EPFL

Laboratory of Intelligent Systems

SEMESTER PROJECT

Title: Aerial Swarm Flocking Control through Hand Motion

Student(s): Nevò Mirzai Hamadani (Robotics)

Professor: Dario Floreano

Assistant 1: Benjamin Jarvis

Project description:

Problem

We are developing aerial swarms that can be easily flown by a single human. A swarm is a complex system with many degrees of freedom that are hard to control with conventional interfaces. We want to leverage the many degrees of freedom of a pair of hands to coordinate the flocking behaviour of an aerial swarm.

Goal

The goal of this project is to create and test a mapping between hand motion and the internal and external degrees of freedom of a swarm. Existing research has shown that hand motion is a good method for controlling the motion of a swarm [1], but we want to explore whether it can also be used to control the shape of the swarm.

Method

1. Literature review on existing methods/similar problems
2. Use a Meta Quest Pro and a Unity simulation to capture hand pose information
3. Develop a mapping between hand pose and the swarm shape
4. Use this mapping to control a simulated swarm
5. Demonstrate this mapping on Crazyflie drones.

[1] M. Macchini, IEEE RAL, 2021

Remarks:

You should present a research plan (Gantt chart) to your first assistant before the end of the second week of the project. An intermediate presentation of your project, containing 8 minutes of presentation and 7 minutes of discussion, will be held on 7 April 2025. The goal of this presentation is to briefly summarize the work done so far and discuss a precise plan for the remaining time of the project. Your final report should start by the original project description (this page) followed by a one page summary of your work. This summary (single sided A4), should contain the date, laboratory name, project title and type (semester project or master project) followed by the description of the project and 1 or 2 representative figures. In the report, importance will be given to the description of the experiments and to the obtained results. A preliminary version of your report should be given to your first assistant at the latest 10 days before the final hand-in deadline. A PDF (dated & signed) of your final version should be
Sent to your first assistant and to the administrative assistant of the lab before noon 5 June 2025 before noon. Failing this deadline will reflect in the final grade. A 20 minute project defense, including 5 minutes for discussion, will take place 12 June 2025. You will be graded based on your results, report, final defense, and working style. All documents, including the report (source and pdf), summary page and presentations along with the source of your programs should be handed-in as a single compressed file on the day of the final defense at the latest.

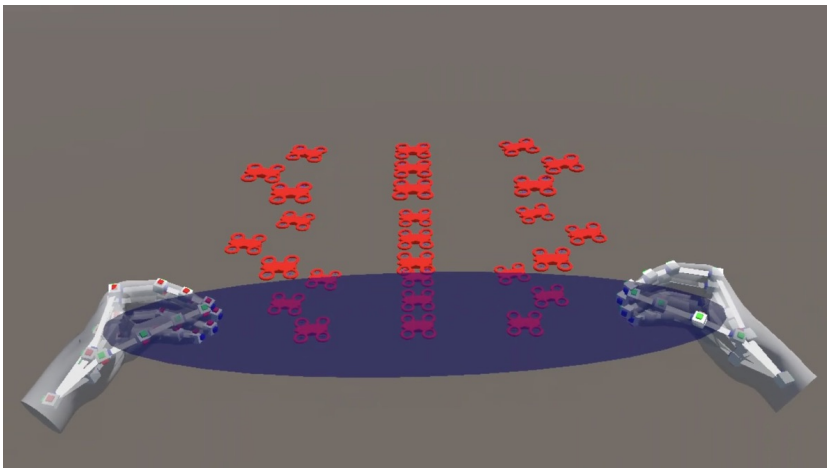
Responsible professor:	Responsible assistant:
Signature:  Dario Floreano	Signature:  Benjamin Jarvis

Lausanne, 3 February 2025

Date: 5/6/2025
Laboratory: LIS – Laboratory of Intelligent Systems

Aerial Swarm Flocking Control through Hand Motion

Semester Project



Description:

Aerial swarms hold significant potential for practical applications such as adaptive formation, area coverage, and cooperative task execution. However, their high-dimensional dynamics and the lack of axis-specific continuous control make them difficult and unintuitive to operate using traditional interfaces like joysticks or pre-programmed hand gestures. A single human operator with a standard controller often struggles to issue real-time, multi-axis commands that simultaneously adjust both the shape and motion of the swarm. As a result, many degrees of freedom remain underutilized, and essential maneuvers—such as compressing the swarm to pass through narrow gaps or reorienting its formation mid-flight—become challenging to execute fluidly.

The goal of this project is to leverage natural hand movements—such as the distance between the hands, hand openness, and hand orientation—to control both the shape and motion of an aerial swarm in real time. Instead of relying on discrete gestures or buttons to switch between predefined formations, the operator should be able to continuously stretch, rotate, and translate the swarm using intuitive hand motions. This allows the human pilot to maintain full spatial awareness and make real-time adjustments without the need to memorize a large set of specialized gestures.

To achieve this, a Meta Quest 3 headset is used in combination with a Unity simulation to capture hand-pose data in real time. These pose measurements are mapped onto a three-dimensional ellipsoid: the distance between the hands controls the ellipsoid's length, a measure of hand openness determines its thickness, and the relative orientation of the palms sets its diagonal elongation. A modified Olfati–Saber flocking controller uses these ellipsoid parameters in simulation, enabling axis-specific dilation and rotation of the swarm's cohesion frame. Once the virtual pipeline is validated, a UDP connection transmits swarm parameters—such as axis-specific separation and velocity vectors—to a ROS2 architecture, which generates the final control commands for a physical Crazyflie swarm. On real hardware, 2D dilation and velocity control are successfully demonstrated.

Early testing in simulation shows that this continuous, geometry-based interface enables the swarm to compress through simulated corridors, expand to cover survey areas, and reorient its diagonal axes. Future work includes implementing 3D shape control and orientation alignment on the real swarm, measuring end-to-end latency, and conducting a user study to evaluate the effectiveness of this control method.

Aerial Swarm Flocking Control through Hand Motion

Nevò Mirzai Hamadani, Benjamin Jarvis, Dario Floreano, *Fellow, IEEE,*

***Abstract*—Aerial swarms can survey large areas rapidly, offer redundancy, and execute tasks a single drone cannot, making them attractive for monitoring, disaster response, and entertainment. Despite this potential, there is still no widely adopted, intuitive interface for commanding their many coupled degrees of freedom. Previous work has mostly focused on discrete hand-gesture commands, which introduce latency and restrict to predefined maneuvers. This work presents a continuous, geometry-based Body–Machine Interface (BoMI) that maps the high DoF pose of the operator's hands—captured by a VR headset—to control the collective behaviour of an aerial swarm. A modified Olfati–Saber algorithm converts hand position and orientation into velocity, axis-specific dilation/contraction, and rotation of the swarm's cohesion frame. Simulation experiments demonstrate stable 3-D flocking with simultaneous control of scale, translation, and axis rotation, while hardware tests on Crazyflie drones demonstrate 2-D dilation and velocity control. The approach removes the need for gesture classification and prior training, while preserving the operator's spatial intuition.**

***Index Terms*—Aerial Drone Swarm, Hand Control, Virtual Reality (VR), Flock control**

I. Introduction

SINCE the early 2010s, drones have seen a rapid increase in popularity, supported by improvements in consumer robotics and accessibility [1]. When deployed as coordinated groups, aerial swarms can cover larger areas faster, provide intrinsic redundancy, and perform tasks that a single platform cannot achieve—making them attractive for environmental monitoring, disaster response, entertainment, and logistics [2]. Yet their many coupled degrees of freedom (DoFs) also make them harder to pilot: an operator must simultaneously manage both the collective pose and the relative spacing of dozens of units. While control techniques for single drones are well established [1], no widely adopted interface yet delivers intuitive, high-DoF command of multi-drone swarms [3]. Traditional teleoperation relies on handheld controllers that demand extensive training and provide limited command dimensionality [4].

Body–Machine Interfaces (BoMIs) have been proposed to address this challenge by mapping natural human motion onto robotic commands [5]. Early BoMIs for single-drone piloting relied on vision-based gesture recognition or wearable sensors to detect predefined hand signs [6]. For single-drone control, gestures mainly modify trajectory. Extending this approach to swarms requires additional gestures for formation changes [7], but the number of gestures grows rapidly with the diversity of possible formations. More importantly, each formation must be triggered by a discrete gesture, so operators cannot guide the swarm in a truly intuitive or continuous manner. As a result, cognitive load increases and real-time adaptability suffers [8].

Continuous geometric mappings have therefore attracted interest. The DronePaint project enabled users to draw free-form trajectories that a drone would follow for light painting, but it operated at low update rates and does not support multi-drone control [9]. Macchini *et al.* analysed spontaneous strategies of naïve users and derived personalized hand-motion mappings for planar, uniform scaling of a simulated swarm [10]. Because that mapping uses a single scalar separation distance, the swarm can only grow or shrink isotropically; it cannot reshape itself along specific axes. Axis-specific dilation, in contrast, enables the swarm to contract and pass through doorways more efficiently—compared with uniform scaling—by adapting to the dominant constraint direction, expand laterally to blanket a search area, or elongate to inspect linear structures such as power lines. Consequently, supporting a *vector* of separations (one value per principal axis) is essential for unlocking the geometric versatility that motivates swarm deployment.

The present work addresses these limitations by using the pre-implemented hand tracking provided by a consumer virtual-reality (VR) headset, with a modified version of the Olfati–Saber algorithm. Translating both hands controls swarm velocity; the distance between the hands and the degree of hand openness modulate scale independently along two principal axes; and the orientation of the hands in space rotates the cohesion frame, enabling diagonal ellipsoidal formations. The mapping runs in real time on a simulated 3D drone swarm and transfers to a 2D swarm of real drones. The results show that VR headset sensors enable continuous, multi-DoF human control of aerial swarms without the need for gesture recognition. This contributes a compact and intuitive control framework suitable for real-world, on-site deployment of human-operated drone swarms.

II. Method

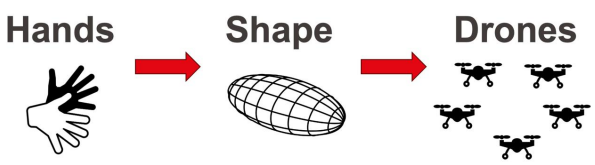


Fig. 1: Flowchart of the high-level process

The framework used to achieve hand control of a drone swarm relies on three main parts: Hand-to-Shape mapping,

The authors are with the Laboratory of Intelligent Systems, Ecole Polytechnique Federale de Lausanne (EPFL), CH1015 Lausanne, Switzerland.

Shape-to-Swarm controls, and swarm communication (Fig. 1). The first two parts happen inside a Unity simulation, while the third one is separate and is used to control the real drones. Hand-to-Shape mapping consists of converting the hand and joint positions into a 3D ellipsoid. Shape-to-Swarm converts the ellipsoid position and multiple elongations into swarm commands. Finally, these commands are communicated to the real Crazyflie swarm for physical execution.

A. Unity simulation

The Unity simulation runs on a PC VR setup in which a Meta Quest 3 headset is connected to the computer via USB-C. This configuration enables Unity to stream the virtual environment directly to the Quest's display while also receiving real-time hand-tracking data from the headset. In practice, the operator wears the Quest 3 to see a 3D scene in which a fleet of virtual drones is spawned and visualized. Within this scene, each virtual drone is initiated and responds to the hand-to-shape and shape-to-swarm control routines.

B. Hand-to-shape mapping

The VR headset tracks the user's hands within a Unity simulation using the XR Hands package for OpenXR[1]. Although human hands can move in many ways, only a few motions feel intuitive and comfortable to use for control tasks [11]. For example, controlling a degree of freedom by flexing a single finger, such as the pinky, would be neither practical nor ergonomic. In contrast, some gestures—like moving the hands closer together or farther apart, or opening and closing the hands—are naturally more intuitive to control. These actions are used to change the shape of a virtual ellipsoid that is rendered between the user's hands, anchored at the palms. This shape aligns well with swarm control logic: it is compatible with relatively simple drone formations and visually represents its degrees of freedom.

The ellipsoid's length is defined by the distance between the two hands (which increases as the hands move apart), while its thickness—representing the other two axes—is derived from the average distance between the index fingertips and the line connecting the palms (Fig. 2). The orientation of the ellipsoid is determined by the relative angle between the palms. Minimal bounds are applied to both length and thickness to prevent drones from getting too close. However, no upper bounds are enforced: as long as both hands remain visible to the headset, the ellipsoid can expand freely. If the hands are temporarily occluded, the system defaults to a neutral ellipsoid shape. This setup allows users to continuously and naturally generate a wide variety of ellipsoid shapes by adjusting hand spacing and openness (Fig. 3).

The hand-frame used is a local coordinate system anchored to the user's palms: its forward axis (x) points from the left palm to the right palm, with its vertical axis (z) aligned to the world's vertical. Once scaled, the modified vectors are rotated back into world space via the hand-frame quaternion so that the swarm's shape and orientation track the user's natural hand motions. This frame is used in order to independently scale each axis in operations regarding the flocking algorithms.

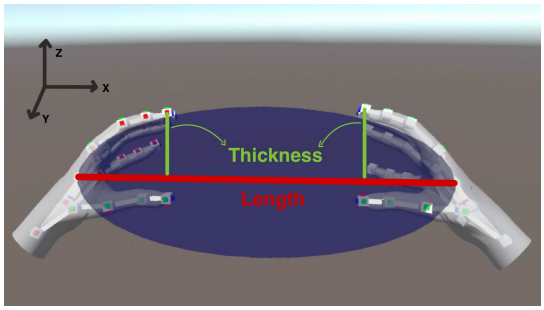


Fig. 2: Hands-to-ellipsoid mapping, the thickness (in green) is measured using the average distance of the index fingertips from the line connecting the palms (in red)

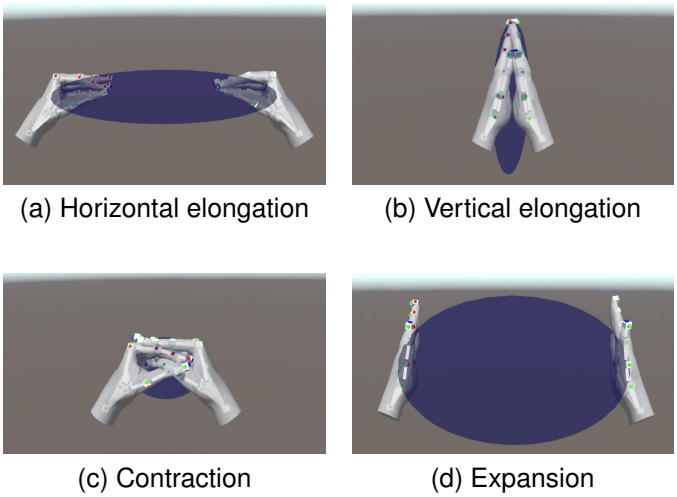


Fig. 3: Ellipsoid shapes generated from different hand poses, showing axis-specific dilation and contraction

C. Shape-to-swarm control

The dimensions of the ellipsoid (length, thickness and orientation) that were found in the first mapping (hand-to-shape) are used in the control of expansion/contraction of the swarm. Two swarm control algorithms were tested: the Reynolds Boids algorithm [12], which is widely used and easy to implement, and the Olfati–Saber algorithm [13], which has the advantage of being more intuitive for teleoperation, especially for new users, as it uses a physical measurement for distance matching [14].

1) Reynolds Boids Algorithm with Axis-Specific Cohesion: The Reynolds Boids algorithm [12] steers each agent with three forces: cohesion ($\mathbf{f}_{\text{coh}}$), separation ($\mathbf{f}_{\text{sep}}$), and alignment ($\mathbf{f}_{\text{ali}}$). Cohesion drives an agent toward the average position of its neighbours, separation repels it from nearby drones to avoid collisions, and alignment matches its velocity to the group's mean. Weighted together, they yield the velocity update

$$\mathbf{u} = w_{\text{coh}}\mathbf{f}_{\text{coh}} + w_{\text{sep}}\mathbf{f}_{\text{sep}} + w_{\text{ali}}\mathbf{f}_{\text{ali}}. \quad (1)$$

[1]XR Hands 1.1.0 package was used for hand tracking.

Hand–frame scaling. To let the swarm dilate differently along each principal axis of the user's hand ellipsoid, cohesion and separation are rescaled in the local *hand frame*. This frame, defined in Sec. II-B, aligns its x-axis with the line between palms and keeps "up" parallel to world Z. Working in the hand frame decouples the user's axes from the global axes; doing the scaling directly in world space would restrict elongations to world X, Y, Z directions and forbid diagonals.

Let $\mathbf{C} = \mathbf{f}_{\text{coh}}$ be the raw cohesion vector in world coordinates and let $R \in \text{SO}(3)$ rotate world vectors into the hand frame. Three steps are performed:

1) *Transform to the hand frame:* $\mathbf{C}_{\text{local}} = R^T \mathbf{C}$.
2) *Scale each axis independently.* The scaling factors

   $$\alpha_x = \text{lengthFactor}\, m, \qquad \alpha_y = \alpha_z = \text{thicknessFactor}\, m$$

   depend on the hand pose: lengthFactor $= L/L_0$ is the ratio of current palm distance L to a baseline L_0; thicknessFactor $= T/T_0$ uses the fingertip spread T and its baseline T_0; m is a user-set *cohesionAxisModifier*. Each component is rescaled as $C'_{\text{local},i} = C_{\text{local},i}/\alpha_i$.
3) *Transform back and weight.* The adjusted vector is returned to world coordinates, $\mathbf{C}' = R\,\mathbf{C}'_{\text{local}}$, then weighted: $\mathbf{f}_{\text{coh}}^{\text{scaled}} = w_{\text{coh}} k_{\text{coh}} \mathbf{C}'$.

Separation is treated identically, except its components are multiplied by α_i to enlarge the repulsive force when that axis is meant to expand. The final control input retains the standard Reynolds form,

$$\mathbf{u} = \underbrace{w_{\text{coh}} k_{\text{coh}} \mathbf{C}'}_{\text{axis-specific cohesion}} + \underbrace{w_{\text{sep}} k_{\text{sep}} \mathbf{S}'}_{\text{axis-specific separation}} + w_{\text{ali}} \mathbf{f}_{\text{ali}},$$

where $\mathbf{S}'$ is the scaled separation vector. Thus the usual Boids equation (Eq: 1) is preserved while allowing the user's hand pose to stretch or contract the swarm along arbitrary, hand-aligned directions.

2) Olfati-Saber Algorithm with Axis-Specific Cohesion: The Olfati–Saber algorithm [13] replaces the simple Boids rules with a more rigorous potential-based approach that guarantees smoother and more stable flocking behavior. Each agent's control input $\mathbf{u}$ is composed of three main terms: a cohesion term $\mathbf{f}_{\text{coh}}$, an alignment term $\mathbf{f}_{\text{ali}}$, and an obstacle-avoidance/migration term $\mathbf{f}_{\text{mig}}$. In compact form,

$$\mathbf{u} = \mathbf{f}_{\text{coh}} + \mathbf{f}_{\text{ali}} + \mathbf{f}_{\text{mig}}.$$

a) Axis-Specific Cohesion.: Olfati–Saber derives cohesion from a decreasing potential function based on the distance between agents. Let each neighbour j have relative position $\mathbf{r}_{ij} = \mathbf{p}_j - \mathbf{p}_i$ in the agent's local (swarm-aligned) frame. A reference distance is defined for each axis, ($d_{\text{ref},x}$, $d_{\text{ref},y}$, $d_{\text{ref},z}$), and updated at runtime from the hand pose (Sec. II-B):

$$d_{\text{ref},\,i} = d_{\text{ref}} \begin{cases} \text{thicknessFactor}, & i \in \{y, z\}, \\ \text{lengthFactor}, & i = x, \end{cases} \qquad i \in \{x, y, z\},$$

where d_{ref} is a baseline scaling constant.

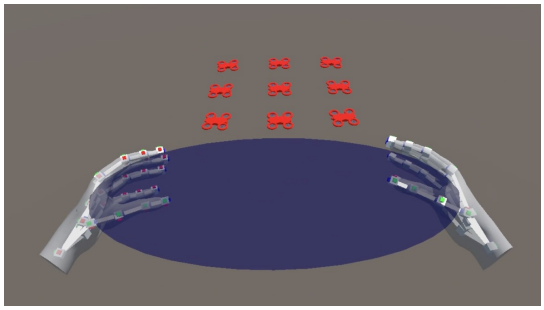
For each axis i, the scalar cohesion contribution depends on $r_{ij,i} = |\mathbf{r}_{ij,i}|$ through the derivative of the Olfati–Saber potential $W(\rho)$ (see Appendix IV-A). Parameters of W are tuned so the swarm converges smoothly without oscillation yet remains responsive to change. Summing the axis-scaled contributions of every neighbour yields the cohesion vector in the local frame:

$$\mathbf{f}_{\text{coh}} = \sum_{j \neq i} \begin{pmatrix} f_{\text{coh},x}(r_{ij,x})\,\text{sign}(r_{ij,x}) \\ f_{\text{coh},y}(r_{ij,y})\,\text{sign}(r_{ij,y}) \\ f_{\text{coh},z}(r_{ij,z})\,\text{sign}(r_{ij,z}) \end{pmatrix}.$$

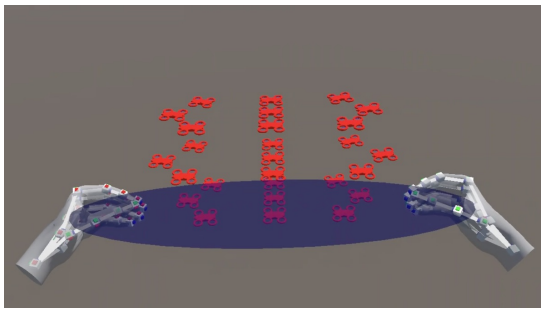
This axis-dependent formulation lets the swarm expand or contract independently along each principal direction, matching the user's hand-defined length and thickness factors.

b) Obstacle Avoidance and Migration.: Each agent also senses obstacles within a radius r_{obs}. If an obstacle is detected at relative position $\mathbf{o}$, an avoidance force pushes the agent away. A migration term $\mathbf{f}_{\text{mig}}$ can then steer the agent toward a target region or altitude [13].

3) Velocity Control: Velocity control is implemented by modifying the alignment force term f_{ali} in the Olfati–Saber algorithm. The midpoint between the operator's hands steers the swarm: when both palms move beyond a small "dead zone" (where the velocity command given is 0) around a calibrated center point, their displacement is scaled and used to set the desired swarm velocity $\mathbf{v}_{\text{des}}$. A minimum velocity is applied immediately outside the dead zone, with no maximum limit in simulation; however, in the physical experiment, velocity was capped at 1m/s for safety. The calibration is done at the start of the interaction: when a designated key is pressed, the system records the current midpoint between the hands as the neutral reference position for velocity commands. The



(a) 2D swarm formation



(b) 3D swarm formation

Fig. 4: Comparison between a 2D (a) and a 3D (b) ellipsoid swarm formation controlled by the Olfati-Saber flocking algorithm.

alignment force then drives each agent's velocity $\mathbf{v}$ toward $\mathbf{v}_{\text{des}}$. In world coordinates:

$$\mathbf{f}_{\text{ali}} \;=\; c_{\text{vm}}\big(\mathbf{v}_{\text{des}} - \mathbf{v}\big),$$

where c_{vm} is the velocity-matching gain. A safety mechanism prevents the swarm from descending below a preset altitude: after calibration, the initial swarm height h_0 is stored, and any commanded descent below h_0 is limited to a gentle recovery speed, while small vertical displacements around h_0 do not force downward motion.

4) Diagonal Control: Diagonal control relies on a single angle—yaw—derived from the orientation of the operator's palms to steer the swarm's elongation direction. Let $\mathbf{p}_R$ and $\mathbf{p}_L$ be the world-frame positions of the right and left palms, and define the palm-connection vector $\mathbf{d} = \mathbf{p}_R - \mathbf{p}_L$. Projecting $\mathbf{d}$ onto the horizontal XY plane (Fig. 2) and comparing its direction to the world Y-axis (noting that comparing with the X-axis is just a 90° rotation) yields the desired yaw angle θ. This yaw angle is applied by rotating the reference axes used for Olfati–Saber's axis-specific distances (Sec. II-C2). In other words, the swarm does not literally turn; instead, it elongates along a diagonal direction determined by θ. To switch from one diagonal elongation to another—for example, from a 45° elongation (Fig. 5) to a –45° elongation—the swarm first contracts along the 45° axis and then begins elongating along the –45° axis.

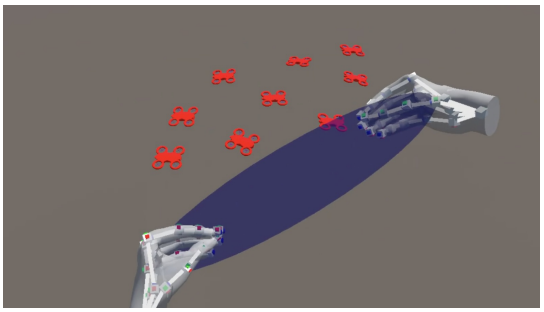


Fig. 5: ∼45° rotation of cohesion axes

D. Hardware Experiments

The system was tested on real Crazyflie drones, controlled through a ROS2 pipeline. Since ROS2 is only supported on Linux, while the Unity simulation must run on Windows due to compatibility with the Meta Quest app, a method was required to connect the two systems and send the hand commands to control the swarm. A UDP connection was chosen to bridge the Windows and Linux machines. This section outlines how the UDP communication and ROS2 components operate.

1) UDP: UDP (User Datagram Protocol) is a lightweight communication protocol used to send data quickly over a network. It does not establish a connection or guarantee delivery, order, or error checking—each packet is sent independently, which makes the transmission very fast and efficient. This is particularly useful in applications where speed is more critical than reliability—as in this case, where losing a few drone commands has minimal consequences, but fast communication is essential to prevent drones from destabilizing and falling. However, the main drawback is that if packets are lost, delayed, or arrive out of order, UDP doesn't correct them.

2) ROS2: The library used to control the swarm is Crazyswarm2[2]. It is a ROS2-based library that provides high-level APIs for managing and controlling Crazyflie drone swarms. Crazyswarm2 was chosen because it handles low-level radio communication, timing, and multi-drone coordination automatically, allowing our system to publish desired velocities and axis-specific dilation commands without writing custom drivers. By using Crazyswarm2, the UDP-received swarm parameters are relayed to each Crazyflie through ROS2 topics with minimal latency.

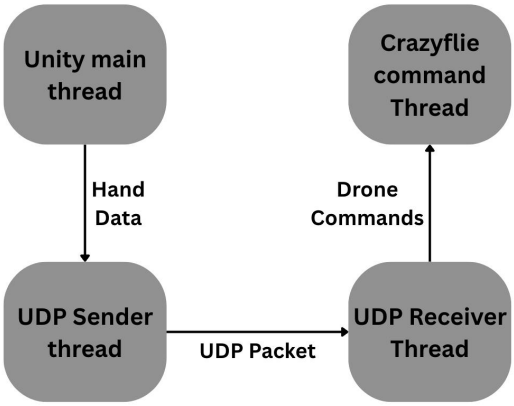


Fig. 6: Thread chain structure of the code

3) Structure of the code: The code is structured in 4 main threads that work in a chain structure (Fig. 6): the Unity main thread, the UDP Sender thread, the UDP Receiver thread and the Crazyflie command thread.

The **Unity main thread** runs inside the VR application and is responsible for capturing hand–tracking data every rendering frame. It queries the XR Hands interface, converts the raw palm and fingertip positions into the high-level swarm parameters described in Section 3 (desired velocity, yaw rate, and the axis-specific dilation factors), and stores these values in shared variables that can be accessed safely by the networking code. Because Unity's update loop already operates close to the headset's display rate, this thread provides fresh input at 90–120 Hz without blocking on any network operations.

The **UDP sender thread** executes in parallel with the main loop. At a fixed interval it locks the shared data, serialises the latest swarm parameters into a JSON packet, and transmits the packet through a pre-opened socket to the computer controlling the crazyflies. The information sent are the reference distances and reference velocity command needed for the Olfati-Saber flocking algorithm (Section II-C2). The payload is well below the maximum UDP datagram size, so no fragmentation is required.

The **UDP receiver thread** listens on the corresponding port and delivers each packet to the ROS2 node that runs the high-level control logic. Upon arrival, an integrity check compares

[2]https://github.com/IMRCLab/crazyswarm2

the packet length with the expected byte count; malformed or out-of-order packets are discarded. Valid packets are parsed immediately, and the decoded swarm parameters are pushed into a thread-safe queue for downstream consumption. Because UDP is connectionless and does not guarantee delivery, the receiver sets default values for each parameter in case no packet was received.

Finally, the **Crazyflie-command thread** takes the most recent swarm command, scales the distance references and desired velocity (the drones in the simulation do not behave nor they have the same size as the real drones), and publishes them on the radio link at 100 Hz. This thread maintains its own timing so that radio bandwidth is used efficiently even if the upstream UDP packets arrive slightly irregularly. As soon as a new command becomes available, it overwrites the older set-point to minimise latency, ensuring that the physical swarm follows the operator's hand motions with minimal delay.

E. Evaluation

For evaluation purposes, obstacles were added to a simulated environment in simulation to test the effectiveness of this type of continuous geometric swarm hand control. Two types of obstacles were used: a simple rectangle as a "wall" obstacle and a rectangle with a hole as a "gate" obstacle. The swarm should move around the wall obstacle and/or pass inside the gate (Fig. 7).

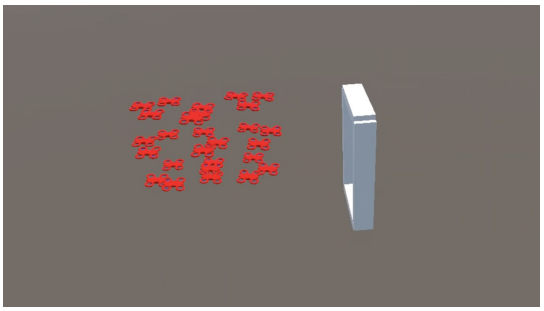
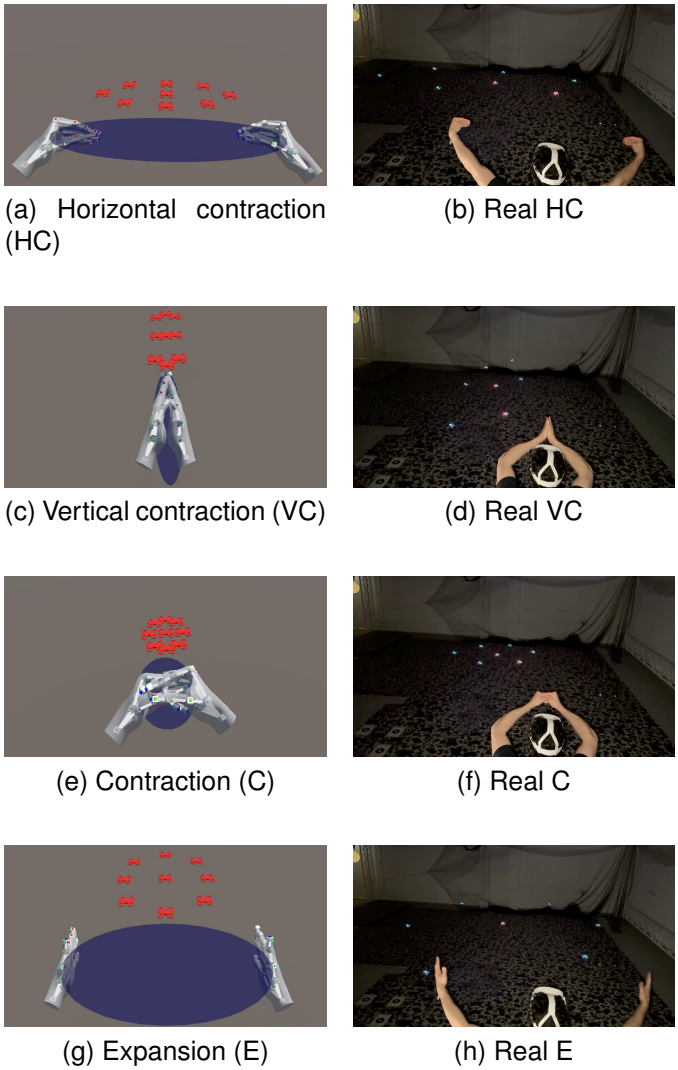


Fig. 7: Initial configuration of the 3D swarm in front of a gate obstacle

III. Results

During the evaluation in the simulated environment, the operator was able to successfully guide the swarm around obstacles and through gates (Figure 9). To navigate around a wall, the swarm was contracted to make it easier to move around; this was done by moving the hands closer together and reducing finger spread (Fig. **??**). To pass through a narrow gate, contraction was applied along a single axis, either by placing the hands side by side, as in a prayer gesture, or by spreading the hands apart and pinching the fingers inward. Translational motion was achieved by moving both hands in the same direction, thereby steering the swarm with a consistent velocity. It should be noted that the simulation's visuals did not sync with the real drones' behavior. The Crazyflies' thrust, responsiveness, size, and weight differ from the simulated models, so what the operator saw in VR did not exactly match the real-world motion. To compensate during real-drone tests, the VR headset was posed on the operator's forehead so they could both perform hand commands and directly observe the physical swarm's movements.



(a) Horizontal contraction (HC)

(b) Real HC

(c) Vertical contraction (VC)

(d) Real VC

(e) Contraction (C)

(f) Real C

(g) Expansion (E)

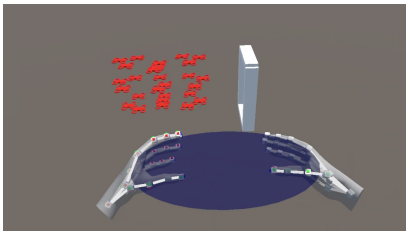
(h) Real E

Fig. 8: Ellipsoid shapes (left column: 8a, 8c, 8e, 8g) generated from different hand poses in 2D alongside their real-world equivalents (right column: 8b, 8d, 8f, 8h), illustrating horizontal contraction, vertical contraction, overall contraction, and expansion.

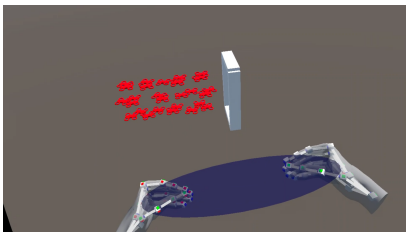
IV. Conclusion

The proposed continuous, geometry-based BoMI enables real-time axis-specific control of an aerial swarm, demonstrated by a stable 3D flocking in simulation and successful 2D dilation and velocity control on physical Crazyflie drones. By mapping hand distance, openness, and orientation to an ellipsoidal formation, the operator can smoothly compress, expand, translate, and rotate the flock.

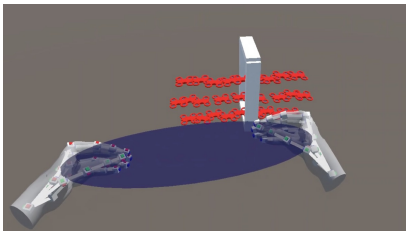
However, several limitations remain. The experiments on real drones were limited to planar (2D) maneuvers and without rotation or diagonal control. End-to-end latency was not



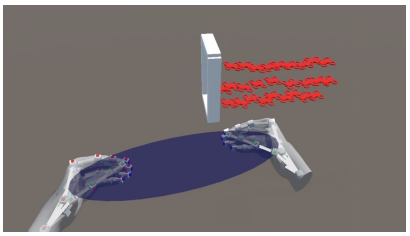
(a) Initial swarm formation approaching the obstacle.



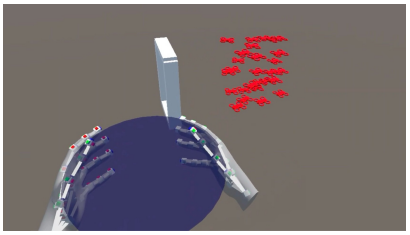
(b) Swarm contracts laterally to pass through the gate.



(c) Swarm enters the gate.



(d) Swarm passed the gate.



(e) Swarm fully exits the gate and resumes formation.

Fig. 9: Evolution of the 3D swarm as it navigates through the gate.

rigorously quantified, and no user study has evaluated operator workload or long-term usability.

Despite these limitations, the intuitive and continuous hand control of drone swarms proposed in this work offers a step toward more responsive field deployment in scenarios requiring human oversight. It allows a single operator to rapidly adapt swarm formation and movement in dynamic contexts such as search-and-rescue, agricultural inspection, or live performances.

References

[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, pp. 460–466, 2015.

[2] 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.

[3] S. Coppola, P. Abichandani, G. Beltrame, and A. H. Diamantis, "A survey on uav swarm applications, architectures, and challenges," *IEEE Access*, vol. 8, pp. 82 635–82 664, 2020.

[4] G. Niemeyer, C. Preusche, and G. Hirzinger, "Telerobotics," in *Springer Handbook of Robotics*. Springer, 2008, pp. 741–757.

[5] M. Casadio, R. Ranganathan, and F. A. Mussa-Ivaldi, "The body–machine interface: A new perspective on an old theme," *Journal of Motor Behavior*, vol. 44, no. 6, pp. 419–433, 2012.

[6] N. Kiselov, "Tello gesture control," https://github.com/kinivi/tello-gesture-control, 2021.

[7] M. Hu, Y. Li, Z. Zhang, and Y. Li, "Hgic: A hand gesture based interactive control system for multi-UAVs," *arXiv:2403.05478*, 2024.

[8] S. Oviatt, "Designing interfaces that support communication and interaction," in *Proceedings of the 10th International Conference on Multimodal Interfaces (ICMI '08)*. ACM, 2008, p. 125–132, shows how discrete gesture schemes increase cognitive load and limit real-time interaction.

[9] V. Serpiva, A. Vakhitov, F. Berestov, D. Anastassov, and D. Rus, "DronePaint: Swarm light painting with DNN-based gesture recognition," in *SIGGRAPH Emerging Technologies*, 2021.

[10] M. Macchini, L. D. Matteïs, F. Schiano, and D. Floreano, "Personalized human–swarm interaction through hand motion," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 179–186, 2021.

[11] J. P. Wachs, M. Kölsch, H. Stern, and Y. Edan, "Vision-based hand-gesture applications," *Communications of the ACM*, vol. 54, no. 2, pp. 60–71, 2011.

[12] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987, pp. 25–34.

[13] 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.

[14] B. Jarvis, C. Toumieh, and D. Floreano, "First-person view interfaces for teleoperation of aerial swarms," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025, p. —.

APPENDIX

A. $W(\cdot)$ function in the Olfati-Saber flocking algorithm

The Olfati–Saber cohesion term is derived from a decreasing potential $W(\rho)$, where $\rho = r/d_{\text{ref}}$ is the normalized inter-agent distance. This potential is defined piecewise to encourage attraction when agents are within range and to vanish beyond $\rho = 1$. Specifically,

$$W(\rho) \;=\; \begin{cases} \dfrac{a+b}{2}\Big(\sqrt{1+(\rho+c)^2} \;-\; \sqrt{1+c^2}\Big) & \\ \qquad\qquad +\; \dfrac{a-b}{2}\,(\rho-1), & 0 \le \rho \le 1, \\ 0, & \rho > 1. \end{cases}$$

Here, $a > b > 0$ and c controls the smoothness near $\rho = 0$. The derivative of W with respect to ρ is

$$W'(\rho) \;=\; \begin{cases} \dfrac{(a+b)(\rho+c)}{2\sqrt{1+(\rho+c)^2}} \;+\; \dfrac{a-b}{2}, & 0 \le \rho \le 1, \\ 0, & \rho > 1. \end{cases}$$

Finally, the cohesion force along each axis i is

$$f_{\text{coh},\,i}(r) \;=\; \frac{1}{r_0}\, W'\Big(\tfrac{r_i}{d_{\text{ref},\,i}}\Big)\, \frac{1}{d_{\text{ref},\,i}},$$

where r_0 is a baseline radius, r_i is the distance along axis i, and $d_{\text{ref},\,i}$ is the axis-specific reference distance. This formulation produces an attractive force for $0 \le \rho \le 1$ and zero force otherwise, ensuring smooth, bounded cohesion behavior.