2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Multi-Agent Spatial Predictive Control with Application to Drone Flocking

Andreas Brandstätter[1a], Scott A. Smolka[2], Scott D. Stoller[2], Ashish Tiwari[3], Radu Grosu[1]

***Abstract*— We introduce *Spatial Predictive Control* (SPC), a technique for solving the following problem: given a collection of robotic agents with black-box positional low-level controllers (PLLCs) and a mission-specific distributed cost function, how can a distributed controller achieve and maintain cost-function minimization without a plant model and only positional observations of the environment? Our fully distributed SPC controller is based strictly on the position of the agent itself and on those of its neighboring agents. This information is used in every time step to compute the gradient of the cost function and to perform a spatial look-ahead to predict the best next target position for the PLLC. Using a simulation environment, we show that SPC outperforms Potential Field Controllers, a related class of controllers, on the drone flocking problem. We also show that SPC works on real hardware, and is therefore able to cope with the potential sim-to-real transfer gap. We demonstrate its performance using as many as 16 Crazyflie 2.1 drones in a number of scenarios, including obstacle avoidance.**

I. INTRODUCTION

A collection of drones can perform tasks that cannot be accomplished by individual drones alone [1]. It can, for example, carry a heavy load while still being much more agile than a single larger drone [2], [3]. In search-and-rescue applications, the drones can explore unknown terrain by covering individual paths that jointly cover the entire area [4]–[6]. These collective maneuvers can be expressed as the problem of minimizing a *positional cost function*, i.e., a cost function that depends on the positions of the drones (and possibly information about their environment). Such a problem formulation requires a method to localize each drone within a common reference frame, e.g. a Global Navigation Satellite System (GNSS) or an indoor localization system.

Off-the-shelf drones, such as Crazyflie [7], DJI [8], and Parrot [9], come equipped with a *positional low-level controller* (PLLC). Such a controller takes a position argument as input and maneuvers the drone to this position, where it then hovers. PLLCs are common in other types of robotic systems, including the Landshark [10] and Taurob [11] unmanned ground vehicles, and the Bluefin®-12 [12] unmanned underwater vehicle. Unfortunately, the PLLC's code is often proprietary, and the exact parameters of the physical drone model might not be available. Since the PLLC and physical drone together form the plant to be controlled, a dynamic model of the plant is often unavailable, for one or both of these reasons.

In this paper, we address the following problem: *Design a distributed controller that minimizes a given positional cost function for robotic agents with black-box PLLCs, no available model of the plant dynamics, and only positional observations of their environment.*

[1] CPS, Technische Universität Wien (TU Wien), Austria
[2] Department of Computer Science, Stony Brook University, USA
[3] Microsoft, USA
[a] Correspondence: andreas.brandstaetter@tuwien.ac.at

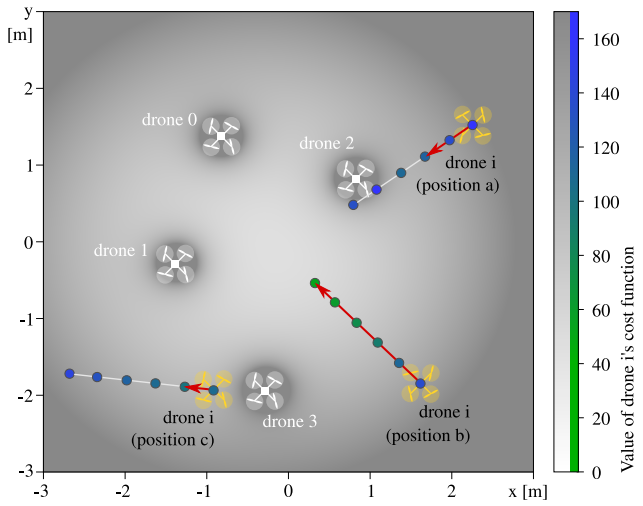

Fig. 1: Given the positions shown for drones 0 to 3, the greyscale heatmap indicates the value of the cost function for a drone i at each point, if it was placed at that point. The direction in which drone i should move is determined by the gradient of the cost function (shown for positions *a*, *b*, and *c*). SPC evaluates the cost at the spatial lookahead (indicated by the colored dots) along this direction and chooses the best value for drone i's next position (the red arrow).

To solve this problem, we introduce *Spatial Predictive Control* (SPC), a novel distributed high-level approach to multi-agent control. In SPC, each agent's controller identifies N equally-spaced points within a maximum look-ahead distance $\epsilon \cdot N$ from its current position in the direction of the negative gradient of a cost function c. The SPC controller then computes the value of c for each of these points and chooses the one with minimal cost as the target location to be sent to the PLLC. The PLLC makes a best effort to reach this location, while in the next time-step, SPC provides an updated target.

SPC vs MPC. To solve the stated problem, one might consider the PLLC as part of the plant and design a Model Predictive Control (MPC) for the high-level control. Such an approach is not applicable since neither a dynamic model of the physical plant nor the internals of the PLLC are available. Even if an approximate dynamic model could be obtained using system identification techniques, and if the code for the PLLC is available (e.g., for Crazyflies), MPC remains a computationally expensive method [13]. This is especially relevant for embedded processors with limited computing capabilities. MPC needs to calculate the predicted behavior for the plant model over a specified time horizon in order to search for an optimal control input. In contrast, SPC does not require a plant model and avoids extensive prediction calculations.

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10160617

SPC vs Planning. Given that (robotic) agents are equipped with PLLCs, one might ask if a controller is actually needed, or would a planning-based approach suffice. Based on the initial positions of agents and obstacles, a plan of way-points could be generated for the PLLC to follow. Since, however, the environment is constantly changing due to the movement of other agents (and possibly obstacles), such a plan would become quickly outdated. This is exactly the type of problem we address with SPC: in every time step, we use the observations of the environment to calculate the next input to the PLLC; the PLLC makes a best effort to reach this position. Hence, the key difference between SPC and planning is the granularity of the time horizon: planning uses long time horizons for calculating trajectories, whereas in our approach, feedback from the plant in every time step is used to recalculate the desired next position.

SPC vs PFC. Potential Field Controllers (PFCs) are well-known controllers for mobile robots. Prior work [14], [15] has considered their application to flocking. PFCs view the cost function as defining a potential field, and thus, PFCs use the gradient of the potential as the force (or acceleration) the controller needs to apply. There are two issues with using PFCs for our stated problem. First, when the environment has obstacles and a large number of moving drones, the cost function becomes time-varying and nonlinear (as in Figure 1), and *local* gradients become misleading. Second, in our setting, we can not set the acceleration directly because we only have access to the PLLC. Nevertheless, we can adapt and use PFC in our setting, but our experiments confirm that it performs poorly compared to SPC, which evaluates the cost function at *multiple* candidate future positions within the spatial look-ahead horizon to find the best next position (in terms of cost-function minimization).

Application to Drone Flocking: After introducing SPC as a general approach, we apply it to a distributed multi-agent system with the goal of achieving flock formation, and maintaining flock formation while moving to a specified target location, avoiding obstacles in the process.

The local cost function c for this problem (see Section III-B) depends only on the locations of the drones. As illustrated for four drones in Figure 1, the negation of the gradient of c (see Section III-C) suggests a direction of movement for drone i. The SPC algorithm (see Section II) evaluates its local cost function at multiple points in that direction. As the figure illustrates, this enables the drone to see peaks and valleys in the cost function that may lie ahead.

The **main contributions of this paper** are:

- We introduce the novel concept of *Spatial Predictive Control*, a control methodology well suited for PLLCs.
- SPC is *model-free*. One only needs to be able to determine the cost at different locations along the direction of the cost function's gradient in order to apply it. Also, SPC does not need to measure the velocity or acceleration of neighboring drones.
- We evaluate SPC using drone flocking in a drone simulation environment.
- We further experimentally validate our approach by achieving flocking with real drone hardware in the form of off-the-shelf Crazyflie quadcopters in a number of different scenarios.

II. SPC FOR MULTI-AGENT SYSTEMS

We describe the distributed control problem addressed in this paper and then present SPC for solving this problem.

A. Distributed Control for Distributed Cost Minimization in the Presence of PLLCs

We consider a multi-agent system consisting of a set D of agents. Every agent $i \in D$ has a state (x_{io}, x_{ih}), where x_{io} is the observable part of its state and x_{ih} is the hidden part of its state. Agent i has a control input u_i and its dynamics is assumed to be given by some unknown function f:

$$(\frac{dx_{io}(t)}{dt}, \frac{dx_{ih}(t)}{dt}) = f(t, x_{io}(t), x_{ih}(t), u_i(t)) \quad (1)$$

Agent i has access to the observable state of a subset $H_i \subseteq D$ of agents. H_i will be referred to as the *neighborhood* of i.

The objective for the multi-agent system is given in terms of a cost function $c(x_{io}, x_{Hio})$ that maps the observable state of agent i (x_{io}) and of its neighbors (x_{Hio}) to a non-negative real value. Here we use x_{Hio} as shorthand for $(x_{jo})_{j \in H_i}$. Agent i's goal is to minimize $c(x_{io}, x_{Hio})$.

In our setting, we do not have ability to directly set u_i. Instead, we can only set a *reference value* $x_{io}^{(r)}$ that is then used by some black-box, low-level controller $PLLC$ to internally set the control input.

$$u_i(t) = PLLC(t, x_{io}(t), x_{ih}(t), x_{io}^{(r)}) \quad (2)$$

Both the dynamics of each agent (function f) and the details of the PLLC (function $PLLC$) are unknown. The cost function c is given. We want to find a procedure that allows each agent to minimize its cost in the above setting.

B. Spatial Predictive Control (SPC)

Let $\nabla_{x_{io}} c(x_{io}, x_{Hio})$ denote the gradient of cost function c with respect to x_{io}. One way to minimize the cost $c(x_{io}, x_{Hio})$ would be to follow the negative of the gradient at every point. However, if the cost function is nonlinear (e.g., has many peaks), then gradients can be misleading. The key observation underlying SPC is that each agent should *look ahead in the observable state space*, in the direction of the negative gradient, to determine the reference value for its observable state. An SPC controller picks N equally-spaced points within a maximum look-ahead distance $\epsilon \cdot N$ from the current observable state and in the direction of the negative gradient, where ϵ and N are parameters of the controller. At any given state $(x_{io}(t), x_{Hio}(t))$, the set Q_i containing these equally-spaced points is given by:

$$Q_i = \left\{ x_{io}(t) - n \cdot \epsilon \cdot \frac{\nabla_{x_{io}} c(x_{io}(t), x_{Hio}(t))}{\|\nabla_{x_{io}} c(x_{io}(t), x_{Hio}(t)\|} \mid n = 0 \,..\, N \right\} \quad (3)$$

Here $\nabla_{x_{io}} c(x_{io}(t), x_{Hio}(t))$ denotes the evaluation of the gradient of c at the point $(x_{io}(t), x_{Hio}(t))$. Our spatial-predictive controller selects the point in Q_i with minimum cost as the next target position $x_{io}^{(r)}$ for agent i:

$$x_{io}^{(r)} = \underset{\tilde{x}_{io} \in Q_i}{\operatorname{argmin}} \left(c(\tilde{x}_{io}, x_{Hio}(t)) \right) \quad (4)$$

Note that the SPC controller recomputes the reference $x_{io}^{(r)}$ at each time step. This is important because this computation of

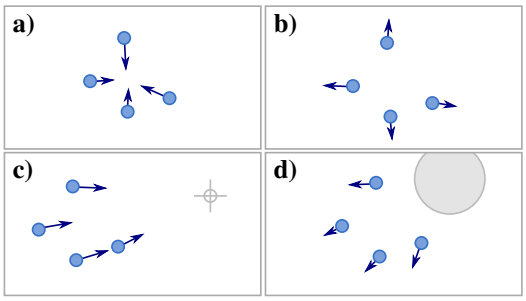

Fig. 2: Directional movements (indicated by arrows) induced by cost-function terms: **a**: *Cohesion*, **b**: *separation*, **c**: *target seeking*, and **d**: *obstacle avoidance*.

the reference does not take into account the motion of the neighbors. However, SPC will respond to any change in neighbors' observable states in its next computation of the reference.

III. Application to Multi-Agent Flocking Problem

This section starts with background on flocking, describes how to use SPC for this application, and then presents metrics to assess the quality of a flocking controller.

A. The Flocking Problem

A set of agents D is in a flock formation if the distance between every pair of agents is "not too large and not too small." These requirements yield the first two terms of our cost function: *cohesion* and *separation*. These two terms are sufficient to cause the agents to form and maintain a flock formation.

In our model, drone i has access to positions of only a subset H_i of drones, namely its local neighbors. Hence, we define a local cost function, parameterized by i, which uses only the positions of drones in $\{i\} \cup H_i$.

B. Cost Function

Consider a drone i, i in D. Let p_j, when it appears in the local cost function of drone i, denote the position of drone j as known to drone i; this may differ from the actual position due to sensing error. Let p_{H_i} denote the tuple of positions of drones in H_i and let r_d be the radius of each drone. We define the cost function $c(p_i, p_{H_i})$ as:

$$\begin{aligned} c(p_i, p_{H_i}) =& c_{coh}(p_i, p_{H_i}) + c_{sep}(p_i, p_{H_i}) + \\ & c_{tar}(p_i, p_{H_i}) + c_{obs}(p_i) \end{aligned} \tag{5}$$

The value of the *cohesion* term increases as drones drift apart, and the *separation* term increases as drones get closer together. Each term has a weight, denoted by a subscripted ω.

Cohesion term:

$$c_{coh}(p_i, p_{H_i}) = \omega_{coh} \cdot \frac{1}{|H_i|} \cdot \sum_{j \in H_i} \|p_i - p_j\|^2 \tag{6}$$

Separation term:

$$c_{sep}(p_i, p_{H_i}) = \omega_{sep} \cdot \frac{1}{|H_i|} \cdot \sum_{j \in H_i} \frac{1}{max(\|p_i - p_j\| - 2r_d, \hat{0})^2} \tag{7}$$

The function $max(., \hat{0})$ ensures positive values when there is sensor noise, but does not further influence the cost function; $\hat{0}$ denotes a very small positive value.

The mission-specific *target seeking* term sets a target location, denoted by p_{tar}, for the entire flock. The *obstacle avoidance* term prevents the drones from colliding with infinitely tall cylindrical objects. Let K denote the set of obstacles. For $k \in K$, let r_k denote the radius of obstacle k, and let p_k denote its center on the xy-plane.

Target-seeking term:

$$c_{tar}(p_i, p_{H_i}) = \omega_{tar} \cdot \left\| p_{tar} - \frac{p_i + \sum_{j \in H_i} p_j}{|H_i| + 1} \right\|^2 \tag{8}$$

Obstacle-avoidance term:

$$c_{obs}(p_i) = \omega_{obs} \cdot \frac{1}{|K|} \cdot \sum_{k \in K} \frac{1}{max(\|\mathcal{P}(p_i) - p_k\| - r_k - r_d, \hat{0})^2} \tag{9}$$

The function $\mathcal{P}(.)$ projects a vector to the xy-plane.

C. Gradient of the cost function

For SPC, the gradient of the cost function is required in Eq. (3), and is given by (for readability, we elide function arguments):

$$\nabla_{p_i} c = \nabla_{p_i} c_{coh} + \nabla_{p_i} c_{sep} + \nabla_{p_i} c_{tar} + \nabla_{p_i} c_{obs} \tag{10}$$

Cohesion gradient:

$$\nabla_{p_i} c_{coh} = 2 \cdot \omega_{coh} \cdot \left(p_i - \frac{1}{|H_i|} \cdot \sum_{j \in H_i} p_j \right) \tag{11}$$

Separation gradient:

$$\nabla_{p_i} c_{sep} = \frac{2 \cdot \omega_{sep}}{|H_i|} \cdot \sum_{j \in H_i} \frac{p_j - p_i}{(\|p_i - p_j\| - 2r_d)^3 \cdot \|p_i - p_j\|} \tag{12}$$

Target-seeking gradient:

$$\nabla_{p_i} c_{tar} = \frac{2 \cdot \omega_{tar}}{|H_i| + 1} \cdot \left(\frac{p_i + \sum_{j \in H_i} p_j}{|H_i| + 1} - p_{tar} \right) \tag{13}$$

Obstacle-avoidance gradient:

$$\nabla_{p_i} c_{obs} = \frac{2\omega_{obs}}{|K|} \cdot \sum_{k \in K} \frac{p_k - \mathcal{P}(p_i)}{(\|\mathcal{P}(p_i) - p_k\| - r_k - r_d)^3 \cdot \|\mathcal{P}(p_i) - p_k\|} \tag{14}$$

D. Flock-Formation Quality metrics

Collision avoidance: To avoid collisions, the distance between all pairs of drones must remain above a specified threshold $dist_{thr}$. We define a metric for the minimum distance between any pair of drones as follows:

$$dist_{min} = \min_{i,j \in D; i \neq j} \|p_i - p_j\| \tag{15}$$

We set $dist_{thr} = 2 \cdot r_d + r_{safety}$, where r_d is the radius of the drone, and r_{safety} is a safety margin.

Compactness: Compactness of the flock is measured by the maximum distance of any drone from the centroid of the flock. It is defined as follows:

$$comp_{max} = \max_{i \in D} \left\| \frac{\sum_{j \in D} p_j}{|D|} - p_i \right\| \tag{16}$$

It is expected to stay below some threshold $comp_{thr}$; otherwise, the drones are too far apart.

Obstacle clearance: Keeping a safe distance from obstacles is required to avoid collisions. We therefore measure the minimum distance from any drone to any obstacle:

$$clear_{obj} = \min_{i \in D; k \in K} \|\mathcal{P}(p_i) - p_k\| \quad (17)$$

For safety, this should always be greater than some threshold $clear_{thr} = r_d + r_k + r_{safety}$.

IV. Experimental Evaluation

We evaluated SPC on the drone flocking problem using simulations and experiments with Crazyflie 2.1 drones.

A. Simulation Experiments

As a simulation framework, we use *crazys* [16], which is based on the *Gazebo* [17] physics and visualization engine and the Robotic Operating System (ROS) [18]. Our SPC algorithm is implemented in C++ as a separate ROS node. It receives position messages from neighboring drones, and control messages, such as the target location or a stop command, from the human operator. It outputs a set-point to the PLLC. The SPC we implemented is fully distributed: there is no central optimizer and no further information is exchanged between ROS nodes. The SPC node calculates the gradient according to Eqs. (11)-(14). The spatial look-ahead parameter N is determined dynamically based on the distance to the target location, where $N^* \in \mathbb{N}^+$ is a system parameter:

$$N = \lceil N^* \cdot max(1, min(1.5 \cdot (\|p_i - p_{tar}\| + 0.5), 3)) \rceil \quad (18)$$

This allows the drones to more quickly reach distant target locations and reduces the controller's computational cost (by reducing $|Q_i|$) once the flock reaches the target. Thereafter, the set-point position $x_{io}^{(r)}$ is determined by Eqs. (3)-(4). Auxiliary functions, like hovering at the starting position, are also implemented in this node. In the simulations, we added Gaussian sensor noise, with $\sigma = 10cm$, for drone position measurements. Cost-function weights and controller parameters (Table I) were determined empirically by analysis of the controller behavior. Note that the maximum look-ahead distance $\epsilon \cdot N$ should not be too large, to avoid "seeing through" other drones or obstacles. On the other hand, a small value for N reduces the granularity of the controller action space, leading to a bang–bang controller if $N = 1$.

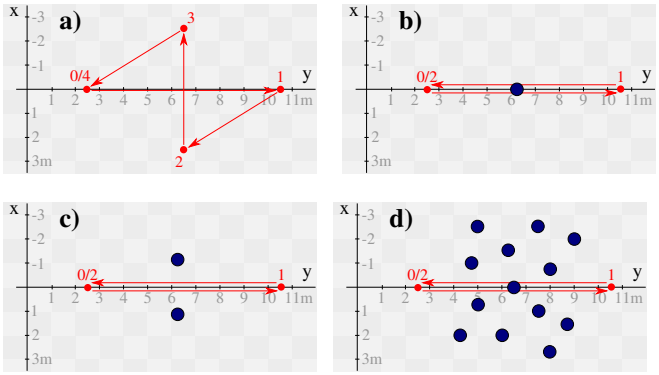

Fig. 3: Experiments were performed using four different scenarios: **a**: without obstacles, **b**: with one obstacle, **c**: with 2 obstacles, and **d**: with 13 obstacles indicated in dark-blue. The direct path between the numbered target locations p_{tar} (red dots) is indicated with red arrows. (z-dimension is elided in these plots, since it is constant for all target locations.)

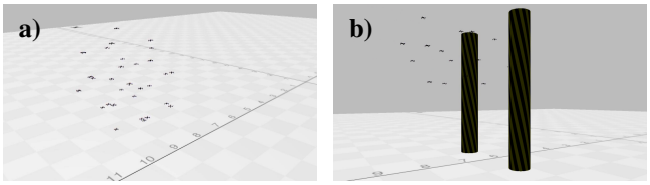

Fig. 4: Simulation experiments using simulation environment. Snapshot of **a**: flock of 30 drones; and **b**: flock of 15 drones with 2 obstacles.

		PLLC A	PLLC B	Hardware
Cost weights	ω_{coh}	$20\,m^{-1}$	$20\,m^{-1}$	$20\,m^{-1}$
	ω_{sep}	$12\,m^{-1}$	$12\,m^{-1}$	$12\,m^{-1}$
	ω_{tar}	$150\,m^{-1}$	$150\,m^{-1}$	$150\,m^{-1}$
	ω_{obs}	$18\,m^{-1}$	$18\,m^{-1}$	$35\,m^{-1}$
SPC parameters	N^*	6	3	3
	ϵ	$0.05\,m$	$0.05\,m$	$0.04\,m$
PFC gain	k	$0.003\,m$	$0.0015\,m$	n.a.
Dimensions	r_d	$0.07\,m$	$0.07\,m$	$0.07\,m$
	r_k	$0.25\,m$	$0.25\,m$	$0.25\,m$
	r_{safety}	$0.06\,m$	$0.06\,m$	$0.06\,m$

TABLE I: Parameters used in simulation experiments and hardware experiments.

To evaluate SPC and its implementation, we defined four path-based scenarios (trajectories), as shown in Figure 3. The end points on the path (shown in red) are provided in a timed sequence as target location p_{tar}. There are four scenarios: without obstacles (Figure 3a), with 1 obstacle (Figure 3b), with 2 obstacles (Figure 3c), and with 13 obstacles (Figure 3d). Simulations were conducted with flocks of size $|D| = 4$, 9, 15, and 30. Using radius $r_H = 0.9m$, the neighborhood is defined by:

$$H_i = \{j \in D \setminus \{i\} \wedge \|p_i - p_j\| < r_H\} \quad (19)$$

To check SPC's robustness to different PLLCs, we experimented with two PLLCs with different step responses. PLLC B reaches its set-point for x- and y-dimensions in less than half the time of PLLC A, while overshooting by about 50 % more. The PLLCs behave very similarly in the z-dimension. Figure 4 show snapshots of the simulations. A video is provided in the Supplementary Material and available at: https://youtu.be/iUkaYrnZz9k.

1) Results: The analysis of the quality metrics for *collision avoidance*, *compactness*, and *obstacle clearance* show that our SPC-based approach successfully maintains a stable flock. In Figure 5, metrics are plotted over time for three representative simulations. Data from the prefix of an execution, when the drones move from random starting positions into flock formation, are omitted when computing the metrics.

2) Computational complexity: The computation time of Eq. (4) is $O(|Q| \cdot (|H_i| + |K|))$. $|H_i|$ is bounded by $|D|$ and also depends on r_H. Introducing a concept of neighborhood for obstacles can reduce the computation time.

3) Comparison with PFC: To compare SPC with [14], [15], we also experimented with a PFC controller based solely on gradients. In this controller, the gradient vector $\nabla_{p_i} c$ is used to determine the next set-point $x_{io}^{(r)}$ for the PLLC as follows:

$$x_{io}^{(r)} = p_i - k \cdot \nabla_{p_i} c(p_i) \quad (20)$$

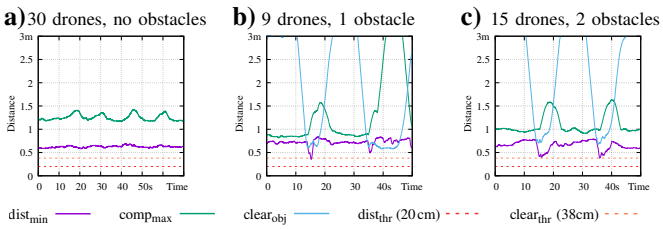

Fig. 5: Quality metrics over time for SPC simulations using PLLC B for exemplary scenarios of **a**: 30 drones with 0 obstacles, **b**: 9 drones with 1 obstacle, and **c**: 15 drones with 2 obstacles. Results for other simulation experiments were very similar. While the flock is passing the obstacle(s) the metrics temporarily degrade, however values $dist_{min}$ and $clear_{obj}$ stay above the respective thresholds, meaning there are no collisions, throughout the whole simulation. Analogously $comp_{max}$ stays below the threshold (5m), indicating that a compact flock is continuously maintained.

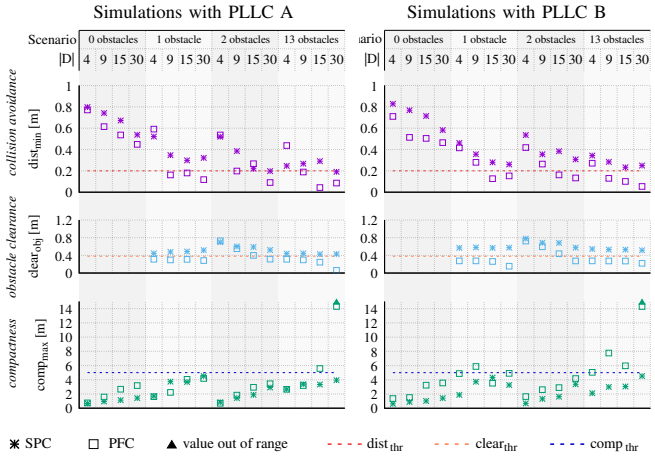

Fig. 6: Performance comparison for SPC and PFC. Values are min (for collision avoidance and obstacle clearance) or max (for compactness) over the simulation duration. SPC satisfies $dist_{thr}$ in nearly every scenario, while PFC frequently violates it, especially in the presence of obstacles. SPC maintains $comp_{thr}$ in every scenario, while for PFC, $comp_{max}$ sometimes gets very high, even out of range.

The control law stated in [14] provides an acceleration vector, which we adapted in Eq. (20) to a positional variant as required by the PLLC. We determined the gain k empirically such that the target of the flock was reached within the same time as our SPC implementation. The gain determines how aggressively the controller moves the drone toward the target location. In the experiments detailed below, the gain is constant, as in [14], [15]. We also briefly experimented with dynamic gain, where k is computed using a function similar to the one in Eq. (18). This had relatively small effects. Compared to the results with static gain reported below: collision avoidance improved slightly for some scenarios; obstacle clearance improved for some cases with PLLC A, while it worsened with PLLC B; and compactness improved moderately.

Figure 6 shows performance metrics for simulations of SPC and PFC controllers. While both perform reasonably well without obstacles, SPC's performance is superior in the presence of obstacles. This validates our hypothesis that SPC is particularly valuable when the cost function is more nonlinear (adding obstacles has that effect). Whenever a drone enters or leaves another drone's neighborhood, the cost function instantaneously changes its value; the gradient changes too. This causes the PFC controller to fail: in these simulations, we observed oscillating behavior and multiple collisions. SPC successfully deals with all of these situations. *In short, SPC is more robust to nonlinearities in the cost function and differences in the behavior of the PLLC.*

B. Hardware Experiments

We also experimented with real drones, specifically, *Crazyflie 2.1*-quadcopters [7]; see Figure 7a. For localization, we used the *Loco-Positioning* system [19]. The drones seamlessly integrated with the localization system, resulting in a (internal) PLLC that enables a drone to hold its position at a given set-point. Stability, however, depends on both the accuracy of the localization system and on the mechanical limitations of the drone. When hovering at a given set-point, we observed noise in the drone's position in the range of 15 cm. This was also noted in [20].

In the hardware implementation, we used ROS with the same software node as in Section IV-A, with only minor parameter modifications. This demonstrates the robustness of SPC with respect to a potential sim-to-real transfer gap. Since Crazyflies are incapable of running ROS on-board, we transmit the position updates to a PC that runs the controller and transmits the set-point position to the drone. Our experiments therefore also show that SPC is resilient to the additional delay introduced by radio transmission of position updates and set-point messages. Our controller, however, could be ported to run directly on ROS-capable drones, since we run it separately for each drone.

For the hardware experiments we used the same scenarios, as in the simulation experiments (Figure 3), except with 13 obstacles. Flocks of size $|D| = 2$, 4, 9, and 16 were used.

1) Results: Figures 7b and 7c show pictures of our experiments in a lecture hall. A video is provided in the Supplementary Materials. To show the drone movements for one example experiment with 16 drones, the recorded traces of the localization system are plotted in Figures 7d, 7e, and 7f.

Figure 8 presents performance metrics for our hardware experiments. Data from the prefix of an experiment, when the drones move from initial starting positions into flock formation, are omitted when computing the metrics. Figure 8 shows that our SPC-based approach successfully maintains a stable flock of Crazyflie drones by satisfying thresholds for *collision avoidance*, *compactness*, and *obstacle clearance* in nearly every scenario for the full duration of the experiment.

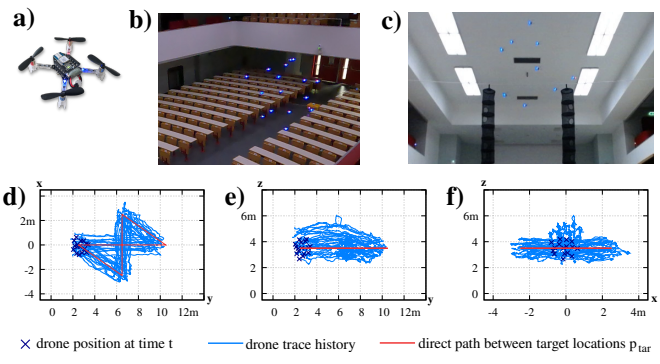

Fig. 7: Hardware experiments. **a**: Crazyflie 2.1 quadcopters were used. **b**: A flock of 16 drones and **c**: 9 drones with 2 obstacles in our lecture hall (a video is in the Supplementary Materials). **d**, **e**, **f**: Recorded traces show the movements of the 16 drones for one exemplary experiment.

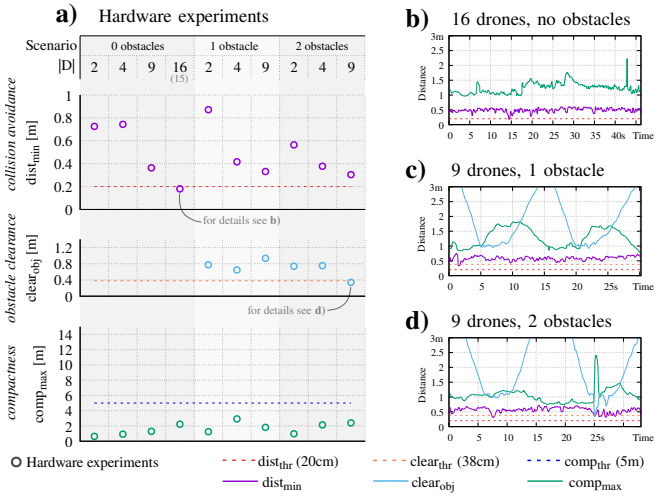

Fig. 8: Performance metrics for hardware experiments. **a**: Values are min (for collision avoidance and obstacle clearance) or max (for compactness) over the experiment. Experiments show that the flock is properly maintained: $dist_{thr}$ is satisfied in every scenario but one (see transient violation at $t = 15s$ in **b**). Similarly for $clear_{thr}$ (see transient violation at $t = 25s$ in **d**). **b**, **c**, **d**: Metrics for the whole duration of the hardware experiment for selected scenarios.

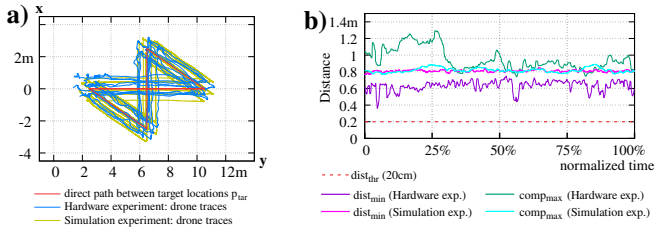

Fig. 9: Comparison of simulation and hardware experiments for 9 drones without obstacles. **a**: Recorded traces show the movements of the drones. **b**: Metrics for the entire duration of the experiment. The hardware-experiment metrics are a bit more noisy. This also explains why the hardware-experiments metrics in Fig. 8 are slightly worse.

Detailed plots for some critical scenarios are shown in Figure 8b, and Figure 8d. There are transient violations of the metrics, which are likely caused by measurement issues in the localization system; our controller, however, is able to promptly re-establish proper operation. Figure 9 provides a comparison of simulation and hardware experiments. It establishes that the controller performance metrics are slightly worse and noisier.

V. RELATED WORK

SPC can be viewed as combining features of MPC and PFC. MPC does a lookahead in time to decide the best control action. It requires a model of the system to compute states at future time points. Intuitively, MPC computes all the states that can be reached in k time steps using different control inputs, picks the best feasible trajectory, and returns the associated control action. In contrast, SPC ignores the system model and feasibility altogether and instead searches for good target states by enumerating promising candidates. Both MPC and SPC recompute their action in each time step using an optimization procedure to handle noise and variability in environment. PFC uses the gradient of the cost to pick the next action, just like SPC, but PFC does not perform any optimization.

Reynolds [21] was the first to propose a flocking model, using cohesion, separation, and velocity alignment force terms to compute agent accelerations. Reynolds model was extensively studied [22] and adapted for different application areas [23]. Alternative flocking models are considered in [24]–[28], and [14]. Other formulations consider swarm control in the context of formation rigidity [29]–[31]. In these approaches, flocks are described using point models. This means that physical properties of agents (e.g., drones) such as mass and inertia, are not taken into account. In our work, we evaluate SPC on a realistic physical drone model, as well as on real hardware.

In addition to these largely theoretical approaches, in [32]–[34], flocking controllers are implemented and tested on real hardware. However, the approach of [33], [34] involve the use of model-predictive control, which is computationally more expensive than SPC. In contrast to SPC, [32] requires the velocity of neighboring drones. Gradient optimization for robot control has been studied in [15]. In contrast, SPC uses spatial look-ahead as opposed to pure gradient descent.

VI. CONCLUSIONS

We introduced the concept of *Spatial Predictive Control* (SPC), and demonstrated its utility on the drone flocking problem. SPC is fully distributed. It is based only on the position of the individual drone itself, and on those of neighboring drones. This information is used to compute the gradient of the local cost function and to perform a spatial prediction for the best next action.

We performed an extensive experimental evaluation of SPC on the drone flocking problem. Our simulation experiments used a physics engine with a detailed drone model. Our results demonstrated SPC's ability to form and maintain a flock, avoid obstacles, and move the flock to multiple target locations. They also highlighted SPC's robustness to sensing noise and PLLC variability, and its role in the controller hierarchy.

We also evaluated the same controller implementation on a flock of Crazyflie 2.1 quadcopters in different scenarios, thereby demonstrating the effectiveness of SPC in controlling real hardware. Needing only a minor parameter adjustment, and no modifications to the control algorithm, SPC proved to be very robust in terms of a potential sim-to-real transfer gap. The hardware experiments also highlighted SPC's capability to perform properly in the presence of significant sensor noise introduced by the localization system and the extra latency introduced by radio transmission of positional and control signals.

We also experimentally compared SPC with a related PFC-based approach of [15]. We found that SPC exhibits superior performance and stability, as its discrete search for an optimal solution enables it to avoid oscillations. SPC is a general technique for designing *middle-level controllers* sandwiched between high-level planners and PLLCs that often come integrated with the hardware.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. R.G. was partially supported by EU-H2020 Adaptness and AT-BMBWF DK-RES and CPS/IoT Ecosystem. This work was supported in part by NSF grants CCF-1954837, CCF-1918225, and CPS-1446832.

REFERENCES

[1] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 837–855, 2018. DOI: 10.1109/TRO.2018.2857475.

[2] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Autonomous Robots*, vol. 30, no. 1, pp. 73–86, 2011. DOI: 10.1007/s10514-010-9205-0.

[3] G. Loianno and V. Kumar, "Cooperative transportation using small quadrotors using monocular vision and inertial sensing," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 680–687, 2018. DOI: 10.1109/LRA.2017.2778018.

[4] D. Câmara, "Cavalry to the rescue: Drones fleet to help rescuers operations over disasters scenarios," in *2014 IEEE Conference on Antenna Measurements Applications (CAMA)*, 2014, pp. 1–4. DOI: 10.1109/CAMA.2014.7003421.

[5] A. Boggio-Dandry and T. Soyata, "Perpetual flight for UAV drone swarms using continuous energy replenishment," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2018, pp. 478–484. DOI: 10.1109/UEMCON.2018.8796684.

[6] N. Michael, S. Shen, K. Mohta, *et al.*, "Collaborative mapping of an earthquake damaged building via ground and aerial robots," in *Proceedings of 8th International Conference on Field and Service Robotics (FSR '12)*, Jul. 2012, pp. 33–47.

[7] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017, pp. 37–42. DOI: 10.1109/MMAR.2017.8046794.

[8] SZ DJI Technology Co., Ltd. "dji developer: Missions." (2021), [Online]. Available: https://developer.dji.com/document/aab56894-31c7-4f38-b12d-616d312965a6.

[9] Parrot. "Parrot for developers: Autonomous flight." (2021), [Online]. Available: https://developer.parrot.com/docs/airsdk/general/autonomous_flight.html.

[10] Black-i Robotics. "Landshark UGV." (2021), [Online]. Available: https://www.blackirobotics.com/landshark-ugv/.

[11] Taurob Technologies. "The Taurob Inspector." (2021), [Online]. Available: https://taurob.com/wp-content/uploads/2020/08/Technical-Product-Sheet.pdf.

[12] General Dynamics. "Bluefin(R)-12 with Integrated Survey Package." (2021), [Online]. Available: https://gdmissionsystems.com/-/media/General-Dynamics/Maritime-and-Strategic-Systems/Bluefin/PDF/Bluefin-12-UUV-Datasheet.ashx.

[13] R. Negenborn and J. Maestre, "Distributed model predictive control: An overview and roadmap of future research opportunities," *IEEE Control Systems Magazine*, vol. 34, no. 4, pp. 87–97, 2014. DOI: 10.1109/MCS.2014.2320397.

[14] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stability of flocking motion," University of Pennsylvania, Tech. Rep., 2003. [Online]. Available: https://www.georgejpappas.org/papers/boids03.pdf.

[15] M. Schwager, "A gradient optimization approach to adaptive multi-robot control," Ph.D. dissertation, Massachusetts Institute of Technology, 2009. [Online]. Available: https://dspace.mit.edu/handle/1721.1/55256.

[16] G. Silano, E. Aucone, and L. Iannelli, "CrazyS: A software-in-the-loop platform for the Crazyflie 2.0 nano-quadcopter," in *2018 26th Mediterranean Conference on Control and Automation (MED)*, 2018, pp. 1–6. DOI: 10.1109/MED.2018.8442759.

[17] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[18] Stanford Artificial Intelligence Laboratory et al., *Robotic operating system*, version ROS Melodic Morenia, May 23, 2018. [Online]. Available: https://www.ros.org.

[19] Bitcraze, *Loco positioning system*, 2021. [Online]. Available: https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/.

[20] J. P. Queralta, C. Martínez Almansa, F. Schiano, D. Floreano, and T. Westerlund, "UWB-based system for UAV localization in GNSS-denied environments: Characterization and dataset," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4521–4528. DOI: 10.1109/IROS45743.2020.9341042.

[21] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87, New York, NY, USA: Association for Computing Machinery, 1987, pp. 25–34, ISBN: 0897912276. DOI: 10.1145/37401.37406.

[22] J. Eversham and V. F. Ruiz, "Parameter analysis of Reynolds flocking model," in *2010 IEEE 9th International Conference on Cyberntic Intelligent Systems*, 2010, pp. 1–7. DOI: 10.1109/UKRICIS.2010.5898089.

[23] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 837–855, 2018. DOI: 10.1109/TRO.2018.2857475.

[24] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006. DOI: 10.1109/TAC.2005.864190.

[25] U. Mehmood, N. Paoletti, D. Phan, *et al.*, "Declarative vs rule-based control for flocking dynamics," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18, Pau, France: Association for Computing Machinery, 2018, pp. 816–823, ISBN: 9781450351911. DOI: 10.1145/3167132.3167222.

[26] S. Martin, A. Girard, A. Fazeli, and A. Jadbabaie, "Multiagent flocking under general communication rule," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 2, pp. 155–166, 2014. DOI: 10.1109/TCNS.2014.2316994.

[27] E. Soria, F. Schiano, and D. Floreano, "The influence of limited visual sensing on the Reynolds flocking algorithm," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 138–145. DOI: 10.1109/IRC.2019.00028.

[28] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1928–1935. DOI: 10.1109/ROBOT.2008.4543489.

[29] F. Schiano and P. R. Giordano, "Bearing rigidity maintenance for formations of quadrotor UAVs," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1467–1474. DOI: 10.1109/ICRA.2017.7989175.

[30] B. Pozzan, G. Michieletto, A. Cenedese, and D. Zelazo, "Heterogeneous formation control: A bearing rigidity approach," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 6451–6456. DOI: 10.1109/CDC45484.2021.9683374.

[31] D. Zelazo, A. Franchi, H. H. Bülthoff, and P. R. Giordano, "Decentralized rigidity maintenance control with range measurements for multi-robot systems," *The International Journal of Robotics Research*, vol. 34, no. 1, pp. 105–128, 2015. DOI: 10.1177/0278364914546173.

[32] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, 2018. DOI: 10.1126/scirobotics.aat3536.

[33] E. Soria, F. Schiano, and D. Floreano, "Predictive control of aerial swarms in cluttered environments," in *Nature Machine Intelligence*, 2021. DOI: 10.1038/s42256-021-00341-y.

[34] E. Soria, F. Schiano, and D. Floreano, "Distributed predictive drone swarms in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 73–80, 2022. DOI: 10.1109/LRA.2021.3118091.