2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Crazyswarm: A Large Nano-Quadcopter Swarm

James A. Preiss*, Wolfgang Hönig*, Gaurav S. Sukhatme, and Nora Ayanian

Abstract— **We define a system architecture for a large swarm of miniature quadcopters flying in dense formation indoors. The large number of small vehicles motivates novel design choices for state estimation and communication. For state estimation, we develop a method to reliably track many small rigid bodies with identical motion-capture marker arrangements. Our communication infrastructure uses compressed one-way data flow and supports a large number of vehicles per radio. We achieve reliable flight with accurate tracking ($<$ 2 cm mean position error) by implementing the majority of computation onboard, including sensor fusion, control, and some trajectory planning. We provide various examples and empirically determine latency and tracking performance for swarms with up to 49 vehicles.**

I. INTRODUCTION

Quadcopters are a popular robotic platform due to their agility, simplicity, and wide range of applications. Most research quadcopters are large enough to carry cameras and smartphone-grade computers, but they are also expensive and require a large space to operate safely. For very large swarms, smaller quadcopters are more attractive. The reduced size of these vehicles motivates different system design choices compared to a typical setup for larger vehicles in smaller numbers.

Here we describe the system architecture for a swarm of 49 very small quadcopters operating indoors. The vehicles use a motion-capture system for localization and communicate over three shared radios. Our system uses off-the-shelf hardware and performs most computation onboard. We publish the software as open-source[1], making our work easily reusable for other researchers. To our knowledge, the system described here is the largest indoor quadcopter swarm to date, and the largest number of quadcopters controlled per radio.

II. RELATED WORK

Unmanned Aerial Vehicle (UAV) swarms have been used indoors for formation flight and collaborative behaviors, outdoors to demonstrate swarming algorithms, and in the media for artistic shows.

Kushleyev *et al.* describe the design, planning, and control of a custom micro quadcopter with experiments involving up to 20 vehicles [1]. The group's infrastructure is described in [2]. While key aspects are similar, our work differs in the following ways: (a) our architecture is designed to reduce communication bandwidth and scales better to larger numbers of quadcopters; (b) we demonstrate interactivity with the swarm; and (c) we use a commercially available platform and provide our source code online.

The Flying Machine Arena at ETH Zurich supports both single- and multi-robot experiments. An overview of the system architecture and applications is given in [3]. Unlike our work, the position controller runs offboard, making the system less robust to packet drops. The additional computational power is used for a latency compensation algorithm to improve accuracy for high-speed flights.

UAVs have been used to demonstrate swarming algorithms outdoors on fixed wing [4], [5] and quadcopter [6] platforms with up to 50 vehicles. Our platform uses a smaller vehicle, allowing us to execute such experiments indoors in a laboratory environment.

Quadcopter swarms have been featured in the media as well. A world record for the most UAVs airborne simultaneously was set in 2015 with 100 quadcopters flying outdoors [7]. In 2016, a TED talk featured about 35 nano-quadcopters flying without a net above a crowd indoors using an ultra-wideband localization system [8]. However, not many technical details about both solutions are publicly available. Our solution requires a motion-capture system, allows dense formations, and shows good scalability.

Fig. 1. The Crazyflie 2.0 miniature quadcopter with four motion-capture markers, LED expansion board (not visible), and battery.

III. VEHICLE

The Crazyflie 2.0 quadcopter (Fig. 1). measures 92 millimeters between diagonally opposed motor shafts and weighs 27 grams with a battery. It contains a 32-bit, 168-MHz ARM microcontroller with floating-point unit that is capable of significant onboard computation. Software and hardware are both open-source. The Crazyflie communicates with a PC over the Crazyradio PA, a 2.4 GHz USB radio that transmits up to two megabits per second in 32-byte packets.

Our payload of motion-capture markers and an LED light expansion board brings the Crazyflie's mass to 33 grams.

*Equal contributors.

All authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA, USA.
Email: {japreiss, whoenig, gaurav, ayanian}@usc.edu

This work was partially supported by the ONR grants N00014-16-1-2907 and N00014-14-1-0734, and the ARL grant W911NF-14-D-0005.

[1] https://github.com/USC-ACTLab/crazyswarm

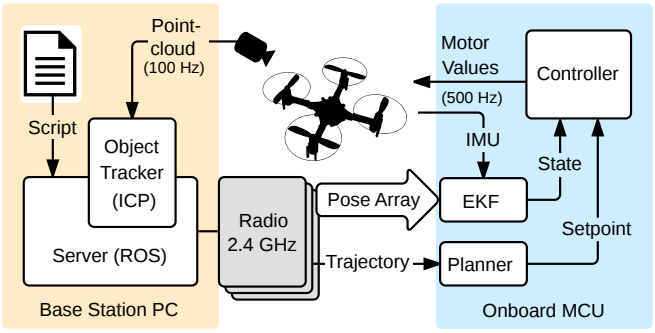

Fig. 2. Diagram of major system components. A point cloud of markers detected by a motion-capture system is used to track the quadcopters. All estimated poses are broadcasted using three radios. Planning, state estimation, and control run onboard each vehicle at 500 Hz.

This reduces battery life from the listed seven minutes to six minutes, and results in a peak thrust-to-weight ratio of $\sim$1.8. The system's dynamics and empirically determined parameters have been discussed in [9] and [10].

The Crazyflie's small size makes it suitable for indoor flight in dense formations. It can survive high-speed crashes due to its low inertia and poses little risk to humans.

IV. ARCHITECTURE OVERVIEW

Our system architecture is outlined in Fig. 2. We track the vehicles with a VICON motion-capture system using passive spherical markers. While the motion-capture system creates a single point of failure, we chose it over alternatives due to its high performance: typical position errors are less than one millimeter [3]. In comparison, a state-of-the-art decentralized localization system using ultra-wideband radio triangulation [11] showed position errors of over 10 centimeters, too large for dense formations. While vision-based methods are both accurate and decentralized [12], the required cameras and computers necessitate a much larger vehicle.

In contrast to related systems [2], [3], we implement the majority of in-flight computation onboard. The base station sends complete trajectory descriptions to the vehicle in the form of polynomials, ellipses, etc. For external feedback, the base station broadcasts vehicle poses using three radios operating on separate channels.

The main onboard loop runs at 500 Hz. In each loop cycle, the vehicle reads its Inertial Measurement Unit (IMU) and runs the state estimator, trajectory evaluator, and position controller. Messages with external pose estimates arrive asynchronously and are fused into the state estimate on the next cycle.

Since the full trajectory plan is stored onboard, the system is robust to significant radio packet loss. If a packet is dropped, the vehicle relies on its IMU to update the state estimate. We quantify our system's stability against simulated packet loss in Section XI-C.

V. OBJECT TRACKING

Standard rigid-body motion-capture software such as `Vicon Tracker` requires a unique marker arrangement for each tracked object. The Crazyflie's small size limits the number of locations to place a marker, making it impossible to form 49 unique arrangements that can be reliably distinguished. Therefore, we obtain only raw point clouds from the motion-capture system, and implement our own object tracker based on the *Iterative Closest Point* (ICP) algorithm [13] that handles identical marker arrangements. Our method is initialized with known positions, and subsequently updates the positions with frame-by-frame tracking.

A. Initialization

With identical marker arrangements for each vehicle, the object tracker needs some additional source of information to establish the mapping between vehicle identities (radio addresses) and spatial locations. We currently supply this information with a configuration file containing a fixed initial location for each vehicle. However, it is not feasible to place each vehicle at its exact configured position before every flight. To allow for small deviations, we perform a nearest-neighbor search within a layout-dependent radius about the initial position, and try ICP with many different initial guesses for vehicle yaw. We accept the best guess only if its resulting alignment error is low (less than 1 mm mean squared Euclidean distance between aligned points).

In future work we plan to eliminate the configuration file, using either infrared LEDs or programmed small motions for vehicle self-identification via the motion-capture system.

B. Frame-to-frame Tracking

Each frame, we acquire a raw point cloud from the motion-capture system. For each object, we use ICP to register the marker positions corresponding to the object's last known pose against the entire scene point cloud. This process is independent for each object, so it can be executed in parallel. This approach assumes that there are no prolonged occlusions during the duration of the flight. We limit ICP to five iterations, which allows operation at 75 Hz with 49 robots using our computer hardware.

Motion-capture systems sometimes deliver a point cloud with spurious or missing points; if undetected, this may cause tracking errors. To mitigate these errors, we compute linear and angular velocities from the ICP alignments and reject physically implausible values as incorrect alignments. In combination with our on-board state estimation, a few missing frames do not cause significant instabilities, making tracking reliable in practice.

VI. STATE ESTIMATION

To reduce communication bandwidth requirements and maintain robustness against temporary communication loss, we fuse motion-capture and IMU measurements onboard in an Extended Kalman Filter (EKF). The filter is driven by IMU measurements at 500 Hz and estimates the states $(\mathbf{p}, \mathbf{v}, \mathbf{q})$ where $\mathbf{p} \in \mathbb{R}^3$ is the vehicle's position, $\mathbf{v} \in \mathbb{R}^3$ is its velocity, and $\mathbf{q} \in S^3$ is the unit quaternion transforming the vehicle's local coordinate frame into world coordinates.

The system inputs are the accelerometer and gyroscope measurements, $\mathbf{a}_m$ and ω_m, respectively. The dynamics are:

$$\dot{\mathbf{p}} = \mathbf{v}, \quad \dot{\mathbf{v}} = \mathbf{q} \odot \mathbf{a}_m - \mathbf{g}, \quad \dot{\mathbf{q}} = \tfrac{1}{2}\Omega(\omega_m)\mathbf{q}, \qquad (1)$$

where $\mathbf{g}$ is the gravity vector in world coordinates, $\Omega(\omega_m)$ is the quaternion multiplication matrix of ω_m as defined in [14], and $\odot$ denotes quaternion-vector rotation. The motion-capture system directly measures $\mathbf{p}$ and $\mathbf{q}$, resulting in a trivial measurement model.

Our EKF implementation follows the indirect error-state approach [14], [15], where IMU measurements drive a dynamics integration and the filter estimates the error and covariance of this integration. This method avoids the problematic extra dimension when representing a 3-DOF unit quaternion by four numbers. Whereas [14], [15] estimate accelerometer and gyroscope bias in the EKF, we have found that these biases do not drift significantly during the Crazyflie's battery life. During several six-minute test flights we found the maximum drift of any axis of the accelerometer to be $0.07\,{}^{\mathrm{m}}/_{\mathrm{s}^2}$ and the maximum drift of any axis of the gyroscope to be $0.0018\,{}^{\mathrm{rad}}/_{\mathrm{s}}$. Therefore, we measure biases on a level floor at startup and subtract these measurements for the duration of the flight. We also omit higher-order dynamics approximation terms, as we have found their effect unmeasurable in experiments.

VII. PLANNING

We select a set of onboard trajectory planning methods that require a small amount of information exchange between the base station PC and the vehicles. Compared to architectures that evaluate trajectories on a PC and transmit attitude and thrust controls at a high rate [2], [3], we shift some planning effort onboard to reduce the needed radio bandwidth.

Quadcopter dynamics are differentially flat in the outputs $\mathbf{y} = (\mathbf{p}, \psi)$, where ψ is the yaw angle in world coordinates [16]. This means that the controls required to execute a trajectory in state space are functions of $\mathbf{y}$ and a finite number of its time derivatives. We take advantage of this fact and plan trajectories in $\mathbf{y}$ using functions that are at least four times differentiable.

A. Piecewise Polynomials

Piecewise polynomials are widely used to represent quadcopter trajectories. It is straightforward to construct a piecewise polynomial that satisfies given waypoint and continuity constraints. For a k-dimensional, q-piece, d-degree polynomial, the $k \cdot q \cdot (d+1)$ polynomial coefficients are concatenated into a single row vector $\mathbf{c}$. Waypoint and continuity constraints are then expressed as linear systems:

$$\mathbf{c}\mathbf{T}_w = \mathbf{w}, \quad \mathbf{c}\mathbf{T}_c = \mathbf{0}, \qquad (2)$$

where $\mathbf{w}$ is a concatenated vector of waypoint values in both $\mathbf{y}$ and its derivatives, and $\mathbf{T}_w$ and $\mathbf{T}_c$ are both simple banded matrices obtained from the time values of the piece breaks. Constructions of $\mathbf{T}_w$, $\mathbf{T}_c$, and $\mathbf{w}$ are detailed in [17].

With a suitably high polynomial degree, the combined system (2) is underdetermined. The left null space of the system matrix thus provides a space for optimizing the trajectory for objectives such as min-risk [18] or observability [17] while implicitly satisfying the waypoint and continuity constraints. Additional constraints, such as thrust or angular velocity limits, are expressed as nonlinear inequalities in a general-purpose nonlinear optimization solver.

Our framework supports piecewise polynomials uploaded from the base station or built into the firmware image.

B. Online Single-Piece Polynomials

For short trajectories, a single polynomial can be sufficient. We implement an online single-piece polynomial planner for tasks such as vertical takeoff/landing and linear movement between hover points. This planner computes a closed-form solution for a degree-7 polynomial starting at the current state in $(\mathbf{p}, \mathbf{v}, \ddot{\mathbf{p}}, \psi, \dot{\psi})$ and ending at a desired state in the same variables. For linear movements, we restrict $\dddot{\mathbf{p}} = \mathbf{0}$ and $\ddot{\psi} = \dddot{\psi} = 0$, which results in a closed-form solution for the polynomial coefficients.

C. Ellipses

Elliptical motion is useful for demonstrations because it can generate a range of visually appealing behaviors from few parameters. Ellipses are parameterized by their center, axes, period, and phase, and are infinitely differentiable, so the differentially flat transformation applies.

Commanding a quadcopter to switch from hovering to elliptical motion produces a large step change in the controller setpoint, potentially causing instability. Our system overcomes this issue by using the single-piece polynomial planner to plan a trajectory that smoothly accelerates into the ellipse, iteratively replanning with longer time horizons until it achieves a trajectory that respects the vehicle's dynamic limits. The procedure is general and can be used to plan a smooth start from hover for any periodic trajectory.

D. Interactive Avoid-Obstacle Mode

The planners discussed so far can follow predefined paths, but are not suitable for dynamically changing environments. For the case of a single, moving obstacle at a known location, such as a human, we use a specialized avoid-obstacle mode. The planner is fully distributed and only needs to know the quadcopter's position $\mathbf{p}$, its assigned home position $\mathbf{p}_{home}$, and the obstacle's position $\mathbf{p}_{obst}$. Let $\mathbf{d}$ be the vector between the current position and the obstacle to avoid:

$$\mathbf{d} = \mathbf{p} - \mathbf{p}_{obst}, \ \delta = \|\mathbf{d}\|_2. \qquad (3)$$

The new desired position $\mathbf{p}_{des}$ can be computed as a weighted displacement from the home position, where we move further away if the obstacle is close:

$$\mathbf{p}_{des} = \mathbf{p}_{home} + \max\left(\frac{f}{\delta + \delta^2}, \delta_{max}\right)\frac{\mathbf{d}}{\delta}, \qquad (4)$$

where f is a scalar gain and δ_{max} limits displacment, preventing collisions when appropriately small relative to the robot spacing. This is similar to a potential field approach. The attraction to the home position is encoded in (4) by

TABLE I
CONSECUTIVE PACKETS DROPPED (SEE TEXT FOR DETAILS.)

Delay	0	1	2	3	4	5+
3 ms	54458	2	280	1174	0	0
10 ms	51755	4111	4	1	0	0

displacing from $\mathbf{p}_{home}$ rather than $\mathbf{p}$. Instead of using the computed $\mathbf{p}_{des}$ directly as control input, we compute a smooth trajectory with bounded velocity and acceleration.

VIII. COMMUNICATION

We use two different kinds of communication: request-response and broadcasting. Request-response is primarily used for configuration while the Crazyflie is still on the ground. This includes uploading a trajectory, changing flight parameters such as controller gains, and assigning each Crazyflie to a group. Broadcasting is used during the flight to minimize latency for position feedback, and to achieve synchronized behavior for taking off, landing, starting a trajectory, etc. Broadcast commands can be restricted to subsets of the swarm by including a group ID number.

Our communication does not use a standard transport layer and hence, we need to handle sporadic packet drops as part of our protocol. In order to achieve low latency, we do not aim for guaranteed packet delivery, but rather for a high probability of reliable communication. In our protocol, all commands are idempotent, so they can be received multiple times without any side effects. This allows us to repeat request-response commands until acknowledged or until a timeout occurs. Swarm-coordination commands, such as taking off, do not wait for an acknowledgement but are repeated several times for a high probability that all Crazyflies receive the command. Since external pose estimates are send at a high fixed rate every 10 ms, there is no need to explicitly repeat such messages.

Empirically, for repeated commands, the likelihood of packet loss depends on the rate at which the command is repeated. We show this effect in Table I. We count the number of consecutive packets dropped from 0 (no packet dropped) to 5+ (5 or more consecutive packets dropped) for 10000 packets sent to six Crazyflies. The results show that repeating messages 5 times every 3 ms for swarm coordination commands is likely to reach all Crazyflies. Furthermore, it is very unlikely to drop more than five consecutive position updates, which is sufficient for stable control.

For 49 vehicles we use three radios, with each vehicle permanently assigned to one radio. We broadcast positions $\mathbf{p}$ as 24-bit fixed-point numbers and compress quaternions $\mathbf{q}$ into 32 bits by transmitting the smallest three values in reduced precision and reconstructing the largest value from the unit quaternion property $\|\mathbf{q}\|_2 = 1$. Compression allows us to fit two position updates in one 32-byte radio packet without degrading the measurements beyond their inherent noise level. Furthermore, we transmit two such radio packets per USB request to the Crazyradio PA, allowing us to broadcast the pose of up to four Crazyflies per USB request.

IX. CONTROL

Our controller is based on the nonlinear position controller of [16], augmented with integral terms for position and yaw error. The control input is the current state $(\mathbf{p}, \mathbf{v}, \mathbf{q}, \omega_m)$ and the setpoint in position $\mathbf{p}_{des}$, velocity $\mathbf{v}_{des}$, acceleration $\ddot{\mathbf{p}}_{des}$, yaw ψ_{des}, and angular velocity ω_{des}.

The controller is a cascaded design with an outer loop for position and an inner loop for attitude. Both loops implement PID feedback terms; the outer position loop adds feedforward terms from the trajectory plan. We define position and velocity errors as

$$\mathbf{e}_p = \mathbf{p}_{des} - \mathbf{p}, \quad \mathbf{e}_v = \mathbf{v}_{des} - \mathbf{v}$$

respectively, and compute the desired force vector as follows:

$$\mathbf{F}_{des} = K_p\mathbf{e}_p + K_i \int_0^t \mathbf{e}_p dt + K_v\mathbf{e}_v + m\mathbf{g} + m\ddot{\mathbf{p}}_{des}, \quad (5)$$

where K_p, K_i, and K_v are positive diagonal gain matrices.

The desired body rotation matrix R_{des} is a function of $\mathbf{F}_{des}$ and ψ_{des}. The orientation error is computed as

$$\mathbf{e}_o = \frac{1}{2}\left(R^T R_{des} - R_{des}^T R\right)^\vee,$$

where R is the matrix form of the attitude quaternion $\mathbf{q}$ and $(\cdot)^\vee$ is the vee operator mapping $SO(3) \to \mathbb{R}^3$. The desired moments are then computed with another PID controller:

$$\mathbf{M}_{des} = K_o\mathbf{e}_o + K_m \int_0^t \mathbf{e}_o dt + K_\omega(\omega_{des} - \omega) \quad (6)$$

with positive diagonal gain matrices K_o, K_m, and K_ω. A simple linear transformation can compute the desired squared rotor speeds from $\mathbf{M}_{des}$ and the projection of $\mathbf{F}_{des}$ to the body z axis, as shown in [16].

This controller is identical to the one presented in [16] except for the added integral terms. The position error integral in (5) compensates for battery voltage drop over time (z-part) and unbalanced center of mass due to asymmetries (x, y-parts). From the attitude error integral in (6), we only use the z-component, which compensates for inaccuracies in yaw due to uneven wear on propellers and motors. Manual trimming and part replacement can compensate for such issues on a single vehicle, but is not feasible on a large fleet.

X. SOFTWARE TOOLS

Many routine tasks become non-trivial when working with 49 robots. We have developed command-line tools for mass rebooting, firmware updates, firmware version query, and battery voltage checks over the radio. A power-save mode turns off the main microcontroller and LED ring while leaving the radio active, permitting powering on the Crazyflie remotely. We can execute such commands and disable or enable subsets of the swarm using a graphical user interface. A Python scripting layer supports development of complex multi-stage formation flight plans.

Our decision to perform significant computation onboard complicates in-flight debugging due to constrained radio telemetry bandwidth and limited permanent storage for logs.

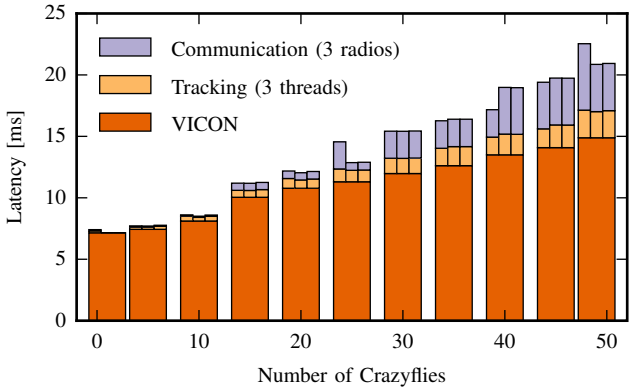

Fig. 3. Software-estimated latency for different swarm sizes, averaged over 2000 measurements. The actual latency is 3 ms higher in all cases. The latency grows linearly with the swarm size and the largest portion is due to the motion-capture system.

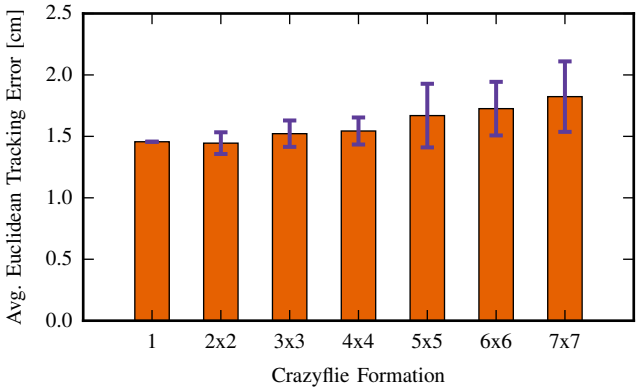

Fig. 4. Measured tracking performance for different swarm sizes, averaged over the swarm and duration of the flight. The error bars show the standard deviation. The performance decreases in bigger swarms because of the additional aerodynamic effects between the quadcopters.

To ease this problem, we structure major onboard procedures as platform-independent modules. We use the `SWIG` package to generate Python bindings to these modules, allowing firmware-in-the-loop testing in simulation via Python scripts.

XI. EXPERIMENTS

Our experiments quantify system performance and demonstrate the overall capabilities of our architecture. All experiments are conducted in a $6\,\text{m} \times 6\,\text{m} \times 3\,\text{m}$ motion-capture space using a VICON Vantage system with 24 cameras. `Vicon Tracker` obtains the marker point clouds on a PC with Windows 7, Xeon E5-2630 2.2 GHz, and 16 GB RAM. Our software is written in C++ using ROS Kinetic for visualization and scripting and is running on a PC with Ubuntu 16.04, Xeon E5-2630 2.2 GHz, and 32 GB RAM. To reduce latency, we do not use any ROS messages in the critical path between receiving data from the motion-capture system and broadcasting estimated poses to the Crazyflies. Sample flight videos are part of the supplemental material.

A. Latency

The total time delay between physical movement and arrival of the corresponding position feedback message to the firmware significantly affects overall system performance [3]. Latency increases with swarm size due to both computational and communication overhead. We estimate the latency in two ways: first, we use the `Vicon DataStream SDK` to query an estimate of the motion-capture system latency, and instrument our code to estimate the runtime of our own software components. Second, we verify those numbers in the physical setup by inducing a sudden change in yaw to the Crazyflie. The delay between the immediate onboard IMU response and the corresponding change in external position measurement captures the full system latency. We read these values in real time via a JTAG debug adapter.

The estimated latencies from the base station software are shown in Fig. 3. The latency grows roughly linearly with the number of objects to track, with the largest portion caused by the motion-capture system (up to 14 ms for the 49-robot case). Object tracking and communication using the Crazyradio is done in separate threads, reducing the overall runtime. The communication latency increases in increments of four added Crazyflies per radio because of our compression algorithm (see section VIII.) Estimated total latency ranges from 8 ms to 23 ms.

The physical measurements show that the actual latency is a fixed 3 ms longer than the estimated numbers, resulting in an actual latency of 26 ms for the full 49-vehicle swarm.

B. Tracking performance

We estimate the tracking performance by integrating the overall Euclidean position error for a flight, excluding takeoff and landing. The flight path for each Crazyflie is a figure-8 repeated three times, with 0.76 $^{\text{m}}/_{\text{s}}$ average speed and 1.5 $^{\text{m}}/_{\text{s}}$ maximum speed. This trajectory is moderately fast for a small quadcopter, reaching a maximum roll angle of 20 deg. We try different square swarm sizes, ranging from 1 to 7×7, in a grid layout with 0.5 m spacing, flying at the same height. This tight formation causes additional aerodynamic disturbances, which affect the controller performance.

Our average error is shown in Fig. 4. The error is below 2 cm per quadcopter even when the whole swarm is flying. However, as visible in the supplemental video, the attitude controller starts to oscillate more to compensate for the additional airflow.

C. Effect of Position Update Rate on Hover Stability

We quantify the effect of position update rate on stability by varying the update rate from the maximal 100 Hz down to 2 Hz and measuring the mean Euclidean position error at hover state. Results are shown in Table II. The range between 100 Hz and 10 Hz shows virtually no change. At 2 Hz, the system is still stable but shows significant oscillations. This demonstrates our system's robustness against communication loss during operation.

TABLE II
HOVER POSITION ERROR AS A FUNCTION OF POSITION UPDATE RATE.

Position Rate [Hz]	100	10	5	$3.\bar{3}$	2.5	2
Avg. Tracking Error [cm]	0.58	0.68	0.91	1.48	1.95	2.18

Fig. 5. Forty-nine Crazyflies flying in a 4-layer rotating pyramid formation. The bottom layer is 3 m × 3 m with 0.5 m spacing between vehicles.

D. Coordinated Formation Flight

To demonstrate our scripting layer, we use a formation of 49 Crazyflies flying in a rotating pyramid as shown in Fig 5. The Crazyflies are initially placed in a 7×7 grid with 0.5 m spacing. The takeoff is done in four different layers, with the user triggering the next action using a joystick or keyboard. The script initially assigns each Crazyflie a group number, according to its layer, and uploads the parameters for its ellipse to execute later. Because of the grouping, the takeoff of a whole layer can be achieved at the same time with broadcast messages. After all Crazyflies are in the air, the online planner plans a trajectory to smoothly enter the ellipse. The user can decide when to end the rotation, which causes another replanning onboard to reach a hover state above the initial takeoff location. Finally, landing is done using the same groups, minimizing aerodynamic disturbances during landing.

E. Swarm Interaction

A human operator is holding (or wearing) an object with motion-capture markers. We track the position of that object and broadcast its position alongside the positions of all quadcopters over the radio. Using the onboard planner described in section VII-D, each Crazyflie computes its desired position, which avoids collisions with the obstacle or other quadcopters.

XII. CONCLUSIONS

We have described a system architecture for robust, synchronized, dynamic control of the largest indoor quadcopter swarm to date. Our system fully utilizes the vehicles' onboard computation, allowing for robustness against unreliable communication and a rich set of trajectory planning methods requiring little radio bandwidth. Tests show good scalability for both latency and tracking performance with respect to the swarm size. For a swarm of 49 vehicles, only 3 radios are required and we obtain a latency below 30 ms, allowing moderately aggressive flight maneuvers using onboard state estimation with mean tracking errors below 2 cm. Our full source code, including tuned parameters such as EKF variances and controller gains, is available at `https://github.com/USC-ACTLab/crazyswarm`.

In future work, we plan to improve reliability by detecting controller failures, improving object tracking failure recovery, and including better planning methods which take aerodynamic effects such as downwash into account explicitly. We have made all software components available as open-source, and hope that our work will have a significant impact as a testbed for experimental verification of existing and new UAV swarming algorithms. In general, our system can serve as a foundation for a wide range of future work in multi-robot planning, coordination, and control.

REFERENCES

[1] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[2] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[3] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

[4] S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J. Zufferey, and D. Floreano, "Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 5015–5020.

[5] T. H. Chung, M. R. Clement, M. A. Day, K. D. Jones, D. Davis, and M. Jones, "Live-fly, large-scale field experimentation for large numbers of fixed-wing UAVs," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 1255–1262.

[6] G. Vásárhelyi, C. Virágh, G. Somorjai, N. Tarcai, T. Szörényi, T. Nepusz, and T. Vicsek, "Outdoor flocking and formation flight with autonomous aerial robots," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014, pp. 3866–3873.

[7] Intel and Ars Electronica Futurelab, "Drone 100," world record for the most UAVs airborne simultaneously, http://www.spaxels.at/show_drone100.html, 2015.

[8] R. D'Andrea, "Meet the dazzling flying machines of the future," TED Talk, 2016.

[9] B. Landry, "Planning and control for quadrotor flight through cluttered environments," Master's thesis, MIT, 2015.

[10] J. Förster, "System identification of the crazyflie 2.0 nano quadrocopter," Bachelor's Thesis, ETH Zurich, 2015.

[11] A. Ledergerber, M. Hamer, and R. D'Andrea, "A robot self-localization system using one-way ultra-wideband communication," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 3131–3137.

[12] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 1, 2015.

[13] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, 1992.

[14] N. Trawny and S. I. Roumeliotis, "Indirect Kalman filter for 3D attitude estimation," University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep. 2005-002, 2005.

[15] S. Weiss and R. Siegwart, "Real-time metric state estimation for modular vision-inertial systems," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 4531–4537.

[16] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[17] K. Hausman, J. Preiss, G. S. Sukhatme, and S. Weiss, "Observability-aware trajectory optimization for self-calibration with application to UAVs," *IEEE Robotics and Automation Letters*, 2017.

[18] J. Müller and G. S. Sukhatme, "Risk-aware trajectory generation with application to safe quadrotor landing," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014, pp. 3642–3648.