Continuation to the Rescue: Seamlessly Handling Battery Interruption in Drones

Rodrigo Delgado, Alexandre Bergel

ISCLab, DCC, University of Chile

***Abstract*—Adequately managing energy consumption is critical when defining a drone flight mission. A challenge when programming the drone is to make the drone behavior support battery replacements without disrupting its logic of execution.**

This paper explores the use of *continuation*, an ability of the programming language to capture a particular state of the application. Such a state can be reinstalled after a battery replacement to let the drone resume its execution. Our initial prototype indicates that using continuation enables a drone behavior to resume its execution when replacing a battery, with a minimal prior preparation of the behavior. Our result indicates that a significant reduction of the engineering effort to handle energy consumption in a drone behavior may be achieved.

***Index Terms*—first-class continuation, drone, battery shortage**

I. Introduction

An Unmanned Aerial Vehicle (UAV) is an aircraft without a human pilot aboard, commonly known as a drone. Most drones enjoy an autonomy that almost exclusively depends on an onboard energy unit, a battery.

A fully-charged battery yield may vary considerably depending on the drone behavior [1]: for example, vertical displacements are notorious to drain the battery. As a consequence, the battery's limited capacity imposes strong constraints on a drone mission.

In the case of a behavior duration exceeding the battery yield expectancy, the behavior has to take into account battery replacements during its execution. Once the battery is replaced, the drone behavior is inevitably reset, leading to a second execution of the instructions executed previously by the battery replacement. As a consequence, the logical steps that occurred before the battery shortages are repeated, which, in general, contributes to an abnormal behavior.

This paper explores the use of *continuation*, a programming language feature, to capture and make a first-class entity of the application logic under execution before the battery drainage.

II. Related Works

Properly handling the battery consumption in drones is known to be a challenging problem, and has been addressed in numerous different fashions:

- On one hand, optimizing the battery usage in a flight mission plan may be addressed using an algorithm recharging using a real-path tracking [2]. Another approach is to reschedule the flight mission planning to ensure a safe return [3].
- On the other hand, efforts have been made to produce recharge bases [4] [5] to ensure a continuous flight without manual and tedious battery replacement.

These valuable efforts provide algorithms and planning strategy to have a better handling of the energy consumption. However, they do not mention the impact on the source code of the drone behavior. In contrast to these related works, this article presents a mechanism to seamlessly integrate energy consumption policy without disrupting the drone behavior.

III. Battery in UAVs

The drone's logical behavior is inevitably reset when the drone is switched off and on. The fact that the drone may be reset at any moment during its mission has severe negative consequences on the overall system architecture. For example, dock stations may be available in the flight path to let the drone charge its battery without interrupting its behavior. However, such dock stations are costly, do not work with any drones, and impose some constraints on the flight course.

IV. Continuation

A *continuation* [6] is a reification of the program control state. It is a data structure that represents the computational process at a given point within the application execution and can be manipulated within the programming language as any other value. Continuation is popular within the Lisp, Scheme, and Pharo programming languages.

Consider the following trivial expression `3 * (4 + 2)`. If we decide to capture the continuation while the value `2` is interpreted, then the continuation is the function λ`(x) = 3 * (4 + x)`. Providing the value `2` to the continuation resumes the evaluation of the expression. Continuation is automatically built from the runtime execution stack. Evaluating a continuation resumes the execution at the exact same point where the continuation was originally captured.

Continuations are useful for encoding numerous language features, including exception, coroutines and green threads.

V. Motivating example and prototype

We will use a contrived, but representative example of a drone behavior. The example is then adapted to make it robust against battery replacement and resetting of the drone. We will then illustrate the property of using continuations.

Example. Figure 1 illustrates the behavior of flying a square-like route. The four sides have the same length.

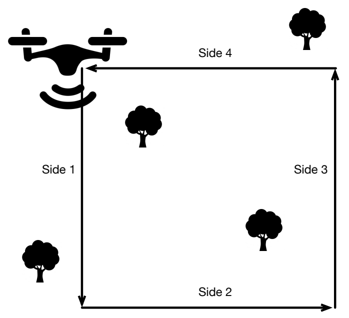

Fig. 1: Example of a drone flight path

Such a simple behavior can be the result of executing the following short program.

```
1  drone takeOff.
2    4 timesRepeat: [
3      200 timesRepeat:
4        [ drone goForward: 1 ].
5      drone turnLeft: 90.
6      ].
7  drone land.
```

Line 1 makes the drone take off and Line 6 makes the drone land. Lines 2 - 5 form a simple loop that four times repeat the sequence of moving forward by 200 meters and turning left by 90 degrees. The program is written in the Pharo programming language and we assume that the program is running outside the drone.

Use of continuation. Producing a resumable application is simple using continuation. Consider the program:

```
1  | continuation |
2  drone whenBatteryShortageDo:
3      [ Continuation currentDo: [ :cc | continuation := cc ] ].
4  drone whenBatteryShortageSave: { continuation }.
5  drone restoreVariablesIfNecessaryAndDo: [ drone takeOff.
6    ^ continuation value ].
7  drone takeOff.
8  4 timesRepeat: [
9    200 timesRepeat: [ drone goForward: 1 ].
10   drone turnLeft: 90.
11 ].
12 drone land.
```

Lines 6-11 are exactly the same than the original program. Lines 1-5 add the necessary logic to make the application resumable. Line 1 declares the variable `continuation`. This variable hold the continuation of the application. Lines 2-3 defines a callback to execute when the battery shortage occurs. The expression in Line 3 has the effect to capture the continuation `cc` and stores it in the variable `continuation`. Line 4 states that in case of a battery shortage, the variable `continuation` has to be saved (either in a flash card or by sending a command to a base). Once the battery is replaced, Line 5 will restore the continuation and evaluate it using a callback. Note that the callback exits the program after evaluating the continuation. Exiting is necessary since the continuation is a function that contains portion of the Lines 7 - 12. We also need to indicate the drone to take off before resuming the continuation, since the continuation was captured while the drone was flying.

[0]http://pharo.org

VI. Conclusion

A resumable behavior has the property to resume the overall behavior by simply restarting the program. This is an important property in case of a drone has to be replaced after a crash or a broken part.

Continuation is a light-weight mechanism offered by the programming language. We carried out our prototype with the Parrot drone 2.0 and the Pharo programming language.

Acknowledgments. We thank Renato Cerro for their comments on an early draft of this paper. We also thank LAM Research por partially sponsoring this research.

References

[1] L. Corral, I. Fronza, N. E. Ioini, and A. Ibershimi, "A measurement tool to track drones battery consumption during flights," in *Mobile Web and Intelligent Information Systems*, M. Younas, I. Awan, N. Kryvinska, C. Strauss, and D. v. Thanh, Eds. Cham: Springer International Publishing, 2016, pp. 334–344.

[2] C. Tseng, C. Chau, K. M. Elbassioni, and M. Khonji, "Flight tour planning with recharging optimization for battery-operated autonomous drones," *CoRR*, vol. abs/1703.10049, 2017. [Online]. Available: http://arxiv.org/abs/1703.10049

[3] S. J. Kim, N. Ahmadian, G. J. Lim, and M. Torabbeigi, "A rescheduling method of drone flights under insufficient remaining battery duration," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 468–472.

[4] K. Fujii, K. Higuchi, and J. Rekimoto, "Endless flyer: A continuous flying drone with automatic battery replacement," *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pp. 216–223, 2013.

[5] B. Michini, T. Toksoz, J. Redding, M. Michini, J. How, M. Vavrina, and J. Vian, ser. Infotech@Aerospace Conferences. American Institute of Aeronautics and Astronautics, Mar 2011, ch. Automated Battery Swap and Recharge to Enable Persistent UAV Missions. [Online]. Available: https://doi.org/10.2514/6.2011-1405

[6] M. Felleisen, D. P. Friedman, E. E. Kohlbecker, and B. F. Duba, "Reasoning with continuations," in *LICS*, vol. 86, 1986, pp. 131–141.