Contingency Formation Planning for Interactive Drone Light Shows

Tsz-Chiu Au[1]

***Abstract*— One of the most appealing applications of drone swarms is drone light shows, in which a group of drones displays an animation by showing a sequence of light patterns in the sky. In this paper, we consider using drone swarms as video game platforms and utilize planning techniques to display pixels in animations correctly while providing a fast response to user inputs. We devise a new sampling algorithm to solve a contingency formation planning problem, which aims to find a contingency formation plan such that drones can always move to the correct positions to display every possible future frame regardless of the user inputs in the future. The algorithm provides interactivity by preemptively relocating hidden drones, which move in stealth mode to the locations of all possible future frames. Our experiments show that the size of the frame buffer and the ratio between the number of drones and the number of pixels can greatly affect the performance of our system.**

I. INTRODUCTION

Drone light shows have emerged as a new kind of entertainment for celebrations and festivals worldwide. Given the success of drone light shows, we go one step further to consider drone swarms as video game platforms. Some existing drone light shows have already demonstrated the potential of using drone swarms for *drone-swarm-based video games*. Fig. 1 shows a drone light show for promoting the new Super Mario Bros. movie. Fig. 2 is a scene in an Ultraman drone light show at Kobe Meriken Park in Japan. While both drone light shows suggest the possibility of using drone swarms for video games, they also highlight some technical challenges in such platforms. The most notable challenge is the *low frame rate* since drones take time to physically move to the correct positions to display a scene, causing difficulties in maintaining an illusion of animations for video games. Moreover, these drone light shows are not *interactive*—no human player could control the characters.

This paper attempts to address these challenges by considering the formation planning problem in drone-swarm-based video games. We describe a framework that shows animations in *real time* by planning the motion of the drones ahead of time. Here, the term "real-time" refers to the guarantee that pixels can be displayed correctly subject to time constraints (e.g., frames can be displayed at specific times). To deal with interactivity, we propose generating *contingency* formation plans such that no matter what the user inputs are, the drone swarm can still display the correct scenes in response to the user inputs. Ultimately, we define the *pixel contingency formation planning problem* that captures the essence of these challenges and devise a new planning algorithm to solve the problem.

[1]Department of Computer Science, Texas State University, USA. chiu.au@txstate.edu

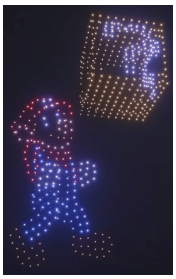

Fig. 1: Mario jumps to hit a bonus brick.[3]

Fig. 2: Ultraman throws a laser knife at a monster.[4]

In summary, the contributions of this paper are:

- We define the pixel contingency formation planning problem for interactive drone light shows with real-time performance guarantees in response to user inputs.
- We devise an algorithm called SPICOMP, a sampling-based algorithm for pixel contingency formation planning. SPICOMP is the first formation planning algorithm that deals with contingency in drone light shows.
- We describe the architecture of our interactive drone light show system, which is a pipeline connecting user's commands to formation plans that control a drone swarm to display light patterns.
- Our experiments show that the frame buffer's size and the number of drones relative to the number of pixels can greatly affect the performance of our system.

This paper is organized as follows. After presenting the related work in Sec. II and the key features of drone-swarm-based video games in Sec. III, we define the planning problem in Sec. IV, describe the frame controller in Sec. V, and present SPICOMP in Sec VI. Finally, we present the experimental results in Sec. VII and conclude in Sec. VIII.

II. RELATED WORK

Some research and commercial endeavors have already capitalized on utilizing drone swarms for entertainment. Kim and Landay introduced Aeroquake, an augmented dance system [1]. Kljun et al. introduced StreetGamesz, a gaming platform in which drones carry projectors to project game-playing elements [2]. Kim et al. presented a survey of drone use for entertainment and augmented/virtual reality [3]. In recent years, research on drone light shows has gained traction. Du et al. presented a framework for designing choreographies comprised of swarm motion primitives [4]. You et al. discussed the use of rotating LED display technology in

[3]https://www.youtube.com/watch?v=shyKVbrbtnA
[4]https://www.youtube.com/watch?v=a-fOhLjMI0U

drone light shows [5]. Mao et al. presented a drone swarm light show design platform for K-12 children [6]. Weng et al. described a multi-view approach for drone light shows [7].

Drone light shows are multirobot systems with a special focus on formation planning. Yu and LaValle proposed planning optimal paths for multiple robots using integer linear programming models that optimize for makespan [8], [9]. Yang et al. wrote a survey on the UAV formation trajectory planning algorithms [10]. Our pixel formation planning problem is similar to the *formation reconfiguration* problem, which aims to find a formation plan to transform an initial formation configuration into a final configuration [11]. By contrast, our work focuses on pixel formation instead of drone formation. A popular approach for formation reconfiguration is the artificial potential field (APF) method [12], [13]. Some works tried to overcome the oscillation and gridlock problem in the APF methods [14], [15]. Nar and Kotecha described a constrained Hungarian method called CHungSDA for drone assignment to waypoints in drone light shows [16]. Huang et al. presented a Hungarian algorithm for graph-based dynamic task assignment problems in UAV light shows [17]. However, these works did not acknowledge the special roles played by hidden drones in drone light shows.

III. DRONE-SWARM-BASED VIDEO GAMES

Our drone-swarm-based video game platforms adopt the same hardware for drone light shows. Each drone is equipped with an LED light bulb that can shine a wide range of color light in all directions. When a drone turns on its LED light, it shows a *pixel*. A *frame* is a collection of pixels that appear at the same time. There are software tools (e.g., Skybrush) for designing drone light shows, but they do not automatically decide when drones should go dark and become *hidden*. Hidden drones can move in stealth mode and suddenly appear at remote locations in a scene. Interactivity can be achieved by preemptively relocating hidden drones to the locations of *all* possible future frames and then turning on the LED lights of the frame chosen by the user.

There are two ways to move a pixel between two consecutive frames: *pixel trajectory tracking* and *pixel hopping*. In pixel trajectory tracking, a drone moves along a trajectory while the LED light is on. In pixel hopping, a drone switches off its LED light, and an adjacent drone turns on its LED light as if the pixel hops from one drone to another. Pixel trajectory tracking can provide a smooth animation of the characters, while pixel hopping is used to show fast-moving objects. However, pixel hopping creates some discontinuity in the apparent trajectories of the hopping pixels.

IV. PIXEL FORMATION PLANNING PROBLEMS

Let $\mathcal{V}$ be a set of drones $\{\upsilon_1, \upsilon_2, \ldots, \upsilon_n\}$. A pose ρ for a drone $\upsilon \in \mathcal{V}$ is the position and the orientation of υ in the world frame, i.e., ρ is $(x, y, z, \theta^x, \theta^y, \theta^z)$, where the position (x, y, z) is the coordinate of the center of υ and θ^x, θ^y, and θ^z are the roll, pitch, and yaw rotations of υ, respectively. A *state* s of a drone extends ρ to include the internal states of the drone relevant to motion planning. If we are using velocity-based controllers to control a drone, s is $(x, y, z, \theta^x, \theta^y, \theta^z, \mathrm{v}^x, \mathrm{v}^y, \mathrm{v}^z)$ where $(\mathrm{v}^x, \mathrm{v}^y, \mathrm{v}^z)$ is the velocity of the drone. Given a state s, we denote the pose by $\rho[s] = (x, y, z, \theta^x, \theta^y, \theta^z)$ and the position by $\mathsf{pos}[s] = (x, y, z)$. The *distance* between two drones υ_1 and υ_2 with states s_1 and s_2, respectively, is $\mathsf{dist}(\upsilon_1, \upsilon_2) = \mathsf{dist}(\mathsf{pos}[s_1], \mathsf{pos}[s_2])$, which is the Euclidean distance between (x_1, y_1, z_1) and (x_2, y_2, z_2). A *motion plan* π for a drone υ is a sequence of control commands for υ. Each control command could be a waypoint on an intended trajectory or a target velocity along an intended trajectory. When υ executes π starting with an initial state s^1, υ acts according to π and moves along a *state trajectory* ζ, which is a sequence of states of υ. We denote the state of υ at time t on ζ by $\zeta[t]$ such that $\zeta[0] = s^1$. Let us define $|\pi|$ to be the duration of the execution of π, which is equal to $|\zeta|$, the duration of ζ.

A *formation* S for $\mathcal{V}$ is $\{s_i\}_{1 \le i \le n}$, where s_i is a state of $\upsilon_i \in \mathcal{V}$. A *formation plan* for $\mathcal{V}$ is $\Pi = \{\pi_i\}_{1 \le i \le n}$, where π_i is a motion plan for $\upsilon_i \in \mathcal{V}$ and $|\pi_{i_1}| = |\pi_{i_2}|$ for any $i_1, i_2 \in [1, n]$. We denote the plan π_i for drone υ_i in Π by $\Pi[\upsilon_i]$. We denote the duration of the execution of Π by $|\Pi|$, which is equal to $|\pi|$ for any $\pi \in \Pi$. Given a formation plan $\Pi = \{\pi_i\}_{1 \le i \le n}$ for $\mathcal{V}$ and an *initial* formation $S^1 = \{s_i^1\}_{1 \le i \le n}$ where s_i^1 is the initial state of $\upsilon_i \in \mathcal{V}$, the *formation trajectory* Ξ of Π starting with S^1 is $\{\zeta_i\}_{1 \le i \le n}$, where ζ_i is the state trajectory of υ_i according to π_i. We denote the formation of $\mathcal{V}$ at time t by $\Xi[t] = \{\zeta_i[t]\}_{1 \le i \le n}$, where $\zeta_i[t]$ is the state of υ_i at time t on $\zeta_i \in \Xi$. We denote the duration of Ξ by $|\Xi| = |\Pi|$. A formation $S = \{s_i\}_{1 \le i \le n}$ is *safe* if and only if $\mathsf{dist}(\mathsf{pos}[s_{i_1}], \mathsf{pos}[s_{i_2}]) \ge D_{\mathsf{safe}}$ where D_{safe} is the *safe distance* between two drones for any $i_1 \ne i_2$. A formation trajectory Ξ is *safe* if and only if $\Xi[t]$ is safe for all $0 \le t \le |\Xi|$. A formation plan Π is safe for an initial formation S^1 if and only if Ξ of Π starting with S^1 is safe.

A frame is a set of pixels displayed simultaneously. A *configuration* of a pixel p is a 4-tuple (x, y, z, c), where (x, y, z) is the position of p with respect to the 3D world frame and $c \in \mathbb{C}$ is a color, where $\mathbb{C}$ is the set of all possible colors that the LED lights on the drones can display. A frame f is a set of pixels $\{p_j\}_{1 \le j \le m}$, where the configuration of p_j is (x_j, y_j, z_j, c_j) and m is the number of pixels in f, where $0 \le m \le n$. We denote the position and the color of p_j by $\mathsf{pos}[p_j] = (x_j, y_j, z_j)$ and $\mathsf{color}[p_j] = c_j$, respectively. The position of every pixel in a frame must be unique in the frame, and $\mathsf{dist}(\mathsf{pos}[p_{j_1}], \mathsf{pos}[p_{j_2}]) \ge D_{\mathsf{safe}}$ for any $j_1 \ne j_2$.

Definition 1: A frame $f = \{p_j\}_{1 \le j \le m}$ is *renderable* at time t by a set of drones $\mathcal{V} = \{\upsilon_i\}_{1 \le i \le n}$ with formation $S = \{s_i\}_{1 \le i \le n}$ at time t where s_i is the state of υ_i if there *exists* an assignment $g : f \to \mathcal{V}$ that assigns every pixel $p_j \in f$ to a drone $g(p_j)$ such that 1) $g(p_{j_1}) \ne g(p_{j_2})$ for any $j_1 \ne j_2$, and 2) the position $\mathsf{pos}[s]$ of $g(p_j)$ at time t is $\mathsf{pos}[p_j]$, where s is the state of $g(p_j)$ at time t.

If f is renderable at time t by $\mathcal{V}$ with S, we say f is *rendered* by S with g at time t. Since we can deduce g from S, we simply say f is *rendered* by S at time t.

A. The Pixel Formation Planning Problem

We consider transforming $f_1 = \{p_j^1\}_{1\le j\le m_1}$ at time t_1 into $f_2 = \{p_j^2\}_{1\le j\le m_2}$ at time t_2, where $t_2 > t_1$. Let $\mathcal{V} = \{v_i\}_{1\le i\le n}$ be a set of drones that takes part in the transformation, where $m_1 \le n$ and $m_2 \le n$. Let $S_1 = \{s_i^1\}_{1\le i\le n}$ be the formation of $\mathcal{V}$ at time t_1, where s_k^1 is the state of $v_k \in \mathcal{V}$ at time t_1. Suppose f_1 is rendered by S_1 with an assignment g_1 at time t_1. Some pixels in f_1 require pixel trajectory tracking during transformation, meaning that the drones who renders these pixels need to move along given trajectories exactly. These trajectories are specified as follows. A *single pixel trajectory specification* $\omega = (p_{j_1}^1, p_{j_2}^2, \zeta_{j_1,j_2}, \pi_{j_1,j_2})$, where $p_{j_1}^1 \in f_1$, $p_{j_2}^2 \in f_2$, ζ_{j_1,j_2} is a state trajectory where $\mathsf{pos}[\zeta_{j_1,j_2}[0]] = \mathsf{pos}[p_{j_1}^1]$, $\mathsf{pos}[\zeta_{j_1,j_2}[t_2 - t_1]] = \mathsf{pos}[p_{j_2}^2]$, and π_{j_1,j_2} is a motion plan that yields ζ_{j_1,j_2} when a drone with state $\zeta_{j_1,j_2}[0]$ executes π_{j_1,j_2}. ω specifies how a pixel should move in pixel trajectory tracking. We denote $p_{j_1}^1$, $p_{j_2}^2$, ζ_{j_1,j_2}, and π_{j_1,j_2} in ω by $p^{\mathsf{start}}[\omega]$, $p^{\mathsf{end}}[\omega]$, $\zeta[\omega]$, and $\pi[\omega]$, respectively. A *pixel trajectory tracking specification* for transforming f_1 into f_2 is a set $\Omega_{1,2}$ of single pixel trajectory specifications, where $p^{\mathsf{start}}[\omega_1] \neq p^{\mathsf{start}}[\omega_2]$ and $p^{\mathsf{end}}[\omega_1] \neq p^{\mathsf{end}}[\omega_2]$ for any different $\omega_1, \omega_2 \in \Omega_{1,2}$. Let $f_1^{\mathsf{track}} = \{p^{\mathsf{start}}[\omega]\}_{\omega\in\Omega_{1,2}}$ and $f_2^{\mathsf{track}} = \{p^{\mathsf{end}}[\omega]\}_{\omega\in\Omega_{1,2}}$ be the set of pixels involved in pixel trajectory tracking in $\Omega_{1,2}$. Note that $\Omega_{1,2}$ could be empty, meaning no pixel trajectory tracking is needed.

Definition 2: A *frame transformation problem* is a tuple $(\mathcal{V}, f_1, f_2, S_1, \Omega_{1,2})$, where 1) $\mathcal{V} = \{v_i\}_{1\le i\le n}$ is a set of drones, 2) $f_1 = \{p_j^1\}_{1\le j\le m_1}$ is a frame at time t_1, where $m_1 \le n$, 3) $f_2 = \{p_j^2\}_{1\le j\le m_2}$ is a frame at time t_2, where $m_2 \le n$ and $t_1 < t_2$, 4) $S_1 = \{s_i^1\}_{1\le i\le n}$ is the formation of $\mathcal{V}$ at time t_1 s.t. f_1 is rendered by S_1 with an assignment g_1 at time t_1, 5) $\Omega_{1,2}$ is a pixel trajectory tracking specification for transforming f_1 into f_2. The solution of a frame transformation problem is a formation plan Π such that 1) Π is safe for S_1; 2) f_2 is rendered by S_2 at time t_2, where $S_2 = \Xi[t_2 - t_1]$ is the formation of $\mathcal{V}$ at time t_2 and Ξ is the formation trajectory of Π starting with S^1; and 3) $\Pi[g_1(p^{\mathsf{start}}[\omega])] = \pi[\omega]$ for all $\omega \in \Omega_{1,2}$.

Definition 2 does not specify how the drones change their colors during transformation. Since we can remotely control the colors of the drones, we can update the colors anytime.

We divide the timeline evenly into time steps. Let $t_k = k \times T_{\mathsf{step}}$ be the start time of the time step k for $k \ge 0$ where T_{step} is the duration of a time step. Suppose we are given a sequence of frames $\mathbf{F} = \langle f_0, f_1, \ldots, f_T\rangle$, where f_k is the frame at time t_k for $0 \le k \le T$. We are also given $\mathbf{\Omega} = \langle \Omega_{0,1}, \Omega_{1,2}, \ldots, \Omega_{T-1,T}\rangle$, where $\Omega_{k,k+1}$ is a pixel trajectory tracking specification for transforming f_k into f_{k+1} for $0 \le k < T$. To simplify our notation, we assume the pixel trajectory tracking specification $\Omega_{k,k+1}$ of any two adjacent frames f_k and f_{k+1} in any frame sequence is implicitly given. We extend Definition 2 as follows:

Definition 3: A *pixel formation planning problem* is a triple $(\mathcal{V}, \mathbf{F}, S_0)$, where 1) $\mathcal{V}$ is a set of drones, 2) $\mathbf{F}$ is a sequence of frames $\langle f_k\rangle_{0\le k\le T}$, and 3) S_0 is the initial formation of $\mathcal{V}$ at time t_0 such that f_0 is rendered by S_0 at t_0. The solution of a pixel formation planning problem is a sequence of formation plans $\mathbf{\Pi} = \langle \Pi_k\rangle_{0\le k<T}$ such that Π_k is a solution of the frame transformation problem $(\mathcal{V}, f_k, f_{k+1}, S_k, \Omega_{k,k+1})$ for $0 \le k < T$.

B. The Pixel Contingency Formation Planning Problem

There is one more twist to the above setting regarding the user inputs. Suppose a player can choose to press a button at time t_{k_1} to make a character jump at time t_{k_2} as shown in Fig. 1, where $t_{k_1} < t_{k_2}$. Suppose the planner does not know t_{k_1}, the time the player makes the decision, until time t_{k_1}. In the worst case, the decision is made just one time step before t_{k_2} (i.e., $k_1 = k_2 - 1$). Before the player makes the decision, the planner needs to consider two scenarios: "jump" and "no jump". In the two scenarios, the sequences of frames starting at t_{k_2} diverge at time t_{k_2+1}. Let $\mathbf{F}_{\text{"jump"}}$ and $\mathbf{F}_{\text{"no jump"}}$ be the sequences of frames starting at t_{k_2} in the two scenarios. Note that the first frame in both $\mathbf{F}_{\text{"jump"}}$ and $\mathbf{F}_{\text{"no jump"}}$ are the same, but the pixel trajectory tracking specifications of the first frame can be different. The planner cannot wait until it knows the player's decision and must generate a solution that can render *both* frame sequences before t_{k_2}. In other words, a planner needs to generate a *contingency* formation plan to cover both scenarios in order to provide the *real-time guarantee* that a frame can be rendered at its designated time regardless of the decision. There is a *decision variable* x associated with the frame at time t_{k_2} that records the decision made by the player. The domain of x is $\mathsf{dom}(x)$, which includes the *options* for x and a symbol nil. In this example, $\mathsf{dom}(x) = \{\text{"jump"}, \text{"no jump"}, \mathsf{nil}\}$. Initially, the planner does not know the decision made by the player, and $x = \mathsf{nil}$. When the player chooses $o \in \mathsf{dom}(x)$ before t_{k_2}, we set $x = o$. Then, at time t_{k_2}, the formation plan for $\mathbf{F}_o$ will be executed since the frames in $\mathbf{F}_o$ will be rendered.

Let us consider the general case in which a player has to make many decisions over time. The given set of frames for rendering is organized in a tree structure called *a frame tree*, which is defined as $\mathbf{T} = (\mathbf{F}, f^{\mathsf{root}}, \mathsf{child}, \mathsf{next})$, where 1) $\mathbf{F}$ is a set of frames, 2) $f^{\mathsf{root}} \in \mathbf{F}$ is the root of the frame tree, 3) child is a mapping $\mathbf{F} \to 2^{\mathbf{F}}$ such that $\mathsf{child}(f)$ is the set of *child frames* of any $f \in \mathbf{F}$, and 4) next is a bijection $\mathbf{F} \times \mathbf{O} \to \mathbf{F}$ where $\mathbf{O}$ is the set of all possible options such that $\mathsf{next}(f, o)$ is a child frame of f.

If $|\mathsf{child}(f)| = 0$, we say f is a *terminal frame*. If $|\mathsf{child}(f)| > 1$, we say f is a *decision frame*. Every decision frame is associated with exactly one decision variable x such that $|\mathsf{dom}(x)| = |\mathsf{child}_k(f)| + 1$, where the domain $\mathsf{dom}(x)$ is a set of options for x. We assume every decision variable x is unique in a frame tree and can only be associated with one decision frame. $\mathsf{next}(f, o) \in \mathsf{child}(f)$ is the next frame if the player chooses $o \in \mathsf{dom}(x)$, where f is the decision frame with decision variable x. One of the options in $\mathsf{dom}(x)\backslash\{\mathsf{nil}\}$ is designated as the *default* option for x. If f^{root} is a decision frame and its decision variable is nil, we assume the default option of the decision variable is chosen.

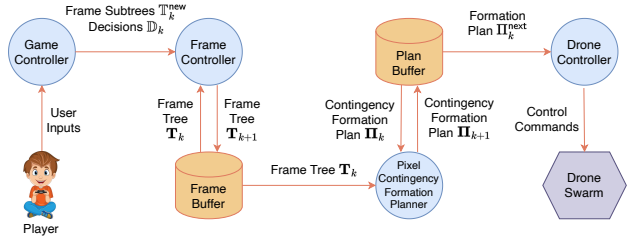

Fig. 3: The architecture of the interactive drone light show system.

Definition 4: A *pixel contingency formation planning problem* is a triple $(\mathcal{V}, \mathbf{T}, S_0)$, where 1) $\mathcal{V}$ is a set of drones, 2) $\mathbf{T} = (\mathbf{F}, f^{\mathsf{root}}, \mathsf{child}, \mathsf{next})$ is a frame tree, and 3) S_0 is a formation of $\mathcal{V}$ such that f^{root} is rendered by S_0. The solution of a pixel contingency formation planning problem is *contingency formation plan* $\mathbf{\Pi}$, which is a set of formation plans $\{\Pi_{k_1,k_2}\}_{f_{k_1}, f_{k_2} \in \mathbf{F}_i \text{ s.t. } f_{k_2} \in \mathsf{child}(f_{k_1})}$ such that Π_{k_1,k_2} is a solution of the frame transformation problem $(\mathcal{V}, f_{k_1}, f_{k_2}, S_{k_1}, \Omega_{k_1,k_2})$.

V. INTERACTIVE DRONE LIGHT SHOW SYSTEM

Fig. 3 shows the architecture of our interactive drone light show system. Unlike computer monitors, which can render a new frame just in time, drone swarms cannot render a new frame instantly since drones take time to move around. Therefore, the formation planner has to know some future frames in order to plan ahead where drones should go. In this paper, we assume the frame controller can obtain the information of some future frames from the *game controller*. There is a *frame buffer* that stores the frames. At each time step, the game controller can generate additional frames that are inserted into the frame buffer. The frame controller stores these future frames in the frame buffer. Moreover, the game controller interprets the user inputs and sets the decision variables in the frame tree in the frame buffer. A *pixel contingency formation planner* updates the contingency formation plan in the *plan buffer* based on the frame tree in the frame buffer so that *all* the frames in the frame tree can be rendered at the designated times. If the update can be finished within one time step, we can guarantee that all frames can be rendered at their designated times. At each time step, the *drone controller* executes the first formation plan for the root of the frame tree in the contingency formation plan. This formation plan will be removed from the frame tree at the end of the time step.

Algorithm 1 is the pseudocode of the frame controller. Initially, the frame controller is given a frame tree $\mathbf{T}_0 = (\mathbf{F}_0, f_0^{\mathsf{root}}, \mathsf{child}_0, \mathsf{next}_0)$ and $\mathbf{\Pi}_0$, where $\mathbf{\Pi}_0$ is a solution of $\mathbf{T}_0$. The execution loop does not terminate until the game controller signals that the game has ended or f_k^{root} is a terminal frame. First, in each time step t_k, the frame controller executes Π_k^{next} that transforms f_k^{root} into f_k^{next} (Lines 6–8). If f_k^{root} is a decision frame, f_k^{next} is $\mathsf{next}_k(f_k^{\mathsf{root}}, o)$, where o is either the chosen option or the default option if no option is chosen.

Algorithm 1 The execution loop of the frame controller.

1: **procedure FrameController**
2: $k := 0$. Let t_0 be the current time.
3: Let $\mathbf{T}_0 = (\mathbf{F}_0, f_0^{\mathsf{root}}, \mathsf{child}_0, \mathsf{next}_0)$ be the frame tree at t_0.
4: Let $\mathbf{\Pi}_0$ be the contingency formation plan for $\mathbf{T}_0$ at t_0.
5: **while** the game has not ended and f_k^{root} is not terminal **do**
6: Let $f_k^{\mathsf{next}} \in \mathsf{child}(f_k^{\mathsf{root}})$ be the unique child frame of f_k^{root}.
7: Let $\Pi_k^{\mathsf{next}} \in \mathbf{\Pi}$ be the formation plan from f_k^{root} to f_k^{next}.
8: Start executing Π_k^{next} for one time step.
9: Obtain new frame subtrees $\mathbb{T}_k^{\mathsf{new}}$ from the game controller.
10: Remove f_k^{root} from $\mathbf{T}_k$. Add $\mathbb{T}_k^{\mathsf{new}}$ to $\mathbf{T}_k$ to form $\mathbf{T}''_{k+1}$.
11: Remove Π_k^{next} from $\mathbf{\Pi}_k$ to form $\mathbf{\Pi}'_{k+1}$.
12: Let $\mathbb{D}_k$ be the decisions made by the player in $[t_{k-1}, t_k)$.
13: Remove subtrees inconsistent with $\mathbb{D}_k$ from $\mathbf{T}''_{k+1}$ to form $\mathbf{T}_{k+1}$. Remove the formation plans for the frames in the removed subtrees from $\mathbf{\Pi}'_{k+1}$ to form $\mathbf{\Pi}''_{k+1}$.
14: Let S_{k+1} be the formation of $\mathcal{V}$ according to Π_k^{next} at t_{k+1}
15: $\mathbf{\Pi}_{k+1} :=$ **SPICOMP**$(\mathcal{V}, \mathbf{T}_{k+1}, S_{k+1}, \mathbf{\Pi}''_{k+1})$
16: **if** $\mathbf{\Pi}_{k+1} = \mathsf{nil}$, **then return** nil
17: $f_{k+1}^{\mathsf{root}} := f_k^{\mathsf{next}}$; $k := k + 1$.

Second, the frame controller updates $\mathbf{T}_k$ and $\mathbf{\Pi}_k$ based on the set $\mathbb{T}_k^{\mathsf{new}}$ of new *frame subtrees* obtained from the game controller at t_k (Lines 9–10). The current frame f_0^{root} and the current formation plan Π_k^{next} are removed from $\mathbf{T}_k$ and $\mathbf{\Pi}_k$ to form $\mathbf{T}'_{k+1} = (\mathbf{F}'_{k+1}, f_{k+1}^{\mathsf{root}}, \mathsf{child}'_{k+1}, \mathsf{next}'_{k+1})$ and $\mathbf{\Pi}'_{k+1} = \mathbf{\Pi}_k \setminus \{\Pi_k^{\mathsf{next}}\}$, respectively, where $\mathbf{F}'_{k+1} = \mathbf{F}_k \setminus \{f_0^{\mathsf{root}}\}$ and child'_{k+1} is the same as child_k except f_0^{root} is removed from the range of child_k (Lines 10–11). Then the new frame subtrees in $\mathbb{T}_k^{\mathsf{new}}$ are attached to $\mathbf{T}'_{k+1}$ to form $\mathbf{T}''_{k+1}$. For each new frame tree $\mathbf{T}_{k'} = (\mathbf{F}_{k'}, f_{k'}^{\mathsf{root}}, \mathsf{child}_{k'}, \mathsf{next}_{k'}) \in \mathbb{T}_k^{\mathsf{new}}$, $f_{k'}^{\mathsf{root}}$ must be a frame in $\mathbf{F}'_{k+1}$, and all other frames in $\mathbf{F}_{k'} \setminus \{f_{k'}^{\mathsf{root}}\}$ are not in $\mathbf{F}'_{k+1}$. Then the frame controller obtains $\mathbf{T}''_{k+1} = (\mathbf{F}''_{k+1}, f_{k+1}^{\mathsf{root}}, \mathsf{child}''_{k+1}, \mathsf{next}''_{k+1})$ by adding $\mathbf{F}_{k'}$ to $\mathbf{F}'_{k+1}$ and updating child'_{k+1} according to $\mathsf{child}_{k'}$ to form child''_{k+1}. A new decision variable is introduced to $f_{k'}^{\mathsf{root}}$ if $|\mathsf{child}''_{k+1}(f_{k'}^{\mathsf{root}})| > 1$ in $\mathbf{T}''_{k+1}$.

Third, the frame controller removes some subtrees in $\mathbf{T}''_{k+1}$ according to $\mathbb{D}_k$, the set of decisions made by the player in the previous time step $[t_{k-1}, t_k)$ (Lines 12–13). A *decision* in $\mathbb{D}_k$ is a pair $(x_{k'}, o_{k'})$, where $x_{k'}$ is the decision variable of the frame $f_{k'} \in \mathbf{F}'_{k+1}$ and $o_{k'} \in \mathsf{dom}(x_{k'}) \setminus \{\mathsf{nil}\}$. For any $(x_{k'}, o_{k'}) \in \mathbb{D}_k$ such that $f_{k'} \in \mathbf{F}''_{k+1}$, we say the subtree of $\mathbf{T}''_{k+1}$ rooted at $\mathsf{next}_{k'}(o'_{k'}) \in \mathsf{child}''_{k+1}(f_{k'})$ is *inconsistent* with the decision $(x_{k'}, o_{k'})$, where $o'_{k'}$ is a non-nil option of $x_{k'}$ that is not $o_{k'}$. The frame controller removes all subtrees inconsistent with any decision in $\mathbb{D}_k$ from $\mathbf{T}''_{k+1}$ to obtain $\mathbf{T}_{k+1}$. After that, the frame controller removes the formation plans for the frames in the removed subtrees from $\mathbf{\Pi}'_{k+1}$ to form $\mathbf{\Pi}''_{k+1}$.

Finally, the frame controller uses SPICOMP, our pixel contingency formation planner, to solve the pixel contingency formation planning problem $(\mathcal{V}, \mathbf{T}_{k+1}, S_{k+1})$ before the next time step t_{k+1} (Line 15). The planner is given at most a duration of T_{step} to solve the problem. Although this timing constraint is tight, the planner does not have to solve the problem from scratch—it can reuse some parts of the contingency formation plan $\mathbf{\Pi}''_{k+1}$ in the previous step to

speed up the planning process. However, if SPICOMP takes too much time or the pixel contingency formation planning problem has no solution, the planner fails to return a solution and the frame controller terminates (Line 16).

VI. PIXEL FORMATION PLANNING ALGORITHMS

This section presents an algorithm called SPICOMP, which stands for the **S**ampling-based **PI**xel **CO**ntingency for**M**ation **P**lanning algorithm. SPICOMP solves the pixel contingency formation planning problem by separating the planning of drones for pixel rendering from the planning of hidden drones. Given a frame tree $\mathbf{T}$, it conducts a forward search starting from the root of $\mathbf{T}$ and iteratively solves the frame transformation problem between every two consecutive frames. After finding a formation plan for transforming one frame to the next, it conducts a backward search to update the motion plans for the hidden drones involved in the formation plan. Since the hidden drones have more freedom to move around, it is easier to relocate them by replanning than deciding where they should go in the forward search. Both the forward search and the backward search use sampling methods to select drones for frame transformation and replanning. The pseudocode of SPICOMP can be found in the accompanying video.

A. The Forward Search

The forward search is a depth-first search (DFS) that expands the child frames of the current frame repeatedly, starting with the root of $\mathbf{T}$. The RandomAssignment function returns a *partial* assignment g_k^{partial} for the pixels in frame f_k. Suppose f_k is partitioned into f_k^{track} and f_k^{hop}, which are the set of pixels involved in pixel trajectory tracking and pixel hopping, respectively. For every pixel $p \in f_k^{\mathsf{track}}$, RandomAssignment must assign p to the same drone assigned to the corresponding pixel in f_0^{root} according to the pixel trajectory tracking specification for p. By contrast, RandomAssignment has the freedom to assign a pixel in f_k^{hop} to any drone not assigned to the pixels in f_k^{track}. RandomAssignment uses a random sampling scheme as follows: the probability of assigning a drone to a pixel $p \in f_k^{\mathsf{hop}}$ depends on the distance between the drone and the pixel. Let $D_{i,j}$ be the distance between a drone $v_i \in \mathcal{V}'$ *not* involved in pixel trajectory tracking and a pixel $p_j \in f_k^{\mathsf{hop}}$. Let $\mathcal{V}_j = \{v_i\}_{v_i \in \mathcal{V}' \text{ s.t. } D_{i,j} \leq D_{\mathsf{max}}^{\mathsf{assign}}}$, where $D_{\mathsf{max}}^{\mathsf{assign}}$ is the estimated maximum distance that a drone can fly within one time step in spite of the time delay for collision avoidance. The weight of a drone $v_i \in \mathcal{V}_j$ is $w_{i,j} = \frac{1}{D_{i,j}+\epsilon}$, where ϵ is a small positive number. Then the probability of assigning $v_i \in \mathcal{V}_j$ to p_j is $\frac{w_{i,j}}{\sum_{v_i \in \mathcal{V}_j} w_{i,j}}$. To avoid assigning a drone to two different pixels, whenever $v_i \in \mathcal{V}_j$ is assigned to p_j, v_i is removed from $\mathcal{V}_{j'}$ for all $j' \neq j$. Then $w_{i',j'}$ is recalculated accordingly for all $v_{i'} \in \mathcal{V}_{j'}$. Note that we use the most constrained variable heuristic to order the pixels for assignment in ascending order of the size of $\mathcal{V}_j$ [18].

After choosing g_k^{partial}, SPICOMP uses FramePlanner to solve the frame transformation problem. FramePlanner returns a *partial* formation plan Π^{partial} for drones that has been assigned in g_k^{partial}, and ignores the unassigned drones. Many existing motion planning algorithms can be used as FramePlanner to solve the frame transformation problem by generating *safe* formation plans quickly (e.g., [13], [19], [20], [21], [22], [23], [24], [25]).

B. The Backward Search

Next, SPICOMP completes g_k^{partial} by finding another partial assignment g_k^{relocate} for the unassigned pixels in $f_k^{\mathsf{unassigned}} \subseteq f_k^{\mathsf{hop}}$. Since these pixels are too far away from all drones or all drones near them have been assigned to other pixels, we must relocate some unassigned drones so that these pixels can be rendered. Let $\mathcal{V}^{\mathsf{unassigned}}$ be the set of unassigned drones not involved in g_k^{partial}. In RelocateHidden, RandomRelocation randomly generates g_k^{relocate}, a mapping from $f_k^{\mathsf{unassigned}}$ to $\mathcal{V}^{\mathsf{unassigned}}$, such that no two drones in $\mathcal{V}^{\mathsf{unassigned}}$ are assigned to the same pixel in $f_k^{\mathsf{unassigned}}$. RandomRelocation uses the same assignment scheme for RandomAssignment, except it calculates $D_{i,j}$ differently. For each $v_i \in \mathcal{V}^{\mathsf{unassigned}}$, RandomRelocation searches for the *earliest* state s_i^{earliest} of v_i that is available for relocation by searching backward in $\mathbf{\Pi}_k^{\mathsf{tmp}}$. Since the forward search constructs $\mathbf{\Pi}_k^{\mathsf{tmp}}$ incrementally, $\mathbf{\Pi}_k^{\mathsf{tmp}}$ contains all the formation plans from f_0^{root} to f_k. Hence, we can trace the state sequence of any drone since f_0^{root}. s_i^{earliest} is the state of v_i in f_i^{earliest} at time t_i^{earliest} such that 1) v_i goes dark after t_i^{earliest} and 2) all frames between f_i^{earliest} and f_k, exclusively, are not decision frames at which v_i has been assigned to move to another pixel's position in another branch of the frame tree. Hence, v_i is ready for relocation at t_i^{earliest}, and the relocation would not affect the formation plans for the other branches of the frame tree. Let $D_{i,j} = \mathsf{dist}(\mathsf{pos}[s_i^{\mathsf{earliest}}], \mathsf{pos}[p_j])$ for every $v_i \in \mathcal{V}^{\mathsf{unassigned}}$ and $p_j \in f_i^{\mathsf{unassigned}}$. $D_{i,j}$ is *valid* if v_i can fly from $\mathsf{pos}[s_i^{\mathsf{earliest}}]$ to $\mathsf{pos}[p_j]$ within a duration of $t_k - t_i^{\mathsf{earliest}}$, where t_k is the designated time of f_k. Let $\mathcal{V}'_j = \{v_i\}_{v_i \in \mathcal{V}^{\mathsf{unassigned}} \text{ s.t. } D_{i,j} \text{ is valid}}$. Then RandomRelocation generates g_k^{relocate} using the same assignment scheme for RandomAssignment except $\mathcal{V}'_j$ replaces $\mathcal{V}_j$.

After generating g_k^{relocate}, RelocateHidden computes the trajectories of the drones chosen by g_k^{relocate} by moving the drones from their earliest states to the pixels' positions. We shall use an existing motion planning algorithm, namely MotionPlanner, to generate these trajectories that avoid collision with other drones in $\mathbf{\Pi}_k^{\mathsf{tmp}}$ and Π^{partial}. Next, RelocateHidden updates $\mathbf{\Pi}_k^{\mathsf{tmp}}$ and Π^{partial} by incooperating the trajectories into the formation plan. Finally, RelocateHidden returns Π, which extends Π^{partial} to include the formation plans for the selected drones. If RelocateHidden fails to find an assignment g_k^{relocate} for all pixels in $f_k^{\mathsf{unassigned}}$ or MotionPlanner failed to find the trajectories, RelocateHidden returns nil and DFS will try again with new random assignments g_k^{partial} and g_k^{relocate}. If the number of trials is greater than $Trial_{\mathsf{max}}$, DFS backtracks to the previous frame in the frame tree.

To avoid repeating the planning effort in the previous time step, both RandomizeAssignment and RandomRelocation use the same assignment in $\mathbf{\Pi}^{\mathsf{last}}$ when $Trial$ is 1. Therefore, if no new frame is added to the frame tree in the previous

time step, both g_k^{partial} and g_k^{relocate} remains unchanged, and $\mathbf{\Pi}^{\mathsf{tmp}} = \mathbf{\Pi}^{\mathsf{last}}$ except the root formation plan is removed.

VII. EXPERIMENTAL EVALUATION

We conducted an experiment to compare SPICOMP with and without the backward search. Moreover, we conducted two additional experiments to examine the effects of changing 1) the number of drones relative to the number of pixels in a frame and 2) the size of the frame buffer. This section presents and discusses the results of the experiments.

Experimental Setup: We implemented our interactive drone light show system in a simulator written in C++. We randomly generated 50 gameplays in which characters were simple geometrical shapes that moved around in random cyclic paths. The motion of the pixels on the shapes was based on pixel trajectory tracking. These shapes can spawn smaller shapes as bullets in response to user inputs. In every gameplay, the number of drones was larger than the number of pixels in any frame. Initially, some drones were put in the initial formation of the first frame, whereas the remaining drones were hidden at random locations. We implemented a formation planner to solve the frame transformation problem. The planner generates motion plans in which drones fly in strange lines. When two drones were on a collision course, the planner will delay the start time of one of the motion plans to avoid the collision [25]. The parameters being used in the experiments are: $Trial_{\mathsf{max}} = 20$, $T_{\mathsf{step}} = 0.1$s. The experiments were conducted on an Apple laptop with M1 CPU and 16GB RAM.

Results: Fig. 4 shows that the running time of an iteration of SPICOMP with the backward search is much shorter than that without the backward search. As the number of drones increases, the gap between the two lines in Fig 4 increases. The error bars are the 95% confidence interval. The ratio of the number of drones to the number of pixels is 1.5 in all cases. Without the backward search, SPICOMP has to decide where to move the hidden drones in the forward search. If the forward search puts the hidden drones away from the positions that require the hidden drones in the subsequent frames, backtracking occurs in the DFS, causing a longer running time. With the backward search, the positions of the hidden drones were decided based on demand, and the replanning often succeeded. Therefore, the backward search helps reduce the running time significantly.

Fig. 5 shows that the running time of SPICOMP decreased sharply as the ratio γ of the number of drones to the maximum number of pixels in a frame increased. When γ was close to 1, there were not enough drones to move around to show every frame. Even when there were enough drones, SPICOMP needed to take a few more samples in each iteration to generate a contingency formation plan. When γ was large, SPICOMP can easily solicit hidden drones for pixel hopping and return a solution quickly.

We also conducted an experiment to show how the required number n_{min} of drones changed with the frame buffer size, where n_{min} is the minimum number of drones with which SPICOMP can return a solution within one time step.

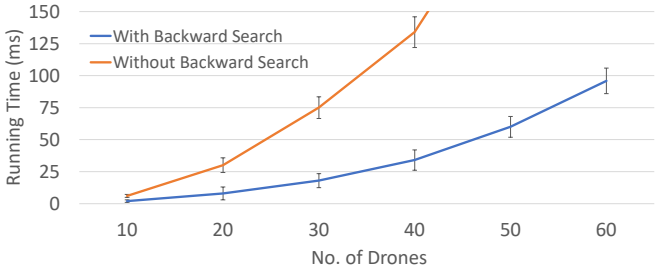

Fig. 4: SPICOMP's running time (with and without the backward search) vs. the number of drones.

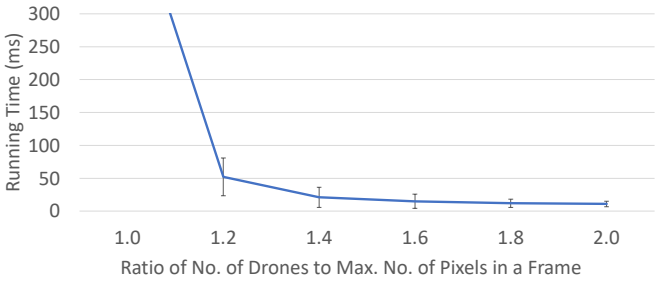

Fig. 5: SPICOMP's running time vs. the ratio of the number of drones to the maximum number of pixels in a frame.

By limiting the number of frames in the frame buffer, we found n_{min} by gradually increasing the number of drones until SPICOMP can return a solution. When the frame buffer size was small, SPICOMP was forced to generate short contingency formation planning that gave drones very little time to react to the user inputs, and thus, it needed many hidden drones. When the frame buffer size was large, the contingency formation plans simply required more drones to operate. Thus, the optimal frame buffer size is one that strikes a balance between the two extremes.

VIII. SUMMARY AND FUTURE WORK

This paper describes an interactive drone light show system for real-time user inputs. Our system relies on a simpling-based contingency formation planner called SPICOMP, which leverages hidden drones to achieve fast response time. To our knowledge, SPICOMP is the first contingency formation planner for drone light shows. According to the contingency formation plan returned by SPICOMP, pixels can be displayed correctly despite the uncertainty of user inputs. However, if a frame tree is too large, SPICOMP could take more than one time step to update a contingency formation plan. Moreover, SPICOMP is a sampling algorithm that cannot guarantee returning a solution, especially when the number of drones is close to the number of pixels in a frame. In the future, we intend to identify the conditions under which a planning algorithm exists that can compute a contingency formation plan in one step.

ACKNOWLEDGMENTS

This work was supported by National Research Foundation of Korea RS-2022-NR069751 (or 2022R1A2C101216813).

REFERENCES

[1] H. Kim and J. A. Landay, "Aeroquake: Drone augmented dance," in *Proceedings of the Designing Interactive Systems Conference*, no. 11, 2018, pp. 691–701.

[2] M. Kljun, K. Čopič Pucihar, M. Lochrie, and P. Egglestone, "Streetgamez: A moving projector platform for projected street games," in *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, 2015, pp. 589–594.

[3] S. J. Kim, Y. Jeong, S. Park, K. Ryu, and G. Oh, "A survey of drone use for entertainment and avr (augmented and virtual reality)," in *Augmented Reality and Virtual Reality: Empowering Human, Place and Business*. Springer, 2018, pp. 339–352.

[4] X. Du, C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Fast and in sync: Periodic swarm patterns for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9143–9149.

[5] A. You, J. Li, Z. Xu, and W. Liu, "Design of entertainment drone with rotating LED display technology," *Journal of Physics: Conference Series*, vol. 2113, no. 1, 2021.

[6] P. Mao, Y. Gao, B. Wang, A. Yan, X. Chi, and Q. Quan, "Fast light show design platform for k-12 children," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9396–9402, 2021.

[7] K.-C. Weng, S.-T. Lin, C.-C. Hu, R.-T. Soong, and M.-T. Chi, "Multi-view approach for drone light show," *The Visual Computer*, 2022.

[8] J. Yu and S. M. LaValle, "Shortest path set induced vertex ordering and its application to distributed distance optimal formation path planning and control on graphs," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 2775–2780.

[9] ——, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[10] Y. Yang, X. Xiong, and Y. Yan, "Uav formation trajectory planning algorithms: A review," *Drones*, vol. 7, no. 1, p. 62, 2023.

[11] H. Duan, Q. Luo, Y. Shi, and G. Ma, "Hybrid particle swarm optimization and genetic algorithm for multi-uav formation reconfiguration," *IEEE Computational Intelligence Magazine*, vol. 8, no. 3, pp. 16–27, 2013.

[12] M. Turpin, N. Michael, and V. Kumar, "Trajectory design and control for aggressive formation flight with quadrotors," *Autonomous Robots*, vol. 33, pp. 143–156, 2012.

[13] H. Wang and M. Rubenstein, "Shape formation in homogeneous swarms using local task swapping," *IEEE Transactions on Robotics*, vol. 36, no. 3, pp. 597–612, 2020.

[14] P. Mao, Y. Gao, B. Wang, A. Yan, X. Chi, and Q. Quan, "Fast light show design platform for k-12 children," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9396–9402.

[15] H. Sun, J. Qi, C. Wu, and M. Wang, "Path planning for dense drone formation based on modified artificial potential fields," in *2020 39th Chinese Control Conference (CCC)*, 2020, pp. 4658–4664.

[16] D. Nar and R. Kotecha, "Optimal waypoint assignment for designing drone light show formations," *Results in Control and Optimization*, vol. 9, 2022.

[17] J. Huang, G. Tian, J. Zhang, and Y. Chen, "On unmanned aerial vehicles light show systems: Algorithms, software and hardware," *Applied Sciences*, vol. 11, no. 16, p. 7687, 2021.

[18] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Prentice Hall, 2020.

[19] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 3–19.

[20] P. Yang, K. Tang, J. A. Lozano, and X. Cao, "Path planning for single unmanned aerial vehicle by separately evolving waypoints," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1130–1146, 2015.

[21] O. Andersson, M. Wzorek, P. Rudol, and P. Doherty, "Model-predictive control with stochastic collision avoidance using bayesian policy optimization," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4597–4604.

[22] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, online collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

[23] F. Ho, R. Geraldes, A. Goncalves, M. Cavazza, and H. Prendinger, "Improved conflict detection and resolution for service UAVs in shared airspace," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1231–1242, 2018.

[24] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for MAVs in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[25] M. Park and T.-C. Au, "Wind field modeling for formation planning in multi-drone systems," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.