Programming Human-Drone Interactions: Lessons from the Drone Arena Challenge

Mousa Sondoqah[+], Fehmi Ben Abdesslem[*], Kristina Popova[†], Moira McGregor[‡], Joseph La Delfa[†], Rachael Garrett[†], Airi Lampinen[‡], Luca Mottola[+], and Kristina Höök[†]
[+]Politecnico di Milano (Italy), [*]RI.SE Computer Science (Sweden),
[†]KTH Royal Institute of Technology (Sweden), [‡]Stockholm University (Sweden)

ABSTRACT

We report on the lessons we learned on programming human-drone interactions during a three-day challenge where five teams of drone novices each programmed a nanodrone to be piloted through an obstacle course using bodily movement. Center to the participants' learning process was the eventual shift from the deceptively simple idea of seamless human-drone interactions, to the reality of drones as non-predictable systems prone to crashes. This happened as participants had to first realize, then to deal with the limitations of the drone's resource-constrained hardware. Coping with these limitations was crucially complicated by the lack of appropriate programming abstractions, which led participants to focus on plenty of low-level, sometimes immaterial details, while losing focus on the ultimate objectives. We find concrete evidence of these observations in how participants handled the visibility problem in debugging drone behaviors, applied different defensive coding techniques, and altered their piloting practice. Our insights may inform further research efforts in drone programming, especially in the vastly uncharted territory of human-drone interactions.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; • **Computer systems organization** → *Robotics.*

KEYWORDS

Human-drone interaction, Drone programming, Challenges

ACM Reference Format:
Mousa Sondoqah[+], Fehmi Ben Abdesslem[*], Kristina Popova[†], Moira McGregor[‡], Joseph La Delfa[†], Rachael Garrett[†], Airi Lampinen[‡], Luca Mottola[+], and Kristina Höök[†], [+]Politecnico di Milano (Italy), [*]RI.SE Computer Science (Sweden),, [†]KTH Royal Institute of Technology (Sweden), [‡]Stockholm University (Sweden). 2024. Programming Human-Drone Interactions: Lessons from the Drone Arena Challenge. In *The 10th Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet' 24), June 3–7, 2024, Minato-ku, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3661810.3663471

1 INTRODUCTION

Aerial drone technology represents a new breed of mobile computing platform, enabling applications in a range of fields such as precision agriculture, film-making, and logistics [13].

Human-drone interactions. Aerial drones are also foreseen to operate in close connection with humans [6, 14]. Human-drone interactions are found in applications such as assisted living, augmented reality games, and artistic performances [2, 11, 17–19]. Programming human-drone interactions is, however, a completely different job compared to mainstream drone applications.

Drones that operate close to humans are usually resource-constrained, for safety and practical reasons. Consider for example the Crazyflie nano-drone, often used in these settings [2, 18, 19]. It weights a mere 27 grams and mounts a single ARM Cortex M microcontroller plus a few low-power sensors. Compare this to the multi-core computing units and the multitude of sophisticated sensors of regular DJI drones.

Human-drone interactions, moreover, are arguably much less predictable compared to the interactions a drone experiences with the surrounding environment in mainstream applications. *Accurately detecting human intentions* is, however, crucial for drones to show intelligent behaviors in response to human behaviors. Doing so on top of constrained hardware greatly complicates matters.

Problem. We argue that the current state of the art in programming human-drone interactions is largely insufficient. We offer concrete evidence as we examine the unfolding of a three-day, hackathon-style drone challenge where five teams each programmed a nanodrone to be piloted through an obstacle course using bodily movement. We intentionally target *novice drone programmers*, as applications with human-drone interactions are likely to be developed by domain experts, not by skilled embedded programmers.

Achieving the challenge goals required coping with the many software/hardware limitations at hand. Doing so was complicated by the lack of appropriate programming abstractions. Participants spent great time dealing with low-level details, such as understanding the readings of proximity sensors to detect humans' intentions, which dragged them away from the key objectives.

Abstractions that existing drone development environments provide towards human-drone interactions are minimal. Participants had to use primitive programming interfaces and debugging facilities, essentially reasoning on raw sensor readings. Contrast this, for example, with the rich development environments available for programming mobile apps for smartphones and the like: *it would be like pretending that one needed to reason with the specific (x,y) coordinates affected by the user's touches on a phone screen, rather than with buttons and windows.*

Road-map. We briefly survey existing work in drone programming and human-drone interactions in Sec. 2 and illustrate the organization and setup of the challenge in Sec. 3.

In Sec. 4, we distill *six lessons* we learn by observing the participants' work and by inspecting their final implementations. We discuss how participants handled the visibility problem [21] in debugging drone behaviors, applied different defensive programming techniques [25], and altered their piloting practice.

These lessons are the stepping stone for us to formulate directions for future work in programming human-drone interactions, discussed in Sec. 5. The aspects we consider crucial include *i)* designing abstractions that allow developers to *reason on human intentions using higher-level concepts* such as "a person is waiving at the drone", rather than raw sensor readings, and *ii)* creating debugging techniques to gain *real-time accurate visibility into a drone's internal states* as its interactions with humans evolve.

We focus on embedded drone programming here, yet also provide a somaesthetically focused discussion and directions on how to better support somatic engagement with drones elsewhere [28].

2 BACKGROUND AND RELATED WORK

We briefly survey drone programming and discuss prior research on human-drone interactions.

Drone programming. Several drone platforms employ ROS [26] as underlying support with bindings available for several programming systems and simulation environments. Based on this, software packages are built to offer basic functionality such as waypoint navigation [20]. Similarly, many commercial drone platforms offer Software Development Kits (SDKs) to create mobile applications for smartphones and tablets [8], which are often proprietary.

Programming drone swarms and teams is a different problem [7, 23]. In these systems, the programmers' commands are translated into a sequence of primitive instructions deployed onto all drones. Simple drone behaviors are shown to produce emergent collecting properties, such as dispersion or flocking.

Common to most works is enabling *autonomous* operation. A drone's program usually already embeds the entire application logic and humans are normally out of the loop. In the setting we consider, humans are integral part of the application logic, as their interactions with the drone directly determine how efficiently the human-drone ensemble achieves the goal.

Human-drone interactions. The social-cultural implications [22] of moving drones from military to civil operations prompt research into human-drone interactions [6, 14]. Research into the relational aspects of drone technology focuses on so-called "natural" human-drone interactions [4] and related emotions [5, 16, 30].

When humans interact with drones, they need to adapt themselves in how to control and move around them. This happens when using drones for work purposes [18], as part of leisure activities [2, 17], artistic performances [11, 19], or in family settings [15]. For example, Popova et al. [24] open up a new design space through the combination of humans and drones. Their account of a design process that unfolds over a significant period of time contrasts with this paper, where participants develop human–drone interactions within a limited time frame and without dedicated training.

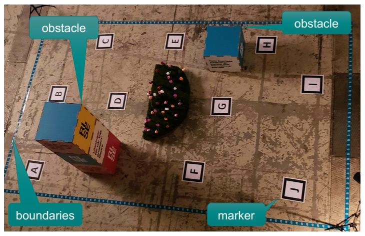

Figure 1: One of the drone arenas, seen from above.

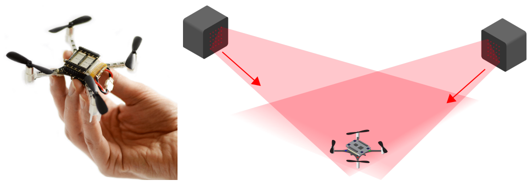

Figure 2: Crazyflie nano-drone.

3 CHALLENGE

We set up the challenge as a three-day hackathon at the R1 Reactor Hall of KTH campus in Stockholm, Sweden[1]. The hall sits 40 m underground and used to host an experimental nuclear reactor. It is used nowadays for exhibitions, concerts, and music videos.

Objective and rules. We take inspiration from the '80s maze action videogame Pac-Man. The goal is to fly a *nanodrone* through an obstacle course set in a *drone arena*, shown in Fig. 1, in a fixed time while collecting as many markers as possible. Drone piloting is achieved by a competitor-participant who physically interacts with the drone in the arena, for example, through gestures that are detected by the drone's onboard *proximity sensors*. Collecting a marker is achieved by flying the drone over it at any height.

The team members implement the drone control logic running on a remotely-connected control station as a Python program. The program determines how a drone reacts to a human pilot's interactions. For example, it steers the drone sideways when a participant moves her hand closer to the drone on one side. Only one human pilot at a time can be active in the arena during a run.

Participants cannot touch the drone, carry any digital device, or install anything in the arena, such as additional markers. We also discourage participants from taking an unnecessarily competitive or rushed approach. We state learning about drones and having fun as the main goals rather than winning, although we do explain that there will be prizes for the most successful teams. Framing the challenge around learning and emphasizing that the participants are free to decide how much time they want to spend working on the challenge, we try to establish a calm and supportive atmosphere.

Technology. We use the Crazyflie 2.1, shown in Fig. 2, and make available to the participants the whole range of expansion decks, including those with the ToFVL53L1x proximity sensors and the PMW3901 optical flow sensor. Besides the Crazyflie environment, we also provide examples of the control logic running on the control station that shows how to achieve simple piloting functionality, such us "pushing" the drone in a given direction when the drone's proximity sensor detects that someone's hands are approaching, or "pulling" the drone when the hands are withdrawing.

[1]We obtained IRB approval from the authors' institution wherever applicable.

We equip the drone arena with the Lighthouse localization system. The system uses two base stations deployed at opposite corners of the arena to emit infrared laser scans, as depicted in Fig. 2. These are detected by the drone using dedicated sensors. The Lighthouse localization system was sufficiently simple to install in a *temporary* location and provided a user experience largely similar to other indoor localization systems, including optical ones [1]. Collecting a marker is achieved by matching the drone's coordinates in the horizontal plane with those of the marker, which we know a priori.

The Lighthouse system works reliably only as long as the laser scans can constantly reach the drone, similar to an OptiTrack system requiring line of sight to the markers aboard the drone. If the path from a base station to the drone is somehow occluded, say because a person moves in between, the drone temporarily loses the base station inputs. The drone may then become unstable yet eventually reclaims a stable behavior, or end up in a crash. These aspects were a significant emergent factor during the challenge.

Schedule and teams. The event begins with an inspiring *opening talk*, followed by a *a two-hour tutorial* on programming the Crazyflie, using teaching material we develop. By the end of the tutorial, even teams with no previous experience can fly their assigned drone.

The remainder of day one and the second day are devoted to *challenge trials*; teams are free to work shorter or longer hours at the Reactor Hall to program their drone to interact with their nominated pilot in the arena. We set up two separate drone arenas for the trials. Experts on our team are available to provide technical support. Day three of the event begins with the obstacle challenge, observing the objectives and rules outlined in Sec. 3. Each team is allowed four attempts. The event concludes with a prize ceremony.

Participation is free of charge and open to anyone irrespective of age, profession and past experience with drones, on a first-come-first-served basis. As we provide all necessary drone hardware, a team only needs to bring a computer for programming. Although the challenge is advertised as open to all, the nature of the challenge makes it compelling for those with some familiarity with drones and/or some programming skills. The timing of the challenge during three weekdays in June, along with its location on a university campus, make it particularly attractive to students. Most participants had previous programming experience, but only a few had prior experience with drones, for example, in activities like aerial filmography that have little overlap with the challenge.

A total of 22 persons participated, split in six teams. All teams but one were entirely composed of university students. Video examples of actual challenge runs are available [9].

4 LESSONS LEARNED

Based on interviews, field notes, video recordings, and code inspection, we report on how the lack of appropriate programming and testing abstractions made it overly complex for the participants to cope with the limitations of the drone hardware. In the following, we discuss the concrete evidence we collected.

4.1 Crashes

The message conveyed during the opening talk concentrated on *success stories*, showing humans smoothly interacting with drones.

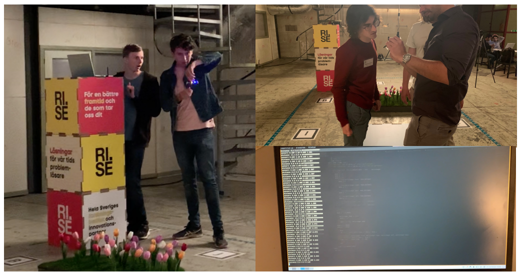

Figure 3: Investigating drone behaviors through logging data.

The speaker demonstrated videos showing gestures accurately commanding drones, akin to what the participants are to accomplish in the challenge, without commenting on how these interactions are technically achieved. Not a single crash or malfunction was shown, yet the drone platform at hand was the *same* as in the challenge.

When the time came for the trials, participants realized that they were essentially operating on a different playground compared to the opening talk. The drone behavior out of the box was described as *brittle* and *hard to predict*. Discussions with our technical support were mostly about how to understand why the drone did not behave as expected and (most often) crashed. Participants eventually figured that much of what they were observing, compared to the opening talk, was due to a fundamentally different setup, which drastically increased the uncertainty of how a drone can *sense human intentions*. Two aspects were key.

Albeit immaterial w.r.t. the work in the opening talk, those demonstrations often relied on some sort of instrumentation *on the human body*, which we explicitly forbid in the challenge instead. For example, videos were shown in the opening talk where hand gestures were recognized with the human wearing a pair of Lighthouse receivers, one on each hand, identical to the one on the drone. This way, hand gestures were tracked exactly like drones, on a control station that was never shown in the video and much greater accuracy that with the ToFVL53L1x proximity sensors.

The navigation sensors in the opening talk, moreover, were never used for anything but feeding the flight controller. The application logic resided entirely on the control station, and no changes were required to the firmware aboard the drone. Adopting the same design was not fully possible in the challenge. As we explain next, the sensor readings required to recognize the piloting commands in the challenge *partly overlap* with those necessary for flight control. From a programming standpoint, managing such a double use of the same sensor greatly complicated matters.

4.2 Visibility

Participants sought to understand the causes of the drone crashes. Debugging on a device with no operating system and only a few LEDs is arguably an instance of the *visibility problem* [21]. Tools exist to address this problem [31] but they are hard for beginners.

Participants used the Crazyflie logging facility to dump as much run-time data as possible on the control station, as shown in Fig. 3. This is a problem per se. First, the amount of data that some of the teams tried to transfer to the control station was *excessive* and caused memory issues on the Crazyflie, eventually leading to physical crashes. Second, the logging data was *not aligned in time* with the interactions between the human and the drone in the arena.

It was indeed incredibly difficult for participants to figure how a given drone movement in space translated to given sensor readings showing up at different times on the control station. To tame this issue, some participants performed several *mock-up flights*. As shown on the left of Fig. 3, team *RoboNerds* entered the arena with the drone in their hands and mimicked an actual run. While doing so, another team member was monitoring on a laptop the sensor readings coming from the drone as it was approaching obstacles or being subject to different piloting gestures.

Based on these observations, we argue the following.

> *Lesson 1:* Gaining visibility into human-drone interactions requires abstractions to select the *slice of the system state* of interest, as merely inspecting the complete state is unfeasible given available resources. Further, a notion of time must be embedded in state information to let developers *map human interactions with the evolution of the drone's internal states*.

Note how the setting that participants had to deal with is quite different compared to mainstream drone applications [1]. In those cases, predefined sequences of actions the drone should perform are usually known. Consider for example a photogrammetry application [23], where a portion of a site is to be swept by following predefined flight trajectories. In the challenge, the drone behavior is exclusively a function of impromptu human interactions, and no predefined sequences to test against exist.

Next to the troubles in gaining system visibility, participants explored different parameter settings in the example code we provided. As an example, determining the "right" threshold to be used when "pushing" the drone in a given direction when the proximity sensor detects the participant's hand was extremely laborious, often leading to unexplainable crashes. The participants noted that the same parameter setting and similar human gestures were often leading to sharply different drone behaviors.

Two factors contribute to these behaviors. The ToFVL53L1x proximity sensors are both slow and imprecise, which is the price for using a harmless, yet resource-constrained nano-drone platform. The time it takes to detect a nearby obstacle may reach up to a few seconds, compounding the timing problem when checking sensor logs at the control station. On the other hand, human-drone interactions often *do not require perfect accuracy of absolute values*. The application logic is likely to operate based on *given thresholds*, like when detecting whether someone's hand is close.

We summarize these observations as a further lesson.

> *Lesson 2:* Low-level sensing parameters may have *unpredictable, and not necessarily deterministic effects when detecting human intentions*. Latency and noise of sensor readings must be tamed before the data is useful, yet absolute values are not as relevant as in general drone applications, as long as we can robustly detect *whether given thresholds are passed*.

The other factor contributing to unexplainable behaviors was the Lighthouse system requiring almost-constant line of sight between the base stations and the drone, or the latter would lose control and crash. The participants did not immediately realize this. Issues with the localization system were not directly related to any parameter setting, yet they were manifesting with the same drone behavior, that is, a crash. Participants fully understood this only when we explained how exactly a drone knows where it is in the arena.

We therefore conclude the following.

> *Lesson 3:* Masking low-level details through proper abstractions may help beginners program human-drone interactions, yet *specific technical aspects exist*, such as the functioning of the localization system, *that even drone novices must be aware of*.

4.3 Defensive Programming

In response to the lack of a full understanding of drone behaviors, participants applied a number of different *defensive programming techniques* [25]: a programming practice meant to avoid issues before they arise. This is concretely achieved by writing code that, in essence, attempts to cover any possible scenario thrown at it, including situations that might never occur.

Using established code inspection techniques [12], we found evidence of defensive programming at multiple places in the participants' code. Multiple teams changed sensor rates when these are used to detect human gestures, up to a $5x$ increase w.r.t. the examples provided. Obtaining more data allowed them to apply aggressive averaging and filtering to exclude outliers. One team used buffers of up to 100 readings to do so, which is 10 times more than what is normally found in flight controllers [3]. Most teams also capped the maximum drone velocity down to half of what the Crazyflie can do, in an attempt to facilitate the human pilot.

A paradigmatic example, however, is the code written by team *Cyber Ravens*, which includes a *single* **if** statement with *14 different branches*, basically identifying as many different situations and different ways to handle them. We verified that indeed the way this team handles each of these situations is unique, that is, there is no way to refactor the code while keeping the same functionality that would reduce the number of branches. Note how this is not necessarily an indication of an inefficient implementation: team *Cyber Ravens* gained second spot in the final challenge. We did, however, run their code in a lab replica of one of the arenas, and across 36 different runs executed by 5 different people, 9 out of the 14 branches were *never* executed.

Defensive programming produced two effects. First, codebases grow, sometimes in ways disproportionate compared to the complexity of the functionality to be achieved. Some of the teams ended up triplicating the number of code lines compared to the examples provided. We also measured the cyclomatic complexity [10] of the teams' code and found that it is 146% higher than the examples, on average. Larger codebases become more difficult to test, adding to the issues outlined in Sec. 4.2. When more complex execution flows are deployed on the drone, moreover, these require more processing and thus yield slightly higher energy consumption. Albeit processing is generally not an issue on drones as most of the energy is spent in operating the motors, affecting the already limited flight time of a nano-drone may become an issue.

Based on this analysis, we derive a further lesson.

> *Lesson 4:* The uncertainty in detecting human-drone interactions may prompt programmers to apply defensive programming techniques, which however have a cost that cannot be neglected. The question also

remains as to *how far* programmers should push this, as the lack of system visibility makes it difficult to understand what is really necessary.

4.4 Piloting

We observed strikingly different approaches at shaping the human-drone interactions necessary to tackle the challenge. Some teams concentrated on creating programs to *shape the drone behavior*. Other teams applied minimal modifications to the examples we provided and rather worked to *shape their own piloting behavior*.

Shaping drones. The process of shaping drone behaviors must reconcile with the limited hardware available. A key example of this issue is in piloting the drone *vertically*. The examples we provided at the start were not sufficient to capture all markers in the final challenge, as some of the markers were placed on top of objects and not on the floor. The ability to control the drone vertically, not implemented in any of the examples, was mandatory.

To achieve vertical control, many teams tried to use the altitude sensor or the flow sensor mounted at the bottom of the Crazyflie. These sensor inputs, however, are normally not used to impart piloting commands, but contribute to the flight control logic to ensure stable flight. Nonetheless, team *Flying Ferrets*, for example, used a workaround to exclude both sensors from the processing of the flight control loop [27] and used those readings to input piloting commands. We found similar approaches in four teams out of five. They did realize that they were taking an important input away from the flight control logic, but eventually gave up on doubling these inputs for the latter, considering this to be too complex.

These design choices greatly sacrificed overall flight stability, adding to the brittle behaviors we discussed earlier. During the final challenge runs, roughly one in two runs resulted in a crash. Arguably, drones behaved more reliably in the examples we provided at the start of the challenge, even though none of them was sufficient alone to accomplish the challenge.

These insights led us to argue the following.

> *Lesson 5:* Shaping drone behaviors when interacting with humans requires a distinct *separation of sensors used to detect human intentions and sensors used for flight control*. Existing drone platforms lack the necessary hardware for this, while compensating for this via software is extremely difficult.

Rather than shaping the drone behavior programmatically, two teams chose to focus on shaping their own movements.

Shaping the pilot. Teams *Robo Geeks* and *Spark Speed* applied very limited changes to the examples we provided and spent most time in the arenas to gain piloting skills. *Robo Geeks* only added 7 lines of code to one of the examples and spent significantly more time in the arenas compared to other teams; they were ultimately penalized, however, because their additions did not enable vertical control, which was necessary as discussed earlier. When recognizing the problem, they decided to simply skip markers on top of objects during the challenge, hoping their improved piloting skills would compensate for this by collecting all other markers more quickly.

Some teams figured new piloting practices. Team *Cyber Ravens* tried to use self-made cardboard paddles, shown on the left of Fig. 4, to extend the area the sensor could detect. Many also assumed

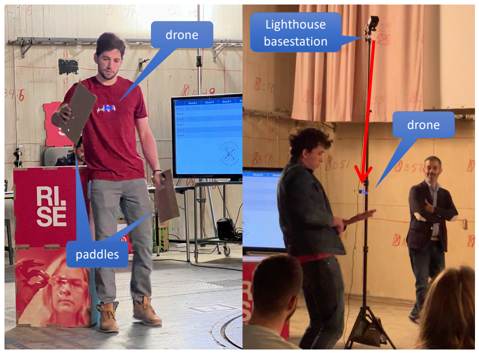

Figure 4: Adapting piloting behaviors.

unconventional poses to keep line of sight between the Lighthouse base stations and the drone, shown on the right of Fig. 4.

These changes to one's piloting practice are, in essence, a direct consequence of the technical shortcomings reported above. The difficulties in programming and testing drone behaviors when interacting with humans brought the latter to give up on making the drone somehow more intelligent, and to resort to changing their own behavior instead. In a way, this is a sign that further work is necessary from a purely technical standpoint, as we further elaborate in Sec. 5, as human behaviors when interacting with drones should be a function of application goals, not of technical hurdles.

We summarize these insights as follows.

> *Lesson 6:* The lack of appropriate abstractions for programming and testing human-drone interactions, together with the inherent hardware limitations of the target platforms, *lead to humans trying to adapt their own behaviors to overcome technical difficulties*, which may not necessarily match the application goals.

5 OUTLOOK

Two directions emerge that are worth additional efforts.

Abstractions. Appropriate abstractions are needed to relieve programmers of low-level details of sensor readings and flight control loops, enabling reasoning at higher semantics levels. As much as programmers of mobile apps reason with buttons and windows in the creation of user interfaces, we advocate the development of analogous concepts *at the interface between humans and drones* [29].

Abstractions of a catalogue of human gestures may be defined, for example, whose implementation is fine-tuned based on the specific hardware and orthogonal to the application goals. Programmers should employ these notions as re-usable building blocks with little to no customization required. Applications should be developed by composing building blocks to achieve application-specific goals, such as capturing the markers in the drone arena challenge.

We particularly advocate abstractions as *programming concepts*, regardless of their specific rendering in a given programming language. The specific implementation in language constructs, as well as the underlying run-time support providing the necessary semantics, may change depending, for example, on whether the execution occurs on the drone or on a control station. The goal should be to define the abstraction semantics *independent of the deployment* *configuration*, providing programmers with an additional degree of flexibility w.r.t. where to execute the application logic.

In contrast, code written by participants of the challenge has no evidence of these higher semantics levels, as participants lose their way in a multitude of sometimes irrelevant details. Their implementations are tied to both the sensing hardware and the deployment configuration. We acknowledge this may be partly due to the specific setting the participants worked in, that is, a hackathon-style event that certainly does not encourage writing reusable code. However, the same teams would have likely achieved much more, in terms of robustness of their implementations, had they been provided with the abstractions we advocate.

Visibility. Gaining sufficient visibility into the system operation is challenging because of two aspects: *i)* human-drone interactions employ resource-constrained hardware, and *ii)* user inputs are unpredictable and unfold over time in response to the drone behavior.

Tackling the first issue is more complex than with regular embedded systems, where plenty of debugging hooks exist and physical connections to the device under test are possible [21]. To increase the realism in testing and debugging, the drone should be let free to fly. The wireless channel linking the drone to a control station is usually a bottleneck per se and often doubles as a control channel.

We argue that the issue should be tackled with a dedicated hardware-software co-design. One may design dedicated debugging hardware, for example, as an additional expansion deck for the Crazyflie platform, which implements the necessary hooks into the main computing unit to monitor the local execution with minimal overhead, so to avoid Heisenbugs [21]. Using state-of-the-art systems on chip, the debugging deck may also offer a *separate* wireless channel to relay the data to a control station, avoiding the overload of the wireless control channel. Programming the debugging expansion deck should, for example, allow programmers to define watches over specific variables to retrieve needed information.

We recognize that the additional debugging deck would come at the cost of increased weight, impacting the flight time. This is the unavoidable price to pay to gain the necessary visibility into the device internals that would incredibly speed up the debugging and testing tasks in developing human-drone interactions.

A dedicated hardware-software co-design may also address issue *ii)* above. The problem is to match the user input with the system state, which was difficult for participants as they had to do that visually, by manually monitoring a screen *and* the pilot in the arena. Similar to the Crazyflie AI-deck, the debugging hardware may be equipped with a low-power camera, which would record the user interactions with the drone *in real time*. By joining the video feed from the camera and the data from the drone computing unit, the debugging hardware could *precisely timestamp* the latter.

A camera would further increase the weight and the on-board processing required to handle the video stream would probably restrict the range of feasible designs. Systems on chip already exist, however, that are apt to the job. The same GAP8 core used on the AI-deck, if appropriately re-purposed, is arguably sufficient.

6 CONCLUSION

Team *Spark Speed* won the challenge, striking the best trade-off between shaping the drone behavior and extending their piloting skills. Team *Cyber Ravens* ranked second, also due to their use of cardboard paddles as an extension of the pilot's hands. Team *Flying Ferrets* ranked third with an accurate tuning of state estimation.

This paper reported an account of the lessons we learned on programming human-drone interactions during the challenge. We recognized that many of the issues at hand originate from the lack of appropriate programming abstractions and the difficulty in gaining the necessary visibility into the system state. We provided directions for future efforts in drone programming, cast into the uncharted territory of human-drone interactions.

Acknowledgments. The work was supported by Digital Futures, project Drone Arena and by WASP-HS, project MMW 2019.0228.

REFERENCES

[1] M. Afanasov et al. 2019. FlyZone: A testbed for experimenting with aerial drone applications. In *MOBISYS*.

[2] B. Baldursson et al. 2021. DroRun: Drone Visual Interactions to Mediate a Running Group *(HRI '21 Companion)*.

[3] E. Bregu et al. 2016. Reactive control of autonomous drones. In *MOBISYS*.

[4] J. Cauchard et al. 2015. Drone and Me: An Exploration into Natural Human-Drone Interaction. In *UbiComp*.

[5] J. Cauchard et al. 2016. Emotion encoding in Human-Drone Interaction. In *HRI*.

[6] J. Cauchard et al. 2021. Toward a roadmap for human-drone interaction. *Interactions* 28, 2 (2021).

[7] K. Dantu et al. 2011. Programming micro-aerial vehicle swarms with Karma. In *SENSYS*.

[8] DJI. 2024. Developer Kit. developer.dji.com.

[9] Drone Arena Challenge. 2024. Example runs. youtu.be/H551SFiZQ-w.

[10] C. Ebert et al. 2016. Cyclomatic complexity. *IEEE Software* 33, 6 (2016).

[11] S. Eriksson et al. 2019. Dancing with drones: Crafting novel artistic expressions through intercorporeality. In *CHI*.

[12] M. Fagan. 2002. Design and code inspections to reduce errors in program development. *Software pioneers: contributions to software engineering* (2002).

[13] D. Floreano and R. Wood. 2015. Science, technology and the future of small autonomous drones. *Nature* 521, 7553 (2015).

[14] M. Funk. 2018. Human-Drone Interaction: Let's Get Ready for Flying User Interfaces! *Interactions* 25, 3 (2018).

[15] M. Gamboa et al. 2021. Ritual Drones: Designing and Studying Critical Flying Companions *(HRI '21 Companion)*.

[16] V. Herdel et al. 2021. Drone in Love: Emotional Perception of Facial Expressions on Flying Robots. In *CHI*.

[17] K. Karjalainen et al. 2017. Social drone companion for the home environment: A user-centric exploration. In *International Conference on Human Agent Interaction*.

[18] M Khan and C. Neustaedter. 2019. An exploratory study of the use of drones for assisting firefighters during emergency situations. In *CHI*.

[19] H. Kim and J. Landay. 2018. Aeroquake: Drone Augmented Dance. In *DIS*.

[20] J. Kramer and M. Scheutz. 2007. Development environments for autonomous mobile robots: A survey. *Autonomous Robots* 22, 2 (2007).

[21] E. Lee and S. Seshia. 2016. *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press.

[22] A. Miah. 2020. *Drones: the brilliant, the bad and the beautiful*. Emerald Group Publishing.

[23] L. Mottola et al. 2014. Team-level programming of drone sensor networks. In *SENSYS*.

[24] K. Popova et al. 2022. Vulnerability as an ethical stance in soma design processes. In *CHI*.

[25] X. Qie et al. 2002. Defensive programming: Using an annotation toolkit to build DoS-resistant software. *ACM SIGOPS Operating Systems Review* 36 (2002).

[26] M. Quigley et al. 2009. ROS: An open-source Robot Operating System. *ICRA Workshop on Open Source Software* (2009).

[27] S. Roumeliotis and G. Bekey. 2000. Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization. In *ICRA*.

[28] M. Sondoqah et al. 2023. Shaping and Being Shaped by Drones: Supporting Perception-Action Loops. In *DIS*.

[29] D. Tezza and M. Andujar. 2019. The state-of-the-art of human–drone interaction: A survey. *IEEE Access* 7 (2019).

[30] S. van Waveren et al. 2023. Increasing Perceived Safety in Motion Planning for Human-Drone Interaction. In *HRI*.

[31] D. Wallace and R. Fujii. 1989. Software verification and validation: an overview. *IEEE Software* 6, 3 (1989).