CoUAS: Enable Cooperation for Unmanned Aerial Systems

ZIYAO HUANG, WEIWEI WU, FENG SHAN, and YUXIN BIAN, Southeast University
KEJIE LU, University of Puerto Rico at Mayagüez
ZHENJIANG LI and JIANPING WANG, City University of Hong Kong
JIN WANG, Soochow University

In the past decade, unmanned aircraft systems (UASs) have been widely used in various civilian applications, most of which involve only a single unmanned aerial vehicle (UAV). In the near future, more and more UAS applications will be facilitated by the cooperation of multiple UAVs. In such applications, it is desirable to utilize a general control platform for cooperative UAVs. However, existing open-source control platforms cannot fulfill such a demand because (1) they only support the leader-follower mode, which limits the design options for fleet control, (2) existing platforms can support only certain type of UAVs and thus lack compatibility, and (3) these platforms cannot accurately simulate a flight mission, which may cause a big gap between simulation and real-world flight. To address these issues, we propose a general control and monitoring platform for cooperative UAS, namely, *CoUAS*, which provides a set of core cooperation services of UAVs, including synchronization, connectivity management, path planning, energy simulation, and so on. To verify the applicability of CoUAS, we design and develop a prototype in which an embedded path planning service is provided to complete any task with the minimum flying time while considering the network connectivity and coverage. Experimental results by both simulation and field test demonstrate that the proposed system is viable.

CCS Concepts: • **Networks** → **Mobile networks**; • **Computer systems organization** → **Robotics**; • **Theory of computation** → *Design and analysis of algorithms*;

Additional Key Words and Phrases: UAV fleet, cooperation, simulation, testbed, path planning, open-source

ACM Reference format:
Ziyao Huang, Weiwei Wu, Feng Shan, Yuxin Bian, Kejie Lu, Zhenjiang Li, Jianping Wang, and Jin Wang. 2020. CoUAS: Enable Cooperation for Unmanned Aerial Systems. *ACM Trans. Sen. Netw.* 16, 3, Article 24 (May 2020), 19 pages.
https://doi.org/10.1145/3388323

The work is supported in part by the National Key Research and Development Program of China under Grant No. 2019YFB2102200, Science Technology and Innovation Committee of Shenzhen Municipality Under project JCYJ20170818095109386, National Natural Science Foundation of China under Grant Nos. 61672154, 61672370, 61972086, Aeronautical Science Foundation of China under Grant No. 2017ZC69011, National Science Foundation under Grant CNS-1730325, and Hong Kong Research Grant Council under GRF 11216618.
Authors' addresses: Z. Huang, W. Wu, F. Shan (corresponding author), and Y. Bian, Southeast University, 2 Sipailou, Nanjing, 210096, P. R. China; emails: {ziyaohuang, weiweiwu, shanfeng, bianyuxin}@seu.edu.cn; K. Lu, University of Puerto Rico at Mayagüez, 259 Norte Alfonso Valdez Cobian Mayaguez, PR 00680, Puerto Rico; email: kejie.lu@upr.edu; Z. Li and J. Wang, 83 Tat Chee Avenue, Kowloon, Hong Kong, P. R. China; emails: {zhenjiang.li, jianwang}@cityu.edu.hk; J. Wang, Soochow University, 1 Shizi Street, Suzhou, 215006, P. R. China; email: wjin1985@suda.edu.cn.

1550-4859/2020/05-ART24 $15.00
https://doi.org/10.1145/3388323

1 INTRODUCTION

An unmanned aircraft system (UAS) consists of one or more unmanned aerial vehicles (UAVs), a ground control station (GCS), and a communication system among them. In recent years, UAS with a fleet of UAVs, especially multi-rotor–based drones, has attracted significant attention from federal agencies, industry, and academia. Many applications can be facilitated by collaborative UAVs [9, 14–16, 21]. For example, in a video surveillance application, multiple UAVs can quickly scan a given area with the help of advanced video processing technologies [18]. Nevertheless, to successfully deploy a multi-UAV application and enable the cooperation among UAVs, many challenging issues must be solved, such as flight control, reliability, safety, and so on [9].

Clearly, to facilitate multi-UAV applications, it is desirable to utilize a general platform to control and monitor UAVs, as illustrated in Figure 1. In the literature, there exist some popular control platforms for UAVs, i.e., GCSs, including Mission Planner [20], QGroundControl [7], and DJI FlightHub [6]. Although all of these GCSs support the basic flight control functionalities, such as flight planning by editing waypoints, communication with UAVs, user-friendly GUIs, flight trajectory displaying on a map, and real-time vehicle status monitoring, the following aspects limit their applicability as general control and monitoring platforms for cooperative UAVs.

- Only leader-follower mode is enabled in the existing GCSs. Though leader-follower mode makes path planning much easier, it cannot fully utilize all UAVs in a fleet to complete a complex task at the earliest time or with shortest flying distance.
- Each GCS only supports a specific set of UAVs or UAV flight controllers. For example, Mission Planner is designed primarily for ArduPilot hardwares and firmwares; QGroundControl only supports UAVs that communicate using the MAVLink protocol; the DJI FlightHub interacts with DJI's own products only.
- There is a lack of energy simulation module in existing GCSs. Without energy simulation module, existing GCSs cannot predict energy consumption through simulation. Thus, the feasibility of a flight cannot be tested before UAVs take off. As a result, some flights may have to be aborted before their tasks are completed due to early energy depletion.

To this end, we propose CoUAS—which is a control and monitor platform that enables easy-to-implement UAV cooperation—to address the aforementioned limitations. Specifically, to address the first limitation and allow UAVs in a fleet to maximize the fleet efficiency, we propose a more generic path planning framework which enables cooperative path planning through swarm functions (e.g., synchronizations, connectivity maintenance) and an embedded path planning service for the multi-UAV cooperation that takes both the UAV network connectivity and coverage into consideration.

To address the second limitation, we provide the hardware-independence to each UAV by introducing a companion Linux-kernel device, which serves as a middleware to interact with UAV autopilots. Since almost every commodity provider and open-source community offers Linux-based SDK for UAV flight control, such UAV companion devices hide the hardware and software difference of UAVs from different manufacturers. Hence, our CoUAS platform is generic enough to work with various UAVs, regardless of their hardwares, firmwares, and communication protocols.

To address the third limitation, we add an energy simulation module to the CoUAS platform. To make the simulation reliable and close to the real-world flight, we make efforts for energy prediction, which can avoid the task abortions in the field. In a fleet with heterogeneous drones, different UAVs may consume different amount of energy even when they fly at the same speed/cover the same distance. Our platform provides an accurate energy model tailored to different types of UAVs, ensuring the feasibility of the real flight under planned paths.

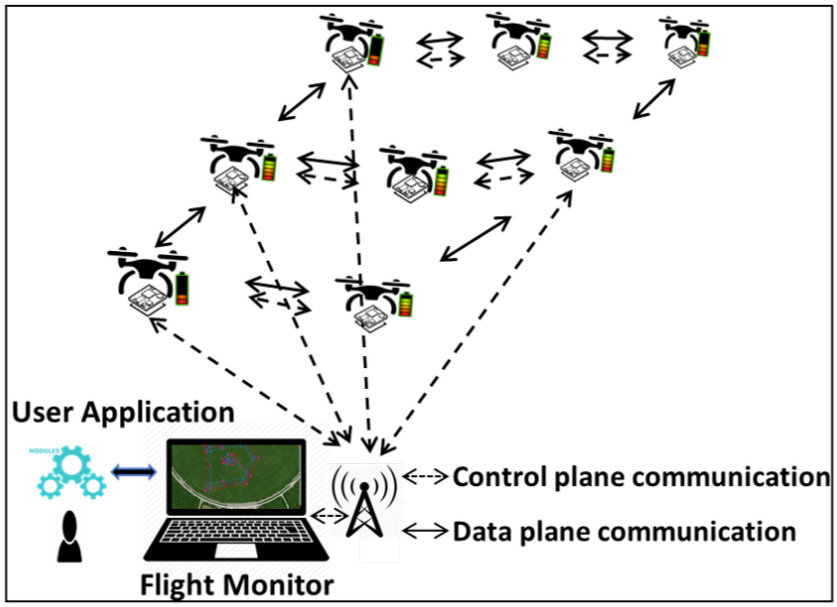


Fig. 1. Architecture of the UAV fleet platform.

Table 1. Comparison of Functionalities among CoUAS and Popular GCSs

	Mission Planner	MAV Proxy	DJI Flight Hub	QGround Control	APM Planner 2	CoUAS
GUI	✓	×	✓	✓	✓	✓
Single-UAV Simulation	✓	✓	✓	✓	✓	✓
UAV APIs[1]	✓	✓	×	×	×	✓
GUI APIs[2]	×	×	×	×	×	✓
Multi-UAV Simulation	×	×	×	×	×	✓
Swarm APIs[3]	×	×	×	×	×	✓
Hadware-independent	×	×	×	×	×	✓
Energy-consumption model	×	×	×	×	×	✓

[1]Interfaces that enable developers to communicate with both UAV and the platform.
[2]Functions that help developers create and control GUI to interact with users and handle the inputs.
[3]Functions that facilitate the cooperation of UAVs.

The key differences of functionalities among CoUAS and popular GCSs are summarized in Table 1. Besides the aforementioned functionalities, CoUAS also offers other features such as path planning for connected fleet, GUI APIs, UAV APIs, and simulation for multiple UAVs, based on which, we implement some basic modules, such as agent manager, emergency monitor, and message center, for ease of application development. A developer can use our platform to achieve rapid development without having to implement the underlying modules.

Our contributions can be summarized as follows:

- CoUAS has the advantage to provide effective cooperation services and manage sophisticated networking protocols. The UAV agent developed in CoUAS includes an independent

middleware that can run on a general operating system. This implies that CoUAS can support not only existing mainstream protocols, but also any specialized airborne communication protocols proposed and developed in the future.

- In addition to the primary cooperation services, CoUAS further embeds path planning service for a fleet, e.g., connectivity maintenance and synchronization during the flight. By taking points to be visited and the number of UAVs as inputs, the path planning service can generate the initial path plan so that the task can be completed at the earliest time while maintaining the connectivity among UAVs. The planned path information is then converted to a series of control commands and disseminated to individual UAVs. When the path is impaired due to environmental factors, e.g., wind disturbance, the planned path can be revised and updated.
- CoUAS provides interfaces to incorporate trained energy models as well as modules to train energy models for different types of UAVs. By collecting energy data through historical flying tasks, we have learned an energy model for Pixhawk-Hexa UAVs. Comparing the simulation results with the field test results, the trained energy model can achieve 94.26% accuracy.
- CoUAS can accurately simulate a flight mission. Moreover, CoUAS supports an easy switch between simulations and field tests during the execution of a task. These advantages come from the system design, where the UAV agent serves as a middleware to hide the UAV hardware difference. As a result, we can replace any UAV models without affecting other parts of the platform, and also use the UAV simulator to conduct simulations prior to the deployment. A demo and the source code of the platform are available for public access in https://github.com/whxru/CoUAS.

The rest of the article is organized as follows: In Section 2, we elaborate on the design and implementation of a prototype. In Section 3, we introduce the main algorithm of the embedded path planning service. To verify the applicability of the proposed system, we then run the algorithm in both simulation and real world environment, the results of which are shown in Section 4. Finally, we discuss related work in Section 5, before concluding the article in Section 6.

2 COUAS SYSTEM DESIGN AND IMPLEMENTATION

The CoUAS platform consists of two types of components: the *UAV agent* installed in each UAV and the *flight monitor* operating on a ground station, as illustrated in Figure 2. In this section, we present the implementation details of the UAV agent and the flight monitor on the CoUAS platform.

2.1 The UAV Agent

A typical UAV or drone system consists of motors, flight control system, gyroscope, compass, GPS, remote control, and battery. The main task of the flight control system is to stabilize the vehicle and control its movement through the control of motors, based on the information from gyroscope, compass, and GPS. The flight control system also provides the drone information and control interfaces to external devices by a pre-defined protocol. As shown in Figure 3, the flight controllers on our current platform are the APM2.8 board and Pixhawk HEXA board. We further install a Raspberry Pi 3 motherboard (RPi) as the mounted Linux-kernel device to run the UAV agent program. The UAV agent is responsible to handle three important types of information or messages, including vehicle status, device control, and exceptions. UAV APIs that communicate between the flight control board and the monitor are provided. The detailed illustration of the UAV agent and its interaction with the UAV flight controller are illustrated in Figure 4.

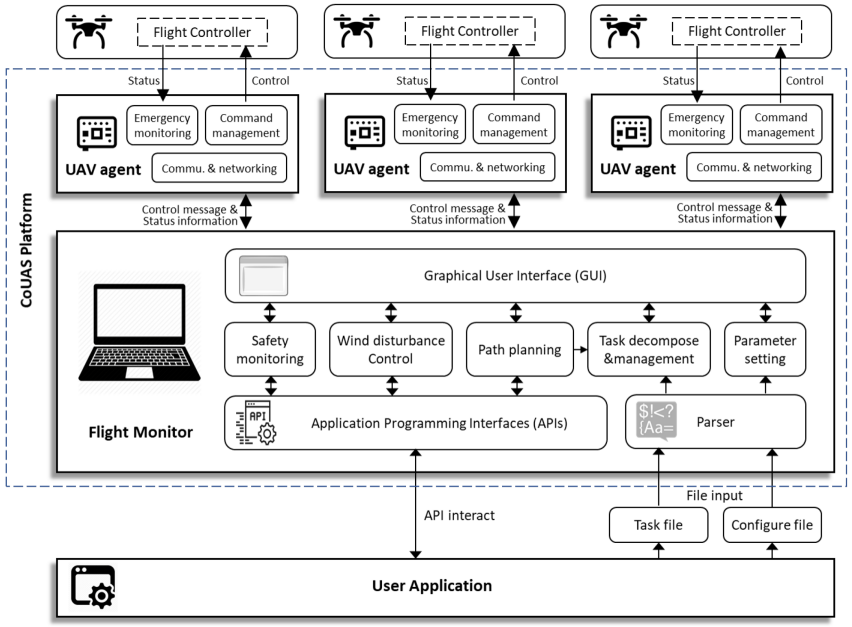


Fig. 2. Illustration of our proposed CoUAS platform.

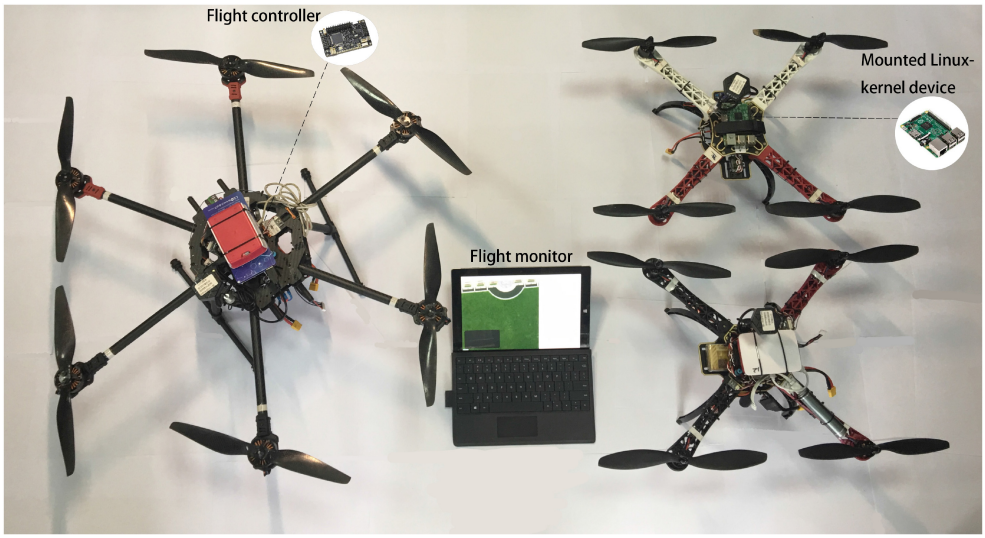


Fig. 3. Hardware components in the CoUAS platform.

2.1.1 Status Information Handling. To be compatible to various underlying flight control boards with hardware differences, we access and update the flight status information through SDKs between the flight control module and the UAV agent program. UAV's status information needs to be transmitted to the flight monitor on the ground. Prior to the transmission, the UAV agent periodically parses the flight status (from SDKs) into the formats needed by the exception monitor

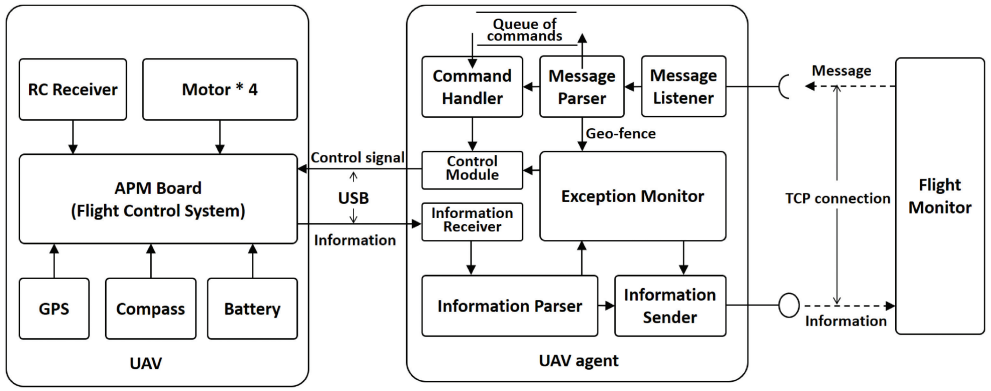


Fig. 4. The detailed illustration of the UAV agent and its interaction with the UAV flight controller.

and the information sender. The information sender further converts it to a character stream for the transmission.

Although the APM2.8 board and Pixhawk HEXA board are used as the flight control in the current CoUAS implementation, our UAV agent design can essentially work with any mainstream UAV flight controllers, because the UAV agent program serves as the middleware that hides the UAV difference from the rest parts of the platform, including the flight monitor. Consequently, for UAVs that utilize other flight controllers, our UAV agent can bridge them to the flight monitor, since almost every commodity provider and open-source community offers Linux-based SDK for UAV flight control.

2.1.2 Control Message Handling. The control message from the flight monitor (on the ground) is transmitted in the format of a character stream, which flows to the message listener of the UAV agent on RPi. There are two types of control messages in CoUAS: *control command* and *parameter setting*. For the former type, commands will be appended to a First-In-First-Out queue. The UAV agent has a command handler that can convert each command into the format that is executable by the flight control module. For the latter type, parameters such as geo-fence boundaries, communication range, and battery life can be handled by the parameter setting message.

2.1.3 Exception Monitoring. The UAV agent also has an exception monitor module and the flight status information is periodically sent to this module for inspection. As a result, the exception monitor can track vehicle's status changes and monitor the emergencies. In case any emergency occurs, the exception monitor either delivers high-priority commands to the flight controller or reports to the flight monitor through the information sender. Exceptions in CoUAS include low battery, crossing the geo-fence boundary, bad health of connection to the monitor application, and so on.

2.2 The Flight Monitor

The main task of the flight monitor is to communicate with each individual UAV and further offer a series of inevitable services for their cooperation. In addition, the flight monitor also provides the interfaces to interact with upper-layer applications and end-users through APIs and GUI, respectively. Figure 5 demonstrates the GUI from the flight monitor in CoUAS platform.

2.2.1 Cooperation Services. According to the information received from each UAV, as illustrated in Figure 6, the service controller in the flight monitor could generate control messages to enable

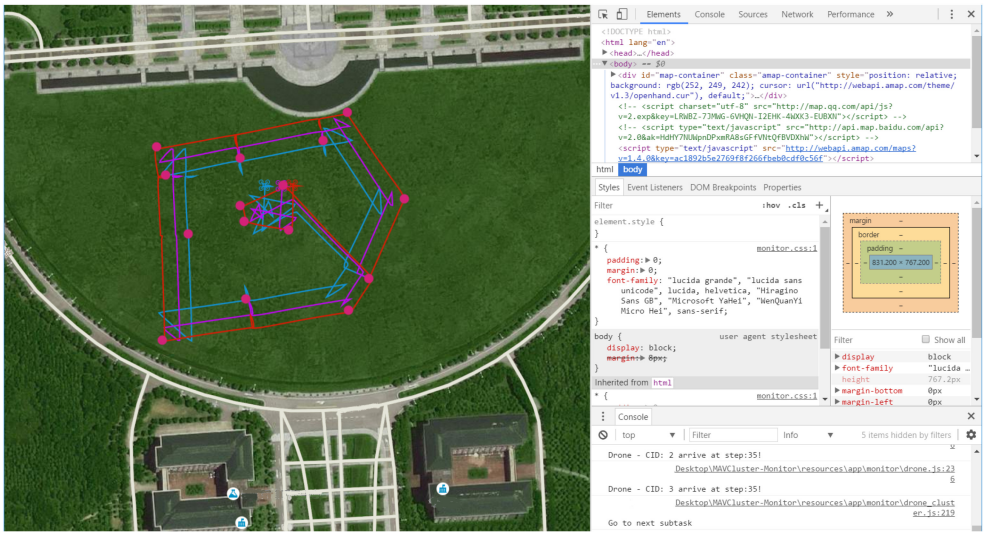


Fig. 5. Example of the GUI of the flight monitor.

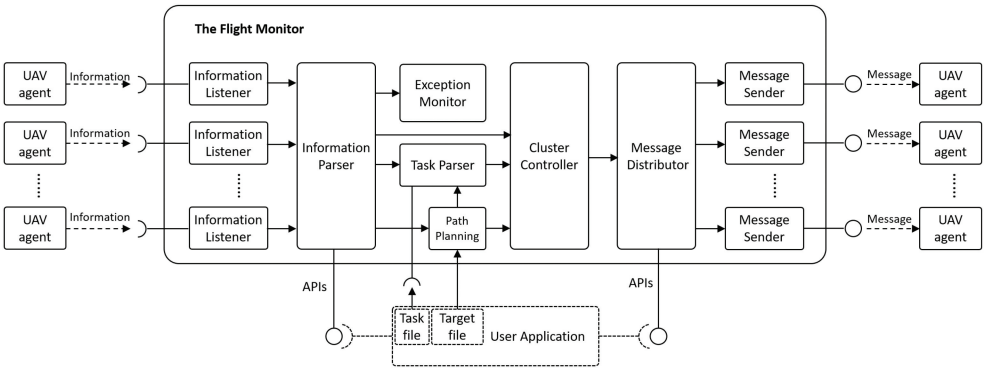


Fig. 6. The detailed illustration of the flight monitor and its interaction with UAV agents and applications.

the following cooperation services. The control messages are high-priority commands to be sent by the message senders through the message distributor.

- Connectivity maintenance: CoUAS enables to connect all UAVs as well as the flight monitor by configuring networking services (such as OLSR) and all the connected UAVs are managed by the agent manager. If the connectivity quality between some UAVs is weak or the transmission errors occur, such exceptions will be thrown and delivered back to the flight monitor to adjust the locations of the UAVs.
- Path planning: A multi-UAV fleet usually needs to visit a target area collaboratively. The path planning service can well schedule the trajectory of each UAV so the fleet can move with connectivity and complete the task from applications at an earlier time. Typically, a set of targets to be covered is passed to this service for processing through GUI or API. Once path planning is done, the generated path will be translated into a sequence of tasks/actions to be executed by each UAV. The algorithm of path planning will be detailed in Section 3.

- Synchronization: We also need a sequence of synchronization values to indicate the status of each UAV and support the connectivity maintenance and cooperation. Each synchronization value is thus a Boolean type, where "true" means a UAV needs to synchronize with other UAVs when finishing the current task.
- Divergence avoidance: Due to the environmental influence (e.g., the wind), UAVs may diverge from their planned trajectories or the original location while hovering in the air. By comparing the real-time trajectory with the planned result, the divergence can be captured and further calibrated.
- Collision avoidance: GPS is widely used to obtain UAV's location. Although GPS is not accurate enough to precisely determine whether two UAVs collided, the collision can be avoided by checking the velocity vectors of any two UAVs and calculating their mutual distances, which should have a sufficient margin for the collision avoidance.

2.2.2 API Interaction. The upper-level user applications can choose to use swarm APIs to implement the aforementioned services. Swarm APIs are implemented mainly by the message center, which handles and delivers information and messages bidirectionally. As shown in Figure 6, the Information Parser module handles every status information from drones, and the Message Distributor module is responsible for handling all control messages to the UAVs. Hence, the status APIs that provide the UAV status information are integrated into the Information Parser and the control APIs that provide the control function are integrated into Message Distributor. To satisfy specific requirements on interacting with users of different tasks, GUI APIs help the developer quickly create visual interfaces are also provided.

2.2.3 File Interaction. The upper-level user applications can also choose to interact with the flight monitor through task files or target files in CoUAS. A task file is an ordered list of actions, while an action consists of a set of key-value pairs. There are two types of pairs, e.g., compulsory pair and optional pair. Compulsory pairs appear in every action, which define the basic of the action. Examples of compulsory pairs include: basic action and its type, connection ID and its value, synchronization and its Boolean value. Optional pairs are important supplementary to the compulsory pairs, but do not necessarily appear in every action. Example of optional pairs include: relative distance and its value, absolute destination and its value. A target file is actually the input of the embedded path planning service that defines a set of targets the user wants to scan with the UAV fleet. The file contains a list of target positions and other configurations such as maximum distance for each pair of UAVs.

2.2.4 UAV Communications. CoUAS enables UAVs as well as the flight monitor to be connected by a high performance Wi-Fi for both the data plane and the control plane. Through the Wi-Fi network, TCP connections are set up for data and control transmission. Each packet is composed of the header and the body. To distinguish different types of packets in CoUAS, we specify a field in the packet header and define seven types of packets, as shown in Table 2. The packet body contains important information, e.g., actions in a sub-task, center position, and radius of the geofence. To support more functions in the future, new types of packets can be defined and added. As UAV agents run on Linux-kernel devices, CoUAS also supports other networking scheme such as the *optimized link state routing protocol* (OLSR) [4], which is very common for setting a wireless mesh network.

2.2.5 Synchronization. To support the cooperation and smoothly execute the tasks cross UAVs, different UAVs synchronize with each other regularly to cope with asynchronous situations (e.g., location deviation, low connectivity quality). The basic idea of the synchronization mechanism is to force UAVs to start their remaining missions simultaneously and synchronize their

Table 2. Types of Packets

Value	Description
0	Request of the connection ID
1	Response to the request of connection ID
2	Report the status of UAV
3	Set the geo-fence
4	Perform action(s)
5	The synchronization signal
6	Signal of closing connection

movement periodically. The accumulated deviations since last synchronization are expected to be calibrated. For example, when a UAV fleet moves with the same speed and direction, their relative positions might vary due to wind disturbance. Thus, we need to periodically adjust their positions. To achieve synchronization, a task assigned to each UAV is divided into a sequence of small subtasks, referred to as steps, where each step contains at most one action. When synchronization index is set to be true, a UAV will be synchronized at the end of the current step (before entering the next step). After the current step is completed, the UAV agent sends a synchronization message to notify the flight monitor that it is ready for the next step and waits until it receives a confirmation message from the monitor. From the perspective of the flight monitor, after collecting all synchronization messages, the flight monitor sends confirmation to all the UAVs where all UAVs waiting for confirmation can then start their next steps. Thus, synchronization can be achieved.

2.2.6 Energy Simulation. In the energy simulation, the predicted energy consumption of a flight in the simulation mode must be able to reflect the situation of real flight. To calibrate the gap between the simulator and the real flight, we train a model by developing a two-step learning framework. The first learning step is to learn a model that maps flying time and distance in real flights to energy consumption. This model is trained through extensive field tests using the nonlinear kernel ridge regression. The flying time and distance generated in the simulations might be different from the flying time and distance in real flights. To calibrate such a gap, we further learn another model to map flying time and distance in simulation to flying time and distance in real flights. To this end, we apply a simple linear regression and learn the parameters that reflect the respective weights/importance of simulated and real flying time/distance. With such a two-step learning model, we are able to map the flying time and distance in simulations to its expected energy consumption with high accuracy.

3 PATH PLANNING WITH CONNECTIVITY AND COVERAGE CONSTRAINTS

In this section, we will detail the algorithm of aforementioned path planning service for the multi-UAV cooperation by considering both the UAV network connectivity and coverage. We will first introduce the problem, then present a near optimal solution and its performance bound.

3.1 Target-point Visiting Problem Formulation

Assume that in a two-dimensional area, a set of target points need to be visited/covered by any UAV of a fleet. Let the point set be $P = \{p_1, p_2, \ldots, p_n\}$, where $p_i = (x_i, y_i)$ refers to the position of the ith target point. Let $U = \{u_1, u_2, \ldots, u_m\}$ be a set of m UAVs. We assume that these UAVs can move freely and we do not consider the vertical movement. At time slot t, the position of u_j is denoted by $u_{j,t} = (x_{j,t}, y_{j,t}), t \in [0, T)$. If one UAV u_j passes through target point p_i, we say that

Table 3. Notations

Symbol	Semantics
P	set of target points
p_i	ith target point
x_i	horizontal position of p_i
x_i	vertical position of p_i
U	set of UAVs
u_i	ith UAV
$u_{j,t}$	position of u_j at time t
$x_{j,t}$	horizontal position of u_j at time t
$y_{j,t}$	vertical position of u_j at time t
d	distance a UAV can move during a flight
ω	UAV's transimission range
$x_{i,j,t}$	whether the distance between u_i and u_j is less than ω at time t
$y_{i,j,t}$	whether target point p_i is scanned by u_j at time t
L_j	total moving distance of u_j

p_i is covered (or scanned) by u_j. The notations are summarized in Table 3. Note, we assume all UAVs move at a constant speed in the planning stage. If a UAV's velocity changes due to wind disturbance during a flight journey, such a change can be captured by the monitoring module and replanning will be triggered.

Since the flight speed of the UAV is limited, the distance that a UAV can move during a time unit should satisfy the *speed constraint*,

$$d(u_{j,t}, u_{j,t+1}) \leq d_{max}, \quad \forall u_j \in U, \ \forall t \in [0, T), \tag{1}$$

where d_{max} is the largest distance that a UAV can move within a time slot.

During a flight, a UAV needs to keep connected with at least one UAV in the fleet or the ground controller. This ensures the information captured by the UAV can be quickly transmitted back to the ground controller and UAVs can cooperatively complete the task. This constraint is referred to as *connectivity constraint*. In other words, the inter-UAV distance should be within the UAVs' transmission range, denoted by w. We use $x_{i,j,t}$ to denote whether the distance between u_i and u_j ($i \neq j$) is less than w at time t, that is,

$$x_{i,j,t} = \begin{cases} 1, & \text{if } d(u_{i,t}, u_{j,t}) \leq w \\ 0, & \text{if } d(u_{i,t}, u_{j,t}) > w \end{cases}. \tag{2}$$

Then the connectivity constraint can be formulated by the following inequality:

$$\sum_{j:u_j \in U, \ j \neq i} d(u_{i,t}, u_{j,t}) \cdot x_{i,j,t} > 0, \quad \forall u_i \in U, \ \forall t \in [0, T). \tag{3}$$

Each target point must be scanned by at least one UAV, referred to be the *coverage constraint*. We use $y_{i,j,t}$ to represent whether target point p_i is scanned by u_j at time t, that is,

$$y_{i,j,t} = \begin{cases} 1, & \text{if } u_{j,t} = p_i \\ 0, & \text{otherwise} \end{cases}. \tag{4}$$

Then the coverage constraint can be denoted by

$$\sum_{j:u_j \in U} \sum_{t:t \in [0,T)} y_{i,j,t} \geq 1, \quad \forall p_i \in P. \tag{5}$$

For UAVs that are not rechargeable during the flight, it is critical to complete the journey as soon as possible. However, considering limited battery supply on UAVs, it is also critical to minimize the longest route among the UAVs. Thus, we define our objective as

$$L_{fleet} = \max_{u_j \in U} L_j, \tag{6}$$

where L_j is the total moving distance of u_j, which can be calculated as $L_j = \sum_{t=0}^{T-1} d(u_{j,t}, u_{j,t+1})$.

We formally define the multi-UAV target-point visiting problem as follows:

Definition 1. Given a fixed number of UAVs and a set of target points distributed in an area, the multi-UAV target-point visiting problem is to plan the trajectory that each drone should move along at each time, to minimize the longest route,

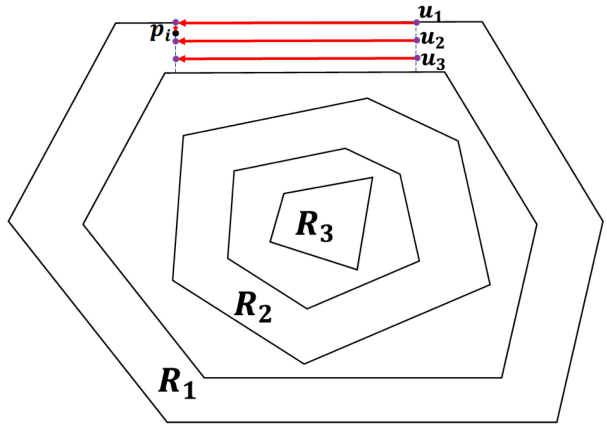
$$\min \max_{u_j \in U} L_j$$
$$\textit{subject to Equations } (1)(3)(5).$$

3.2 Polygon-guided Scanning Algorithm

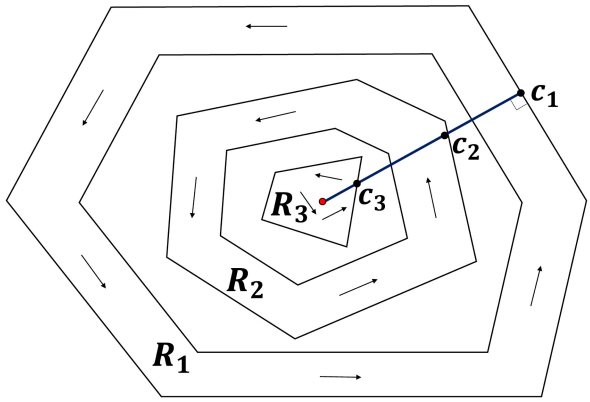
In this section, we propose a UAV fleet planning algorithm, called *Polygon-guided Scanning Algorithm* (PSA), to cooperatively visit the target points as soon as possible. We note that no matter what kind of planning method is applied, the set of points that constitutes the convex hull of all the target points should be scanned. Thus, our high-level idea is to let the UAV fleet move along the convex hull of the target points and cover the maximum possible range under communication connectivity constraint. Since there may exist target points that cannot be covered in the first round of scanning, the fleet will iteratively scan the remaining target points round-by-round. Such a scanning process would partition the area into sub-areas through iterations. Observing that the optimal solution needs to incur a travel distance that is at least the perimeter of the convex polygon, we will design a partition method that ensures the scanned area in each iteration/round is the maximum possible area, making the travel distance of the fleet approach that of the optimal solution. In what follows, we will first introduce the partition of the area and the overall scanning scheme for the UAV fleet, then we will introduce the detailed scheme for a round of scanning. Finally, we will give the performance analysis of the proposed algorithm.

3.2.1 Overall Scanning Scheme. Given all the current non-scanned target points, we can compute the corresponding convex hull, the minimum convex set that contains all the target points to be scanned. Let $V_k = \{v_{k,1}, v_{k,2}, \ldots, v_{k,h_k}\}$ be the computed convex hull in the kth round of scanning, where h_k is the number of vertices of the convex hull. The convex hull V_k is called the *outer boundary*. We make UAV fleet move along the outer boundary and cover the maximum possible range under the communication connectivity constraint (keep the formation to be evenly spaced on a line with a length $w_{max} = (m-1) \times w$). This would generate a ring-like area, denoted by R_k, as illustrated in Figure 7(a), which is exactly the area of the kth round of scanning. Let $V'_k = \{v'_{k,1}, v'_{k,2}, \ldots, v'_{k,h_k}\}$ be the points constituting the inner boundary of the ring R_k. Obviously, the inner bounder is parallel to and lies inside the outer boundary. Let P_k be the set of target points inside the area R_k (e.g., for $k = 1$, $P_1 = P \cap R_1$). The specific scanning scheme for target points P_k will be described in detail later.

After scanning the set P_k, the remaining target points $P \setminus \bigcup_{i=1}^{k} P_i$ all locate inside the inner boundary of R_k. We continue the scanning by treating $P \setminus \bigcup_{i=1}^{k} P_i$ as new given target points and computing the convex hull of these points again to determine the scanning area R_{k+1} in the next round of scanning. We repeat this operation until all the target points are scanned. Let the total



(a) The partition of the area where $K = 3$



(b) The transfer of UAV fleet from one scanning area to the next scanning area when $K = 3$

Fig. 7. An example of area partition and transfer of UAV fleet.

number of rounds be K. Then, we have $P = \bigcup_{k=1}^{K} P_k$. Figure 7(a) shows a schematic diagram when $K = 3$. The detailed description of the overall scanning scheme is shown in Algorithm 1.

3.2.2 Transfer Scheme of Two Adjacent Rounds. Now, we introduce the transfer scheme on how the UAV fleet will travel from a scanned area R_k to the next area R_{k+1} to be scanned in the next round, as shown in Figure 7(b). In our scheme, the UAV fleet will travel from R_k to R_{k+1} along a line as follows: During the process that all the drones move from the outermost scanning area R_1 to the innermost one R_K, the shortest distance between the outer boundary of R_1 and the outer boundary of R_K is treated as the shortest *transfer distance*. It is not difficult to see that this value is the shortest distance from the points in V_K to the outer boundary of R_1. Let the intersection points between the line with the shortest transfer distance and the outer boundary of $R_k, k = 1, 2, \ldots, K$ be $C = \{c_1, c_2, \ldots, c_K\}$. Figure 7(b) illustrates a segment of c_1c_K that achieves the shortest transfer distance, which is denoted as L_{trans}. In the kth round of scanning, the fleet starts the flight from

c_k. After scanning the area R_k, the fleet returns to c_k. Then, it moves along $c_k c_{k+1}$ to c_{k+1} and starts the next round of scanning.

Remark 1. Although the fleet needs to adjust the formation when all the drones enter the next scanning area, considering the length of the flight along the outside of a scanning area is usually much longer than that along the inside, the moving distance caused by this adjustment can be ignored, which implies that under our planning the longest route of the UAV fleet is determined by the drone that moves along the outer boundary of each area $R_i, i = 1, 2, \ldots, K$.

ALGORITHM 1: Polygon-guided Scanning Algorithm (a.k.a. PSA)

input: the set of target points P, the set of drones U, maximum communication distance w
output: the flight planning and the corresponding flight cost of the UAV fleet L_{fleet}

Let the largest scanning width be $w_{max} \leftarrow (m-1) \times w$, let $k \leftarrow 1$, let $L_{fleet} \leftarrow 0$;
while *P is not empty* **do**
  Compute the set of points V_k that constitute the convex hull of P (e.g., by adopting the method of Graham scanning);
  Compute the set of points $V_k^{'}$ that constitute the inner boundary of the area of the current round of scanning, based on V_k and w_{max};
  Get the sub-area R_k and target points P_k in round k based on V_k, $V_k^{'}$, and P;
  Call Algorithm to scan sub-area R_k and get the corresponding flight cost L_k;
  $L_{fleet} \leftarrow L_{fleet} + L_k$;
  $P \leftarrow P \setminus P_k$;
  $k \leftarrow k + 1$;
end
Compute the transfer distance L_{trans};
$L_{fleet} = L_{fleet} + L_{trans}$;
return L_{fleet};

3.2.3 The Planning for a Single Round of Scanning. Now, we introduce the detailed planning for scanning target points in a single round, the specific ring-like area R_k.

As mentioned before, during the process of scanning target points, we will keep all the drones on a line. We call these lines formed by the UAV fleet the *scanning lines*. We will make the scanning lines *perpendicular* to the parallel lines of the outer and inner boundary of area R_k at any time. The scanning process then can be viewed as the traversal of the scanning lines. Figure 8(a) illustrates the scanning lines (dashed lines) during the scanning. In general, UAVs in the fleet formation will travel in parallel along the outer boundary, except the case that some target points are covered by the scanning line of the fleet during the flight. In that case, the drones should adjust and travel along the scanning line to cover the target points that fall on that line. Besides, in our planning, the distance between any two adjacent drones remains unchanged all the time, except for the case when they meet a vertex of the scanning area where the drones need to adjust their directions.

We first determine the flight cost caused by traversing between the scanning lines. Assume that the perimeter of the outer boundary (convex polygon) in area R_k is L_k. As shown in Figure 8(a), during the traversing between the scanning lines in the area R_k, the outermost drone u_1 travels along the convex polygon and contributes the maximum flight distance. Thus, the flight cost of the fleet caused by traversing between the scanning lines is L_k.

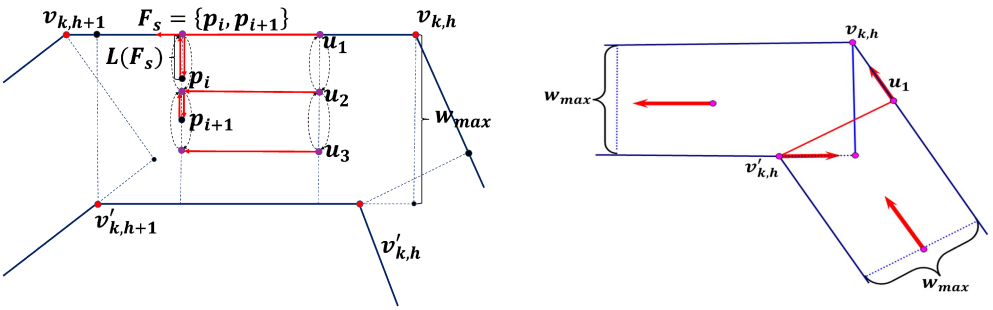


Fig. 8. (a) The scheme for single round of scanning; (b) Adjustment in the corner region.

Then, we determine the flight cost to adjust the fleet and travel along the scanning line for visiting target points lying on a single scanning line. Since target points must be passed by scanning line, we can divide the set of target points P_k in R_k into a number of subsets. Denoted by F_s the sth subset that contains target points lying on the same scanning line. Figure 8(a) illustrates a set F_s with two target points that lie on the same scanning line. Assume that there are S such subsets. Then, $P_k = \bigcup_{s=1}^{S} F_s$ where $F_i \cap F_j = \varnothing (i \neq j)$.

ALGORITHM 2: Polygon-guided Scanning Single-Turn Algorithm (PSTA)

input: the set of drones U, the scanning area R_k, the set of target points P_k, maximum communication distance w

output: the flight planning and corresponding flight cost of the UAV fleet L_k for the kth round of scanning

```
Compute the subset of points F_s(s = 1, 2, ..., S) where each subset includes all the target
points lying on the same perpendicular line of the two boundaries;
Compute the length L(b_k) of the outer boundary of R_k;
Let L_k ← L(b_k);
for s = 1 to S do
    Let the UAV fleet move to the scanning line that passes through F_s;
    Let the fleet fly towards the inner boundary along the scanning line, until all the target
    points in F_s are scanned, and then return back;
    Let the flight cost caused by travelling along the scanning line be L(F_s);
    L_k ← L_k + 2 · L(F_s);
end
return L_k;
```

To obey the connectivity constraints of all the drones, the whole fleet will make a minor adjustment and fly towards the inner boundary of R_k simultaneously to reach the target points covered by the current scanning line. If there are multiple target points on the scanning line, the flight cost will be determined by the minimum moving distance to scan all the target points, which can be easily measured. Let $L(F_s)$ be such a cost generated on a scanning line F_s. When all the target points on the current scanning line have been scanned, the fleet will return to their starting points on this scanning line and move on to the next scanning line, as demonstrated by the red lines in Figure 8(a).

Therefore, the flight cost of the adjustment would be $2 \cdot L(F_s)$. We use $L_{adjust} = \sum_{F_s \subseteq P} 2L(F_s)$ to denote the total travel distance generated by the minor adjustments during the whole flight.

Note that after finishing scanning along an edge of the outer boundary, we need to further adjust the fleet formation and make it scan in parallel along the next edge of the outer boundary, as illustrated in Figure 8(b). Such an adjustment would not increase the flight cost of the fleet, since the outermost UAV still travels with the longest distance along the outer boundary.

Algorithm 2 presents the details of the scanning process for a specific sub-area. Line 1 constructs the subsets F_s. Line 2 computes the total flight cost when the fleet traverses between scanning lines. Lines 4–9 guide the fleet to scan the target points F_s falling on a scanning line and computes the corresponding flight cost.

3.2.4 Theoretical Analysis. We further theoretically analyze the worst-case performance of the algorithm. Intuitively, according to the partition scheme of PSA, the convex polygon generated in each round of PSA is the minimum convex set containing the remaining non-visited target points, thus the optimal solution needs to travel with a distance equaling the perimeter of the convex polygon in each round. This allows us to bound the difference of UAV fleet travel distance between PSA and the optimal solution. Let $L(PSA)$ and $L(OPT)$ be the UAV fleet travel distance incurred by algorithm PSA and the optimal solution, respectively. Note that during the travel along convex polygons, or more exactly traversing between scanning lines, the UAV fleet needs to adjust and move along the scanning line to cover the target points on the scanning line. In the following theorem, we show that the extra travel distance generated by PSA compared to the optimal solution is upper bounded by L_{adjust}, which is usually a short moving distance caused by minor adjustment when traversing along the convex polygons.

Theorem 1. *The gap of the flight distance between PSA and the optimum solution satisfies* $L(PSA) - L(OPT) \leq L_{adjust}$.

Proof. There are three constraints in the target-point visiting problem. The coverage constraint requires us to scan all the target points in target points set P, which indicates that we must scan the vertices on the convex hull of target points. However, connectivity constraint requires the connectivity of drones in real time, hence the largest scanning width of drones is w_{max}. According to the partition scheme of PSA, each sub-area R_k is the maximum region that the UAV fleet can scan along the convex polygon. Therefore, in the optimum solution, the distance the UAV fleet needs to travel within each sub-area R_k is at least L_k. Moreover, as L_{trans} is the shortest distance from the outermost boundary to the innermost boundary, the optimal solution should also incur another travel distance L_{trans} to move the fleet from the outermost area to the innermost area. Hence, we can get the lower bound of the optimum solution that $L(OPT) \geq L_{trans} + \sum_{k=1}^{K} L_k$.

Note that the travel distance of the UAV fleet generated by algorithm PSA contains three parts: the distance L_k incurred by travelling along the convex polygon (which is the distance the outermost UAV travels with), the distance L_{trans} incurred by moving along the transfer line, and the distance $2L(F_s)$ incurred by covering the target points on a scanning line F_s. Accordingly, the total travel distance of PSA is $\sum_{k=1}^{K} L_k + L_{trans} + L_{adjust}$ where $L_{adjust} = \sum_{F_s \subseteq P} 2L(F_s)$. Therefore, the extra travel distance generated by PSA compared to the optimal solution is at most L_{adjust}. □

4 EXPERIMENTAL RESULTS AND ANALYSIS

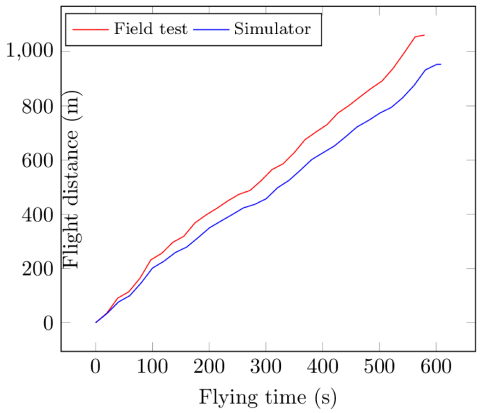
In this section, we conduct field tests and simulations to demonstrate the effectiveness of our CoUAS platform in simulating UAVs cooperation and real-world flights. We will perform the tasks in both the simulator environment and the field tests, demonstrating that our simulator can be used as a reliable platform for testing.

Fig. 9. Trajectories of three UAVs in field test (left) and simulation environment (right) when completing a task with 30 points.

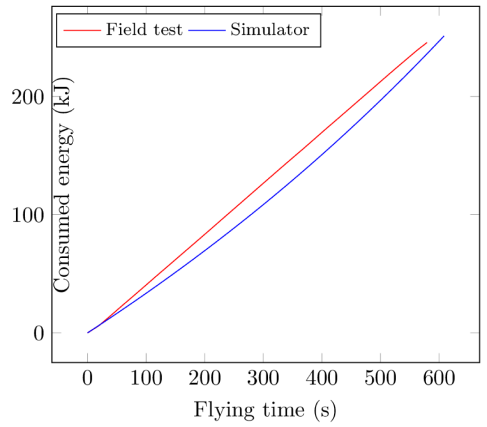
4.0.5 Comparison between the Trajectories Generated in the Simulator and Field Test for a Single Task. We first set a target-point visiting task that consists of 30 target points and three UAVs as a fleet. The length and width of the field are set to be 120 m. All target points are randomly generated in this area. By default, the tasks are performed at the speed of 4 m/s. With the provided synchronization mechanism, when a UAV moves to a point at which it needs to change its direction, it would check the distance to others and wait for others if necessary to preserve global connectivity. We first run the task in our simulator to generate trajectories and predict the energy consumption based on the prediction model trained for UAVs. Then, we use the following two types of UAVs to conduct field tests: copters with frame QUAD or HEXA controlled by Pixhawk and copters with frame QUAD controlled by APM. The flight distance is calculated according to the real-time coordinates from GPS. The time of flight is calculated according to the system clock of Raspberry Pi 3b on UAV. The energy consumption is measured by computing the real-time voltage and current of battery read from the power module.

Figure 9 shows the trajectories generated in our CoUAS simulator (left one) and the field test (right one). From the figure, we can see that the trajectories generated in the simulator are similar to those in the field tests. The trajectory between two inflection points is more irregular in field test, which is caused by the unstable environment conditions in reality. Despite this, after examining log data from the connectivity management and synchronization modules of CoUAS, we find that the fleet manages to maintain the connectivity and visits all target points while satisfying the battery capacity.

4.0.6 Comparison of Flying Distance and Energy Consumption between Simulation and Field Test for a Single Task. For the same task, we recorded the flying distance and energy consumption during the whole flight in both simulation and the field test. Figure 10 shows the flying distance and energy consumption on one of the three UAVs, which is a copter with frame HEXA controlled by Pixhawk. We can see from the figure that during the whole flight, at any given time, the flying distance and consumed energy in simulation and field test are consistently close.

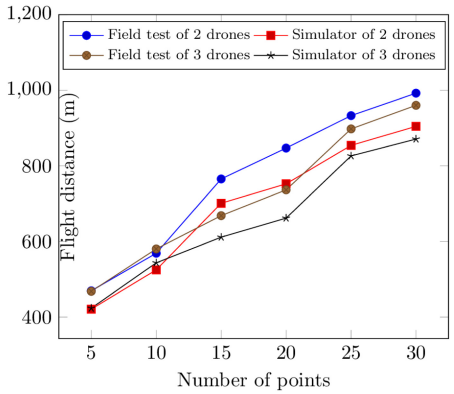


(a) Flight distance versus flying time.

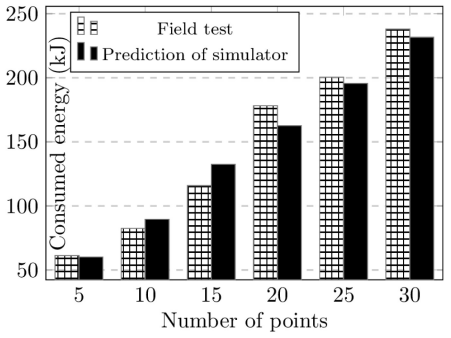


(b) Energy consumption versus flying time.

Fig. 10. Flying distance and energy consumption in simulation and field test for a single task.



(a) Flight distance for different number of points.



(b) Energy consumption for different number of points.

Fig. 11. The impact of the number of target points on the flight distance and flying time.

4.0.7 Comparison of Flying Distance, Flying Time, Energy Consumption in Simulation and Field Test over Multiple Tasks. We generate several different tasks by changing the number of target points from 5 to 30. For each task, we generate a path plan and conduct simulation and field test accordingly. The flying distance and energy consumption are recorded in each task.

As we can see from Figure 11, the flying distance and energy consumption in the simulation is very close to that in the field tests for multiple tasks. In terms of the energy consumption, the gap between the field test and simulator is about 5.74% on average. These together validate that the simulator can reliably reflect the real flight.

5 RELATED WORK

This article mainly focuses on developing an easy-to-use unified framework to facilitate the design, deployment, and testing of multi-UAV applications. In this section, we mainly review relevant multiple UAV cooperation testbeds and the existing work in path planning.

Over the years, some research efforts have been made on designing testbeds [2, 5, 10, 19] for multiple UAVs. Such testbeds for UAVs are mainly focusing on the control of fixed-wing UAVs, which

cannot be easily extended to that of multi-rotor UAVs. In recent years, multi-rotor consumer UAVs have become more and more popular. Multi-rotor UAV testbeds emerge quite recently. For example, the Group Autonomy for Mobile Systems (GAMS) project [8] aims at providing a distributed operating environment for accurate control of one or more UAVs. The OpenUAV project [26] provides a cloud-enabled testbed including standardized UAV hardware and an end-to-end simulation stack. Itkin et al. [11] propose a cloud-based web application that provides real-time flight monitoring and management for quad-rotor UAVs to detect and prevent potential collisions. Gazebo [13] and AirSim [23] aim at providing 3D simulation environment that contains realistic scenarios to support research such as aerial 3D reconstruction and object tracking. However, few works provide a generic platform that aims at simplifying the design of multi-UAV cooperative applications and their development. What is more, few works have exploited the connection characteristics among multiple UAVs.

In the past few years, a number of research works [1, 12, 17, 24] have studied the path planning problem with the coverage constraint. However, they have not considered the connectivity requirement of multiple UAVs during the flight. Only a few works have addressed the same connectivity requirement of UAV fleet during flight as considered in this article. Yanmaz et al.[27] and Schleich et al. [22] develop and design distributed heuristic path planning algorithms to cover a given area and maintain the connectivity of UAVs. However, they fail to guarantee either the full coverage of the area or hard-constraint of connectivity. Most recently, Tateo et al. [25] and Charrier et al. [3] study the problem of multi-agent connected path finding to visit a given set of points for cooperative connected agents. However, the problem is established on a finite graph with a discretized environment, while real-world environment is continuous and, moreover, the online synchronization as well as adjustment among UAVs during the flight is not considered.

In summary, to the best knowledge of the authors, there is a lack of viable solutions that can provide a generic framework/testbed to support cooperative UAV fleet control/monitoring with connectivity and facilitate the deployment of multi-UAV applications.

6 CONCLUSION

In this article, we develop an open-source system named CoUAS that enables the cooperation of multiple UAVs in a fleet. The proposed system provides generic interface that hides the hardware difference of UAVs to facilitate the cooperative UAV development and also offers a series of cooperation services such as synchronization and connectivity management, energy simulation, and cooperative path planning to enable upper-layer user application designs. We further detail the main algorithm of path planning service provided in our CoUAS platform and evaluate the system performance with simulations and field tests to validate that the proposed system is viable. A demo and source code are published in *https://github.com/whxru/CoUAS* for open access.

REFERENCES

[1] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and João Valente. 2011. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *J. Field Robot.* 28, 5 (2011), 667–689.

[2] Jeremy W. Baxter, Graham S. Horn, and Daniel P. Leivers. 2007. Fly-by-agent: Controlling a pool of UAVs via a multi-agent system. In *Proceedings of the International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer, 219–230.

[3] Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. 2019. Reachability and coverage planning for connected agents. In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1874–1876.

[4] Thomas Clausen and Philippe Jacquet. 2003. Optimized Link State Routing protocol (OLSR). RFC Editor. http://tools.ietf.org/html/rfc3626.

[5] Michael A. Day, Michael R. Clement, John D. Russo, Duane Davis, and Timothy H. Chung. 2015. Multi-UAV software systems and simulation architecture. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS'15)*. IEEE, 426–435.
[6] DJI. 2019. DJI FlightHub. Retrieved from: https://www.dji.com/flighthub.
[7] Dronecode. 2019. QGroundControl. Retrieved from: http://qgroundcontrol.com/.
[8] James Edmondson. 2019. Group Autonomy for Mobile Systems. Retrieved from: https://github.com/jredmondson/gams.
[9] Lav Gupta, Raj Jain, and Gabor Vaszkun. 2015. Survey of important issues in UAV communication networks. *IEEE Commun. Surv. Tutor.* 18, 2 (2015), 1123–1152.
[10] Jonathan How, Yoshiaki Kuwata, and Ellis King. 2004. Flight demonstrations of cooperative control for UAV teams. In *Proceedings of the AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*. 6490.
[11] Mason Itkin, Mihui Kim, and Younghee Park. 2016. Development of cloud-based UAV monitoring and management system. *Sensors* 16, 11 (2016), 1913.
[12] Asif Khan, Evsen Yanmaz, and Bernhard Rinner. 2015. Information exchange and decision making in micro aerial vehicle networks for cooperative search. *IEEE Trans. Contr. Netw. Syst.* 2, 4 (2015), 335–347.
[13] Nathan Koenig and Andrew Howard. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'04)(IEEE Cat. No. 04CH37566)*, Vol. 3. IEEE, 2149–2154.
[14] Liangzhi Li, Kaoru Ota, and Mianxiong Dong. 2018. Humanlike driving: Empirical decision-making system for autonomous vehicles. *IEEE Trans. Vehic. Technol.* 67, 8 (2018), 6814–6823.
[15] Ting Li, Kaoru Ota, Tian Wang, Xiong Li, Zhiping Cai, and Anfeng Liu. 2019. Optimizing the coverage via the UAVs with lower costs for information-centric Internet of Things. *IEEE Access* 7 (2019), 15292–15309.
[16] Kejie Lu, Junfei Xie, Yan Wan, and Shengli Fu. 2019. Toward UAV-based airborne computing. *IEEE Wirel. Commun.* 26, 6 (2019), 172–179.
[17] Ivan Maza and Anibal Ollero. 2007. Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms. In *Distributed Autonomous Robotic Systems 6*. Springer, 221–230.
[18] Xiangyun Meng, Wei Wang, and Ben Leong. 2015. Skystitch: A cooperative multi-UAV-based real-time video surveillance system with stitching. In *Proceedings of the 23rd ACM International Conference on Multimedia*. 261–270.
[19] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. 2010. The grasp multiple micro-UAV testbed. *IEEE Robot. Autom. Mag.* 17, 3 (2010), 56–65.
[20] Michael Oborne. 2019. Mission Planner. Retrieved from: http://ardupilot.org/planner/.
[21] Alena Otto, Niels Agatz, James Campbell, Bruce Golden, and Erwin Pesch. 2018. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks* 72, 4 (2018), 411–458.
[22] Julien Schleich, Athithyaa Panchapakesan, Grégoire Danoy, and Pascal Bouvry. 2013. UAV fleet area coverage with network connectivity constraint. In *Proceedings of the 11th ACM International Symposium on Mobility Management and Wireless Access*. 131–138.
[23] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*. Springer, 621–635.
[24] P. B. Sujit and Randy Beard. 2007. Cooperative path planning for multiple UAVs exploring an unknown region. In *Proceedings of the American Control Conference*. IEEE, 347–352.
[25] Davide Tateo, Jacopo Banfi, Alessandro Riva, Francesco Amigoni, and Andrea Bonarini. 2018. Multiagent connected path planning: PSPACE-completeness and how to deal with it. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 4735–4742.
[26] Open UAV Team. 2018. The OpenUAV Project. Retrieved from: https://openuav.us/.
[27] Evşen Yanmaz. 2012. Connectivity versus area coverage in unmanned aerial vehicle networks. In *Proceedings of the IEEE International Conference on Communications (ICC'12)*. IEEE, 719–723.

Received October 2019; revised January 2020; accepted March 2020