise, the agent $\mathcal{A}_i$ will recalculate a new set of waypoints list and update its control signal in the next control period to safely achieving its mission objectives based on Ψ^i and $\Theta^{ij}[t_r^i, t_r^i + T]$.

Algorithm 2 Decentralized Safe Control using Real-time Reachability Analysis at Agent $\mathcal{A}_i$.

Input: a current set of states $\mathbf{X}_i$, an unsafe set Υ_i, received reachable messages $\text{Reach}_j[t_r^j, t_r^j + T]$, $j \in \{1, 2, \ldots, N\}$, and $j \neq i$, a current set of waypoints $\mathcal{W}_i$
Output: a new control signal $\mathbf{u}_i$ and set of waypoints $\mathcal{W}_i^{'}$

```
1: procedure REACHABILITY-BASED CONTROL
2:     $\text{Reach}_i[t_r^i, t_r^i + T] \leftarrow$ Algorithm 1
3:     $\Psi^i \leftarrow \text{Reach}_i[t_r^i, t_r^i + T] \cap \Upsilon_i$
4:     for every received reachable message
       $\text{Reach}_j[t_r^j, t_r^j + T]$ of $\mathcal{A}_j$ do
5:         $\Theta^{ij}[t_r^i, t_r^i + T] \leftarrow \emptyset$
6:         % decode message and check usefulness
7:         if $\text{Reach}_j[t_r^j, t_r^j + T]$ is useful then
8:             % determine an intersection of reachable sets
9:             $\Theta^{ij}[t_r^i, t_r^i + T] \leftarrow \text{Reach}_i[t_r^i, t_r^i + T]$
                                 $\cap\ \text{Reach}_j[t_r^j, t_r^j + T]$
```

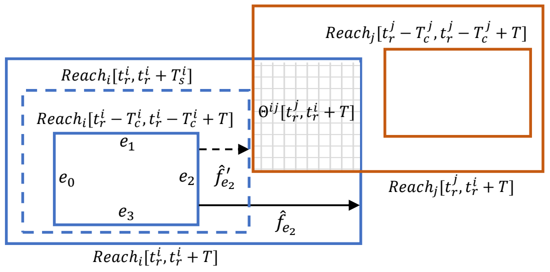

$Reach_j[t_r^j - T_c^j, t_r^j - T_c^j + T]$
$Reach_i[t_r^i, t_r^i + T_s^i]$
$Reach_i[t_r^i - T_c^i, t_r^i - T_c^i + T]$
$\Theta^{ij}[t_r^j, t_r^i + T]$
e_1
e_0
e_2
$\hat{f}_{e_2}'$
e_3
$Reach_j[t_r^j, t_r^i + T]$
$\hat{f}_{e_2}$
$Reach_i[t_r^i, t_r^i + T]$

Fig. 4. An illustration of a potential collision between the agents $\mathcal{A}_i$ and $\mathcal{A}_j$. The intersection $\Theta^{ij}[t_r^i, t_r^i + T]$ is not empty because the positive outward derivative $\hat{f}_{e_2}$ has a large value that causes the crossing.

Figure 4 illustrates an example where $\Theta^{ij}[t_r^i, t_r^i + T]$ is not empty, i.e., there is an intersection between the computed reachable set of $\mathcal{A}_i$ and the received reachable set from $\mathcal{A}_j$. Here, the positive outward derivative (denoted as $\hat{f}_{e_2}$) of the face e_2 has a large value that causes the crossing. The WPUpdate function takes as input $\Theta^{ij}[t_r^i, t_r^i + T]$ to calculate the maximum amount (denoted as $\hat{f}_{e_2}^{'}$) that the face e_2 can be lifted to the right, and determine a safe time interval T_s^i from t_r^i such that $\mathcal{A}_i$ can safely move forward based on its current state and input signal. Moreover, one can see that any outward extensions of other faces will be safe in this case. Taking into accounts of the information acquired from analyzing $\Theta^{ij}[t_r^i, t_r^i + T]$, the current set of waypoints $\mathcal{W}_i$ is then modified to avoid a potential collision. For instances, the agent $\mathcal{A}_i$ can move further to the right by the amount defined by $\hat{f}_{e_2}^{'}$ and T_s^i, then travel in other directions. We note that WPUpdate function will try to calculate new a feasible path with the least effort to achieving a mission goal with a collision-free guarantee instead of finding an optimal solution. Consequently, the new control signal $\mathbf{u}_i$ is computed based on the current state set of $\mathbf{X}_i$ and the updated set of waypoints $\mathcal{W}_i^{'}$. Note that in this example, we assume that the intersection Ψ^i between $\text{Reach}_i[t_r^i, t_r^i + T]$ and the unsafe region Υ_i is empty. If Ψ^i is not empty, the values of $\hat{f}_{e_2}^{'}$ and T_s^i can be also determined similarly based on analyzing the union $\Psi^i \cup \Theta^{ij}[t_r^i, t_r^i + T]$. The following assumption gives a condition where an agent can always calculate a new feasible list of waypoints and update its control signal to achieve a mission within the allowable time. It also guarantees the continuity of the status of the agent when performing the control update.

Assumption 6: The agent $\mathcal{A}_i$ can always calculate a new feasible list of waypoints and update its control signal to achieve a mission within the time interval $\delta_w^i \leq t_r^i + T_c^i - t_w^i$.

Theorem 7 (Soundness): Suppose that the reachable set intersection $\text{Reach}_i[t_r^i, t_r^i + T] \cap (\text{Reach}_j[t_r^j, t_r^j + T] \cup \Upsilon_i) \neq \emptyset$. Let T_s^i be a safe time interval from t_r^i such that i) $T_s^i \leq \min(T + t_r^j - \gamma_i - \gamma_j - t_r^i, T)$, and ii) $\text{Reach}_i[t_r^i, t_r^i + T_s^i] \cap (\text{Reach}_j[t_r^j, t_r^j + T_s^i] \cup \Upsilon_i) = \emptyset$. The Algorithm 2 is sound in the sense that if $T_s^i \geq T_c$ then $\mathcal{A}_i$ can achieve its mission without either entering the unsafe region Υ_i or hitting $\mathcal{A}_j$.

Proof: We have that $\text{Reach}_i[t_r^i, t_r^i + T]$ contains all possible trajectories of the agent $\mathcal{A}_i$ from t_r^i to $t_r^i + T$, and $\text{Reach}_j[t_r^j, t_r^j + T]$ also includes all possible trajectories of the agent $\mathcal{A}_j$ from t_r^j to $t_r^j + T$. Since $T_s^i < \min(T + t_r^j - \gamma_i - \gamma_j - t_r^i, T)$ so $\text{Reach}_i[t_r^i, t_r^i + T_s^i] \subset \text{Reach}_i[t_r^i, t_r^i + T]$ and $\text{Reach}_j[t^j, t^j + T_s^i] \subset \text{Reach}_j[t_r^j, t_r^j + T]$ that contain all possible trajectories of the agent $\mathcal{A}_i$ and $\mathcal{A}_j$ up to a safe interval T_s^i from t_r^i, respectively. Because $\text{Reach}_i[t_r^i, t_r^i + T_s^i] \cap (\text{Reach}_j[t_r^j, t_r^j + T_s^i] \cup \Upsilon_i) = \emptyset$, the collision avoidance property is always satisfied in the time interval $[t_r^i, t_r^i + T_s^i]$. In addition, if $T_s^i \geq T_c > t_w^i - t_r^i + \delta_w^i$, so $\mathcal{A}_i$ has enough time to recalculate a feasible waypoints to avoid Υ_i and a potential collision with $\mathcal{A}_j$ based on Assumption 6. Consequently, the condition $T_s^i \geq T_c$ guarantees that the agent can always update control signal in the next control period to avoid a collision.

VI. Case Study

In this section, we demonstrate the capability of our approach in performing a real-time decentralized control for a group of quadcopters conducting a distributed patrol mission with safety guarantees. Figure 5 shows a scenario where three quadcopters $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ cooperatively conduct a patrol mission to cover an oil spill following three distinct segments $A - B - C$, $C - D - E$ and $E - F - A$, respectively. Each segment represents a set of initial way-points provided by a user where each agent travels back and forth on it.

The dynamics of a quadcopter is given by the equations for acceleration in the three-dimensions coordination (x, y, z) as follows [35].

$$\ddot{x} = (\frac{f}{m})(\sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi)),$$
$$\ddot{y} = (\frac{f}{m})(\sin(\psi)\sin(\theta)\cos(\phi) - \sin(\phi)\cos(\theta)),$$
$$\ddot{z} = (\frac{f}{m})\cos(\theta)\cos(\phi) - g,$$

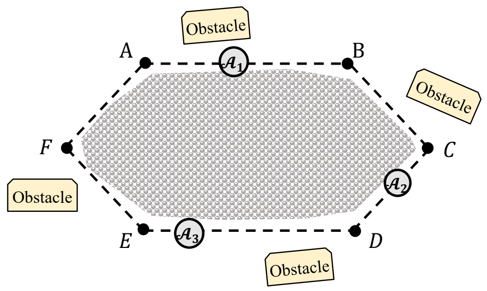


Fig. 5. A group of three quadcopters conducting a distributed patrol mission with safety guarantees.

where θ, ϕ, and ψ are the pitch, roll, and yaw angles in the body frame, f is sum of the propeller forces, m is the mass of the quad-copter, and $g = 9.81m/s^2$ is the gravitational acceleration constant. We assume that the height of the quadcopter is constant and only planar motion to waypoints in the (x, y) plane is desired, i.e., $\ddot{z} = 0$ and there is no acceleration in the vertical direction. This assumption yields: $f = \frac{mg}{\cos(\theta)\cos(\phi)}$.

The quadcopter dynamics can be further simplified by assuming the quadcopter's planar orientation ψ is zero. This assumption is reasonable in practice since the quadcopter's planar orientation ψ can be measured and controlled to stay within a range $[-\sigma, +\sigma]$, where σ is a very small value. Substituting the above equation with $\psi = 0$, we have the simplified model of the quadcopter as $\ddot{x} = g \times \tan(\theta)$, and $\ddot{y} = g \times \frac{\tan(\phi)}{\cos(\theta)}$.

To control the position of the quadcopter in the (x, y) plane, two PID controllers are designed to supply the control inputs (i.e., the pitch θ and the roll ϕ) for the quadcopter. In every control period, θ and ϕ are computed based on the current position of the quadcopter and the current target position (i.e.,, the current way-points it needs to follow). Based on the control inputs, the current positions and velocities information given from GPS and the motion dynamics of the quadcopters, each agent executes the real-time reachable set computation algorithm (Algorithm 1) *inside* the controller. This algorithm computes the reachable set of the quadcopter from its current local time to the next T seconds specified by a user.

Implementation and Experiment Setting. The implementation of our proposed approach and all experimental results are available at https://github.com/LuanVietNguyen/DSC-CPS. We implemented our approach as a Java package in StarL [26], which is a novel platform-independent framework for programming and simulating distributed robotic systems on Java/Android. StarL is specifically suitable for controlling a distributed network of robots over WiFi since it provides many useful functions and sophisticated algorithms for distributed applications such as point-to-point motion, mutual exclusion, registration and geocast. In our approach, we abuse the reliable communication network of StarL assumed to be asynchronous and peer-to-peer. There may be message dropouts and transmission delays. However, we assume that every sent message is eventually delivered to an agent with some time guarantees.

For the experiment, the PID controller of each agent has control period of $T_c^i = T_c = 200$ milliseconds. Each agent computes a reachable set from its current local time to the next $T = 2$ seconds with an initial reach time step $\Delta = 1$ milliseconds, and the allowable runtime for the computation is $T_{run} = 10$ milliseconds. The computed reachable set is then encoded and sent to another quadcopter with a transmission delay less than 2 milliseconds, and there is no message dropouts. When a reachable set message arrives, the quadcopter immediately decodes the message to reconstruct the current reachable set of the sender. The GPS error is assumed to be 2%. The time-synchronization error between the quadcopters' clocks and the virtual global clock is $\gamma = 3$ milliseconds. All experiments were performed with Android Studio 3.5 executed on an x86-64 laptop with 2.8 GHz Intel(R) Core(TM) i7-7700HQ processor and 32 GB RAM.

Experiment Scenario and Results. Suppose that the agent $\mathcal{A}_3$ escapes at the middle of operation and there are only two agents $\mathcal{A}_1$ and $\mathcal{A}_2$ to continue conducting a patrol mission. Thus, each of them will cover the new segments $A-B-C-D$ and $D-E-F-A$ respectively. Suddenly, $\mathcal{A}_3$ comes back at E and travels toward D, while $\mathcal{A}_2$ also approaches E. Therefore, there may be a potential collision between $\mathcal{A}_2$ and $\mathcal{A}_3$, and the control must be updated to avoid that.

Figure 6 shows the *intermediate* reachable sets of the three quadcopters $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ in the next $T = 2$ seconds time interval corresponding to their local clocks t^1, t^2 and t^3, respectively. These intermediate reachable sets (apparently shown in the zoom area) are represented by lists of hyper-rectangles which are essential for verifying the local safety property at runtime. Here the local safety property $\varphi \triangleq \dot{x}_i < 100 \wedge \dot{y}_i < 100$, (i.e., the maximum allowable velocities along the x-axis and y-axis of each agent are less than 100) is always satisfied. For checking the global safety property, we use the interval hull of these hyper-rectangles. Because the intermediate reachable set may contain hundreds of hyper-rectangles, so it is too large to be transferred via a network. Instead, we transfer the corresponding interval hull that covers all possible trajectories of a quadcopter in the time interval of 2 seconds. We note that transferring the interval hull instead of the convex hull of the reachable set increases the approximation errors and yields more conservative results. However that significantly reduces a transfer time, and is then more practical in performing a real-time distributed control for safety-critical systems.

In Figure 6, we can observe that the intersection between the computed reachable sets of A_2 and A_3 is not empty, so there will be a potential collision between them in the next 2 seconds. The intersection indicates that the maximum horizontal position that A_2 can travel to the right and A_3 can move to the left at $x \approx 1861$ from their current positions to avoid a collision. Also, the safe time instances in the future of A_2 and A_3 are $T_s^1 = 1.415$ seconds and $T_s^2 = 0.8474$ seconds, respectively. The agents use these useful information to update their waypoints. Since T_s^1 and T_s^2 are both greater than the control period T_c, a reasonable control strategy for each agent is that A_2 and A_3 will stop at the next control period. Then, A_2 will travel back toward the point E and perform a patrol mission on the segment $A - F - E$ while A_3 will move back toward the point D and continue a patrol mission on the segment $B - C - D$. Consequently, each agent recalculates

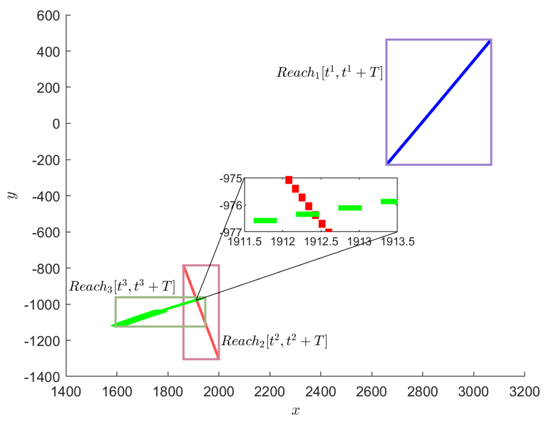


Fig. 6. The blue, red and green reachable sets of the three quadcopters $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ in the next $T = 2$ seconds time interval corresponding to their local clocks t^1, t^2 and t^3, respectively. The zoomed area shows a very short-time interval reachable set of the quadcopters. As the intersection between the computed reachable sets of A_2 and A_3 is not empty so there will be a potential collision between them in the next 2 seconds.

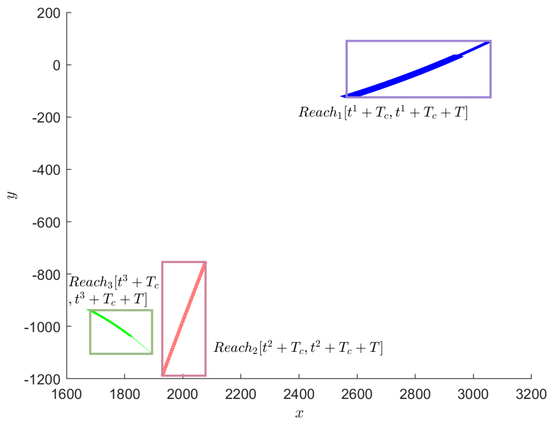


Fig. 7. The reachable sets of the three quadcopters $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ in the next $T = 2$ seconds time interval corresponding to their local clocks $t^1 + T_c$, $t^2 + T_c$ and $t^3 + T_c$, respectively. After updating the control signals, the computed reachable sets of A_2 and A_3 is no longer intersected.

the new list of waypoints and the PID controller then updates a control signal to drive the motion of the agent following the new reference signal. Figure 7 shows the reachable sets of the three quadcopters in the next control period. In this figure, the computed reachable sets of $\mathcal{A}_2$ and $\mathcal{A}_3$ in the next 2 seconds is no longer intersected so the system will continue satisfying its safety properties. We note that the proposed approach can be also applied to unmanned ground vehicles and fixed-wing aircrafts. Indeed, their controllers can be modeled as hybrid automata including different operation modes (e.g., left, right, straight and stop), and the vehicle dynamics can be modeled as a planar-body motion with two degrees of freedom, similarly to the platform of quadrotors presented in this paper.

Scalability. Next, we investigate the scalability of our approach. For a distributed system with N agents, the total time required to performing all activities (including reachable set computing, encoding, transferring decoding, checking for a collision, recalculating waypoints and updating a control signal) of each agent is

$$T^i_{total} = T^i_{run} + \delta^i_e + (N-1) \times (\delta^i_t + \delta^i_d + \delta^i_c + \delta^i_w).$$

For our case study, with $T^i_{run} = 10$ milliseconds, the total time for each agent (T^i_{total}) are $T^1_{total} = 33.6232$ milliseconds, $T^2_{total} = 37.9$ milliseconds and $T^3_{total} = 38.9363$ milliseconds, respectively. We can observe that T^i_{total} is significant less than the control period of $T^i_c = 200$ milliseconds, then the procedures of reachable set computation, collision checking, waypoints and control updates do not affect the performance of the PID controller. Based on the equation of T^i_{total}, we can estimate the lower bound of the number of agents that our real-time reachability analysis-based control approach can handle. Let $\bar{\delta}_e$, $\bar{\delta}_t$, $\bar{\delta}_d$, $\bar{\delta}_c$ and $\bar{\delta}_w$ be the maximum times allocated for reachable set encoding, transferring decoding, checking for a collision, recalculating waypoints and updating a control signal, then the number of agents, at least that our approach can deal with

$$\bar{N} = \frac{T_c - T_{run} - \bar{\delta}_e}{\bar{\delta}_t + \bar{\delta}_d + \bar{\delta}_c + \bar{\delta}_w} + 1.$$

VII. Related Works

In this section, we situate our proposed methodology to the existing literature in the three following categories.

Safe Control and Shield Synthesis for Distributed CPSs. There are only a few works that can provide an efficient control for distributed CPSs with safety guarantees. The authors of [3] proposed a decentralized control algorithm that uses vehicle-to-vehicle communication to determine whether automatic control is needed to prevent a collision. However, the approach is limited to a particular scenario of two-vehicle collision avoidance, where the vehicle dynamics are piecewise continuous. Although the work proposed in [7] introduces an interesting distributed control framework for a real-time path planning under adversarial environment, it does not provide a real-time safe guarantee for multi-agent systems. Sadigh et al. [5] introduced an offline algorithm to synthesize safe controllers given the constraints derived from the probabilistic signal temporal logic specifications, while our approach can perform an online DSC based on conducting real-time reachability analysis, where the agent's dynamics is nonlinear. Bharadwaj et al. [36] presented the automatic construction of minimum-cost shield to enforce safety for multi-agent systems. Another shield synthesis approach proposed in [37] can be used to ensure the safety in a reinforcement learning setting for real-timed systems.

Monitoring, and Verification for Distributed CPSs. Examples of safety monitoring and verification for distributed CPSs include the formal approach for verifying parameterized protocols in mobile CPSs introduced in [16], the distributed graph queries for runtime monitoring of safety-critical CPSs presented in [17], and the formal modeling and offline verification based on model checking and Multirate PALS methodologies proposed in [38]. The work of [20] utilizes contraction theory and convex optimization to generate certifiably safe trajectories for robotic systems with nonlinear dynamics subject to bounded disturbances, input and online state constraints. Convex optimization was also applied to compute fail-safe trajectories in real-time for self-driving cars in [21]. Although these works can provide safety guarantee for specific distributed CPSs in a centralized manner, the main difference from our approach is that they do not provide an adaptive decentralized control for a generic distributed CPS to achieve its mission objectives with safety guarantees during runtime operation.

The approaches proposed in [29]–[31] are very first works using set-based verification and prediction for a system of automated vehicles, whose dynamics can be linearized. Althoff, et al. [32] introduced the set of interconnectable modules which enforces the safety of assembled robots through self-programming and self-verification. The most related to our work is the online verification proposed in [33] which utilizes CORA toolbox [10] to perform a reachability analysis that can guarantee safe motion of mobile robots with respective to walking pedestrians modeled as hybrid systems. However, this work does not consider a time-elapse for encoding, transferring and decoding the reachable set messages between each agent, which plays an important role in distributed systems.

Reachability Analysis Techniques and Tools. There are several significant research and tools have been introduced to compute the set of reachable states for a distributed CPS modeled as a network of hybrid automata, where the motion dynamics can be either linear (e.g., d/dt [39] and SpaceEx [40]) or nonlinear (e.g., Flow*[13] and dReach[14]). The works presented in [41], [42] can be also used to verify distributed CPSs. However, all of these approaches do not consider a real-time aspect and have expensive computation cost. Our decentralized control approach utilizes the face-lifting technique [24] to efficiently compute the reachable set of an agent with a high precision degree in real-time.

VIII. Conclusion and Future Works

In this paper, we have presented a DSC approach based on conducting real-time reachability analysis for distributed CPSs. Each agent can periodically compute the local reachable set from the current local time to some time instant in the near future, and then broadcast a message containing the computed reachable set to the other agents via a shared communication channel. When an agent receives such a reachable set message, it can perform a peer-to-peer coupled constraint collision checking based on its own current state and the reachable set of the sender. The agent can also compare its computed reachable set to unsafe regions such as obstacles. If the agent detects a potential collision, it will recalculate waypoints and update the corresponding control signal in the next control period to avoid that happens. We demonstrated the applicability of our approach by using it to efficiently perform a real-time decentralized control for a group of quadcopters conducting a distributed patrol mission with safety guarantees. We also discussed the scalability of our approach.

Future Works. To increase scalability and practicability of the proposed method, we plan to i) model all activities as a periodic scheduling model with a period, a deadline, and a bound on execution time, ii) exploit the benefit of parallel processing, i.e., each agent simultaneously handles multiple reachable set messages and collision checks, and iii) cope with network communication issues such as message drop-outs and arbitrary communication delays. iv) deploy our DSC method on a real-platform and extend it to work with distributed learning-enabled CPSs that have heterogeneous agents whose motion dynamics are unknown, but can be learned and verified [43]–[45].

Acknowledgments

The material presented in this paper is based upon work supported through Defense Advanced Research Projects Agency (DARPA) contract number FA8750-18-C-0089, Air Force Office of Scientific Research (AFOSR) award FA9550-22-1-0019, the National Science Foundation (NSF) grant numbers 2028001 and 2220418. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of DARPA, AFOSR, nor NSF.

References

[1] T. Fraichard and H. Asama, “Inevitable collision states—a step towards safer robots?” *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.

[2] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, “The system-level simplex architecture for improved real-time embedded system safety,” in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2009, pp. 99–107.

[3] M. R. Hafner, D. Cunningham, L. Caminiti, and D. Del Vecchio, “Cooperative collision avoidance at intersections: Algorithms and experiments,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1162–1175, 2013.

[4] L. V. Nguyen, H.-D. Tran, and T. T. Johnson, “Virtual prototyping for distributed control of a fault-tolerant modular multilevel inverter for photovoltaics,” *IEEE Transactions on Energy Conversion*, vol. 29, no. 4, pp. 841–850, 2014.

[5] D. Sadigh and A. Kapoor, “Safe control under uncertainty with probabilistic signal temporal logic,” in *Proceedings of Robotics: Science and Systems XII*, June 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/safe-control-uncertainty-probabilistic-signal-temporal-logic/

[6] P. Zhou, D. Zuo, K.-M. Hou, and Z. Zhang, “A decentralized compositional framework for dependable decision process in self-managed cyber physical systems,” *Sensors*, vol. 17, no. 11, p. 2580, 2017.

[7] M. Abdelkader, H. Jaleel, and J. S. Shamma, “A distributed framework for real time path planning in practical multi-agent systems,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 626–10 631, 2017.

[8] C. Le Guernic and A. Girard, “Reachability analysis of hybrid systems using support functions,” in *Computer Aided Verification*. Springer, 2009, pp. 540–554.

[9] A. Girard, C. Le Guernic, and O. Maler, “Efficient computation of reachable sets of linear time-invariant systems with inputs,” in *Hybrid Systems: Computation and Control*. Springer, 2006, pp. 257–271.

[10] M. Althoff, “An introduction to cora 2015,” in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[11] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, “Hytech: A model checker for hybrid systems,” in *Computer aided verification*. Springer, 1997, pp. 460–463.

[12] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, “Spaceex: Scalable verification of hybrid systems,” in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 379–395.

[13] X. Chen, E. Ábrahám, and S. Sankaranarayanan, “Flow*: An analyzer for non-linear hybrid systems,” in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 258–263.

[14] S. Kong, S. Gao, W. Chen, and E. Clarke, “dreach: δ-reachability analysis for hybrid systems,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 200–205.

[15] S. Balasubramaniyan, S. Srinivasan, F. Buonopane, B. Subathra, J. Vain, and S. Ramaswamy, “Design and verification of cyber-physical systems using truetime, evolutionary optimization and uppaal,” *Microprocessors and microsystems*, vol. 42, pp. 37–48, 2016.

[16] L. Zhang, W. Hu, W. Qu, Y. Guo, and S. Li, “A formal approach to verify parameterized protocols in mobile cyber-physical systems,” *Mobile Information Systems*, vol. 2017, 2017.

[17] M. Búr, G. Szilágyi, A. Vörös, and D. Varró, “Distributed graph queries for runtime monitoring of cyber-physical systems,” in *International Conference on Fundamental Approaches to Software Engineering*. Springer, Cham, 2018, pp. 111–128.

[18] H.-D. Tran, L. V. Nguyen, P. Musau, W. Xiang, and T. T. Johnson, “Decentralized real-time safety verification for distributed cyber-physical systems,” in *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*. Springer, 2019, pp. 261–277.

[19] A. Majumdar and R. Tedrake, “Funnel libraries for real-time robust feedback motion planning,” *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[20] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, “Robust online motion planning via contraction theory and convex optimization,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5883–5890.

[21] C. Pek and M. Althoff, “Computationally efficient fail-safe trajectory planning for self-driving vehicles using convex optimization,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1447–1454.

[22] D. Fridovich-Keil, S. L. Herbert, J. F. Fisac, S. Deglurkar, and C. J. Tomlin, “Planning, fast and slow: A framework for adaptive real-time safe trajectory planning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 387–394.

[23] S. Manzinger and M. Althoff, “Tactical decision making for cooperative vehicles using reachable sets,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 444–451.

[24] T. Dang and O. Maler, “Reachability analysis via face lifting,” in *Hybrid Systems: Computation and Control (HSCC ’98)*. Springer, 1998, pp. 96–109, lNCS 1386.

[25] D. L. Mills, *Computer network time synchronization: the network time protocol on earth and in space*. CRC press, 2017.

[26] Y. Lin and S. Mitra, “Starl: Towards a unified framework for programming, simulating and verifying distributed robotic systems,” in *Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems 2015 CD-ROM*, 2015, pp. 1–10.

[27] T. A. Henzinger, “The theory of hybrid automata,” in *IEEE Symposium on Logic in Computer Science (LICS)*. Washington, DC, USA: IEEE Computer Society, 1996, p. 278.

[28] N. Lynch, R. Segala, F. Vaandrager, and H. B. Weinberg, *Hybrid i/o automata*. Springer, 1996.

[29] M. Althoff and J. M. Dolan, “Online verification of automated road vehicles using reachability analysis,” *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[30] M. Althoff and S. Magdici, “Set-based prediction of traffic participants on arbitrary road networks,” *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 187–202, 2016.

[31] D. Althoff, M. Althoff, and S. Scherer, “Online safety verification of trajectories for unmanned flight with offline computed robust invariant sets,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3470–3477.

[32] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, “Effortless creation of safe robots from modules through self-programming and self-verification,” *Science Robotics*, vol. 4, no. 31, 2019.

[33] S. B. Liu, H. Roehm, C. Heinzemann, I. Lütkebohle, J. Oehlerking, and M. Althoff, “Provably safe motion of mobile robots in human environments,” in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 1351–1357.

[34] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[35] T. Luukkonen, “Modelling and control of quadcopter,” *Independent research project in applied mathematics, Espoo*, 2011.

[36] S. Bharadwaj, R. Bloem, R. Dimitrova, B. Konighofer, and U. Topcu, “Synthesis of minimum-cost shields for multi-agent systems,” in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 1048–1055.

[37] R. Bloem, P. G. Jensen, B. Könighofer, K. G. Larsen, F. Lorber, and A. Palmisano, “It’s time to play safe: Shield synthesis for timed systems,” *arXiv preprint arXiv:2006.16688*, 2020.

[38] K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky, “Designing and verifying distributed cyber-physical systems using multirate pals: An airplane turning control system case study,” *Science of Computer Programming*, vol. 103, pp. 13–50, 2015.

[39] E. Asarin, T. Dang, and O. Maler, “The d/dt tool for verification of hybrid systems,” in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 365–370.

[40] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, “SpaceEx: Scalable verification of hybrid systems,” in *Computer Aided Verification (CAV)*, ser. LNCS. Springer, 2011.

[41] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele, “A hybrid approach to cyber-physical systems verification,” in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 688–696.

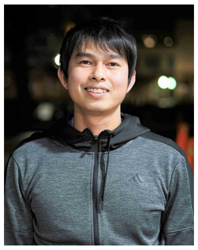
[42] S. M. Loos, A. Platzer, and L. Nistor, “Adaptive cruise control: Hybrid, distributed, and now formally verified,” in *International Symposium on Formal Methods*. Springer, 2011, pp. 42–56.

[43] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, “NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems,” in *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[44] H.-D. Tran, P. Musau, D. M. Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, “Parallelizable reachability analysis algorithms for feed-forward neural networks,” in *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*. IEEE, 2019, pp. 51–60.

[45] ——, “Star-based reachability analysis for deep neural networks,” in *23rd International Symposium on Formal Methods (FM’19)*. Springer International Publishing, October 2019.

Luan V. Nguyen is currently an Assistant Professor at the Department of Computer Science, University of Dayton. He received his Ph.D. degree in Computer Engineering at the University of Texas at Arlington in May 2020. Dr. Nguyen’s research interest is to develop formal verification techniques and state-of-the-art tools to enforce safety, reliability, security and resiliency of autonomous cyber-physical systems (CPS), with practical applications across CPS domains such as automotive, medical devices, aerospace, robotics, power and energy systems.



Hoang-Dung Tran is currently an Assistant Professor in the School of Computing at the University of Nebraska (UNL), Lincoln. He earned a Ph.D. degree in Computer Science at Vanderbilt University in August 2020. His research interests are specification language, formal verification, monitoring, safe control, planning, and learning for autonomous cyber-physical systems (CPS), focusing on in-road and off-road autonomous driving, human-robot-interaction in construction and surgery. His work on verification of learning-enabled CPS has won the prestigious IEEE TCCPS outstanding dissertation award 2021.

Taylor T. Johnson is A. James and Alice B. Clark Foundation Chancellor Faculty Fellow and Associate Professor of Computer Science at Vanderbilt University. He received his B.S.E.E. from Rice University and M.S. and Ph.D. from the University of Illinois at Urbana-Champaign. He is a recipient of the Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) Award and several best paper and artifact awards. His research interests are in formal methods and verification for autonomous CPS that incorporate machine learning components.

Vijay Gupta is in the Elmore Family School of Electrical and Computer Engineering at the Purdue University. He received his B. Tech degree at Indian Institute of Technology, Delhi, and his M.S. and Ph.D. at California Institute of Technology, all in Electrical Engineering. He received the 2018 Antonio J Rubert Award from the IEEE Control Systems Society, the 2013 Donald P. Eckman Award from the American Automatic Control Council and a 2009 National Science Foundation (NSF) CAREER Award. His research interests are broadly at the interface of communication, control, distributed computation, and human decision making.