DroneCharge: A Python Framework for Automated Quadcopter Charging

Miroslav Zoricak
IT-University of Copenhagen
mzor@itu.dk

Kamil Androsiuk
IT-University of Copenhagen
kami@itu.dk

Emil Bech Madsen
IT-University of Copenhagen
aebm@itu.dk

ABSTRACT
(Something about drones and their usage), however, (something about battery life insufficient for many tasks). Our solution to the problem (the 'new' stuff, the framework), (something about what value this brings and how it could impact drone usage)

Author Keywords
Drone, swarm, automated charging, quadcopter, framework

INTRODUCTION
Aerial drones, such as four-propeller quadcopters or single-propeller mini-helicopters, have a lot of potential use, but the use of commercially available drones are limited by the short amount of time they can stay airborne before requiring a recharge. Many applications could be considered where a drone would need to stay airborne for long periods of time to do various tasks. For instance, monitoring areas using various sensors or spreading pesticide on a field of crops.

Although battery life will increase as technology advances, there is a current need for alternatives. If drones could recharge themselves or somehow become sustainable this would increase the usage scenarios of drones significantly. Our idea is not to have drones stay airborne indefinitely, but to have the tasks that the drones are performing continue in spite of the need for a recharge. This could either be done by having other fully-charged drones take over the task or by allowing the drone to resume its task after it has been recharged. In this report, we outline a framework aimed at simplifying this functionality for drone-application developers. We focus on quadcopters such as the AR Drone or the Crazyflie nano-copter, but see no reason this would not work on any aerial drone capable of stationary hovering such as toy helicopters.

RELATED WORK
Bürkle et. al proposed the idea of a swarm of drones fulfilling a task rather than a single drone in their article *Towards Autonomous Micro UAV Swarms* [1]. They suggested a system for inter-drone cooperation to seamlessly orchestrate several drones with different types of sensors in an environment. They also utilized simulation to test their system. This system was far larger than the produced solution and encompassed several other aspects that we did not consider. It provided with the idea of simulating drone movement during development for a much more efficient development process.

Sima Mitra presented an Autonomous Quadcopter Docking System [2] that allowed drones to return to a charging station using computer vision for locating the docking station. In it, she showed that autonomously returning to a charging station was fully possible without a fixed locationing system. Although our framework expects some kind of locationing system for the airborne drones, this was useful as it demonstrated the automated landing on a specified target as battery depletes. The inherent limits of the reliance on computer vision were, however, not desirable.

Both Altug et al. [3] and Ducard et. al. [4] demonstrated the use of visual feedback for positioning and autonomous control of a drone. This was useful for the practical part of this project in which a kinect was used to track drones in order to autonomously perform tasks.

Also relevant is the work of The Intelligent Mechatronics Lab at Boston University, in which a conductive surface was used to charge a drone without human interaction [5]. This is of paramount importance in terms of realizing this solution, as full automation would not be possible unless the drone could charge itself without human interference.

Finally, as an alternative to drone swarms, where battery issues are solved by an abundancy of drones, Abidali et. al. [6] offered an alternative through the use of solar power. In their report, they outlined the design of SolarCopter, the worlds first solar-powered quadcopter capable of sustained flight.

Bürkle et al. focused primarily on swarm usage and capabilities, but mentioned very little about how the actual replacement of the drones would work. We left the execution of a task as an extensible abstract concept, and specifically focused on providing automatic drone-swapping to implementers of these tasks while including implementation of basic tasks such as movement and landing.

Ducard et. al. and Altug et. al. both showed that drone control through the use of a visual feedback system was possible, but the actual execution of tasks were highly specific and non-extensible for the common developer. We aimed at allowing developers using visual feedback for position and movement to easily be able to perform tasks.

Sima Mitra focused on computer vision to return to a charg-

ing station. We noted that there are inherent problems in Computer Vision such as the need for an unobstructed view of the intended target, and argued that more absolute positioning schemes such as GPS or external tracking of a drone would be more suitable, especially for swarm-based task execution in which the system needs a complete view of the operating area to simoultaneously use several drones.

While the work of Boston University indeed showed that conductive charging of aerial drones was possible, we primarily focused on getting a drone to the charging station, and left the addition of conductive charging as a means to further automate task execution efforts.

The SolarCopter provided an alternative to the drone swarm, but adds complications in regards to drone size and weather requirements. One could imagine a mixture, however, in which a solar powered drone could charge conventional drones and act as a swarm staging area.

Figure 1 highlights the features of the various solutions. While we did not attempt to achieve sustainable drones, we, to some degree, dealt with all other application areas that are addressed by the mentioned solutions.

	Bürk.	Mit.	Alt.	Duc.	Bost.	Abi.	DC
Visual control		✓	✓	✓			✓
Auto-landing	✓	✓					✓
Tasks	✓						✓
Extensibility	✓						✓
Cooperation	✓		✓	✓			✓
Simulation	✓						✓
Auto-charging		✓			✓	✓	✓
Sustainable						✓	
Autonomy	✓	✓	✓	✓	✓		✓
Low Resource		✓	✓	✓	✓		✓

Figure 1. Feature comparison

METHODOLOGY

The project was developed in two parts; a practical part and a theoretical part. The theoretical part consisted of a simulation of an *ideal* drone, with which the framework was incrementally developed. The practical part consisted of a visual locationing system using Microsoft Kinect[[ref]], with which a quadcopter drone was controlled using the native SDK. Initially, a Crazyflie drone was used for the practical part, but after some difficulties with stability, it was decided utilize a Wizard-of-Oz approach for drone movement. This allowed us to focus on the framework itself rather than implementation issues of a single type of drone. In the later stages of the project, we were offered the use of an AR Drone[[ref]], but decided to leave it as an experiment should time permit it.**[[Change this if we actually try the AR drone]]**

While others such as Mitra and Ducard et al. before us had attempted to do various parts of this project in isolation, DroneCharge brought together the components in order to create a practical solution on a more approachable level.

The swarm concept presented by Bürkle et al. also utilized many of the same concepts, but in a very large-scale and hardware-expensive fashion out of reach of the common developer. We relied on the various proof of concepts shown by our predecessors and envisioned a system in which the various components were useful without being out of reach for the average developer in terms of hardware and time.

DRONECHARGE

Vision

DroneCharge was envisioned as a framework to be used by drone application developers to easily achieve task-execution using swarms of drones, and as such needed to fit any and all suitable drones. As all drones have their own native APIs for control, this meant that a driver-oriented architecture had to be utilized. While the drivers provided us with means of controlling the drone, we also needed a way of hooking into the tasks performed by the drones in order to seamlessly exchange drones when rquired. This was done by having developers define tasks in terms of a series of subtasks, and to have our framework execute those tasks in the right order. That enabled us to hook into the execution of a task and to exchange drones when a recharge was needed.

Architecture

Our overall architecture is depicted in Figure 2. The most interesting concepts are the ideas of drone, task and environment. A task is performed in an environment that has a number of drones at its disposal, all of which are configured by the individual developer. As a developer, you would implement drivers for your specific drone that inherits from the drone class, and either utilize the built-in tasks or create your own extending from the task class. A drone instance representing each physical drone is then added to the environment along with their respective charging locations, as well as any number of tasks to be performed.

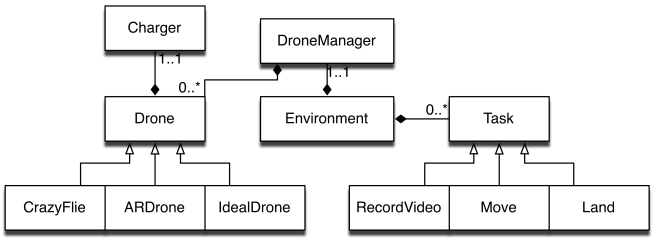

Figure 2. DroneCharge architecture

Drivers

Drivers are, as is the definition, a way for the system to interact with the hardware devices connected to it without knowing the hardware specifications at compile-time. In our case, those hardware devices are drones and their locationing system. Any developer using DroneCharge must implement a class extending from the Drone-class that implements neccesary features such as movement as well as battery- and location-queries. These are called drone-drivers. Whether a drone has positioning built-in or if it uses a tertiary system

such as tracking with a Microsoft Kinect is irrelevant, as long as the drone-driver has the ability to obtain its position.

Tasks

Tasks define how individual steps of a task are performed. They contain a list of capabilities required to perform the task such as movement or video-recording, as well as instructions on how to perform the task. DroneCharge comes with a limited number of built-in tasks such as movement and landing, but any task can be performed by extending the task class and explicitly performing the task through the commands defined in the drone-drivers. The framework will only assign drones to perform the task if they meet the requirements for it, so you are guaranteed to have the right type of drone as long as your task and required capabilities are aligned. As an example, a custom driver for a drone with a video camera attached to it could implement a function called *RecordVideo* as well as contain the capability *record-video*. You could then define a task whose execution consisted of turning on a video camera. By specifying *record-video* as a required capability of the task, the framework would not perform the task until a drone with said capability was available.

The Task Tree

By allowing tasks to be added to other tasks, we obtained a tree-structure view of tasks which enabled us to do a depth-first search traversel of tasks, and in that way achieve a sound logical structuring of tasks and subtasks. The tree is traversed in such a way that only leaf-node-tasks are actually performed. This enabled developers to logically split their tasks into subtrees however they saw fit and gave us increased flexibility.

[[Image of graph]]

Assumptions and requirements

When using the DroneCharge framework, developers had to be aware of the assumptions and requirements made in terms of the capabilities and implementation of the drivers and the drones themselves.

For the drone drivers, we assumed that a drone or its driver could report its battery level in a linear fashion. That is, the battery could not be reported as full until the battery was depleted as was the case with some drones such as the Crazyflie. Furthermore, there had to be some way to obtain the position of the drone in relative or absolute terms. A developer also had to define at what battery level the drone was considered to be low on battery. It was also required that a charger was available for each individual drone.

In terms of defining tasks, we made the requirement that any single (sub)task could not require more battery to perform than was considered to be the low battery limit. That is, if the drone was considered to be at low battery when it reached 10%, all individual subtasks had to require no more than 10% power to perform. Finally, the point of operation that was the farthest from the charging station of the drone was required to be able to be performed utilizing less battery power than was defined as being the low battery level.

Additionally, it was noted that a drone partaking in the execution of a task had to have all capabilities needed to perform all subtasks in the task tree in which the current subtask is located. This is upheld by the framework itself, as it will not choose drones lacking any capabilities. This ensured that we would not have to exchange drones due to capabilities but only due to battery levels.

EVALUATION

[[Here we'll iterate over the design of the framework through the experiences gained in the sample usage, as well as note the knowledge gained from the questionnaires posted on online drone-forums. We'll look at what we did but leave discussion of what could have been done differently/better to the discussion section.]]

Developer survey

To gauge whether this would be useful for real-world developers, we posted a survey along with a description of the system on several of the largest drone-development internet forums. Although only a limited amount of developers replied, five in total, we felt comfortable that the level of interest was adequate.

When asked if the proposed framwork seemed useful, three developers replied that they either agreed or strongly agreed, one replied neutrally and one developer disagreed. When asked whether they had ever faced implementation issues that this framework would solve, three developers were neutral while one developer agreed and another strongly agreed. Finally, when asked if they would be interested in using such a framework, three developers agreed, one developer strongly agreed and only a single developer disagreed.

One developer also noted that our suggested way of positining the drone was inadequate, as drift would influence its absolute position. This caused us to reconsider our positioning and movement methods.

All in all, we believe it is safe to say that the interest in automatic charging of drones is present, although responses may have been skewed by the possibility that only interested developers replied to the survey.

The experiment / Usage example

Intial Experiments - The Crazyflie

In the initial stages of this project, we experimented with developing drivers for the Crazyflie Nano Quadcopter, using the Microsoft Kinect for locationing. These efforts provided us with valuable knowledge on the current state of art in regards to drone capabilities. Specifically, while the Crazyflie in and of itself was a suitable drone for freestyle flying, its rather limited control mechanisms and lack of hovering function severely impeded its use for the DroneCharge framework. It was very difficult if not impossible to make it land close enough to a charger for any real use, and task

execution was unsafe at best. On a more positive note, it also gave us confidence that our driver-oriented architecture was correct, and helped us define the granularity of the interface between the framework and the drone by giving us some experience with operating drones programatically. Combined with the feedback from the developer in the conducted survey, we changed our approach to drone-positioning and control.

Simulated approach - Wizard of Oz

Due to time constraints, it was decided that we would not pursue a second type of drone, even though an AR Drone became available at a later stage. Instead, we focused on developing the framework and relying more on simulation and visualization for validation. We created visualization tools to illustrate the task tree as a coloured graph, and created usage scenario in which a Microsoft Kinect tracked a drone. Rather than the drone moving itself, a Wizard of Oz approach was used in which a human actor purposefully moved the drone in the pattern indicated by the framework in order to perform the task. This was done using the video-feed from the Kinect and adding overlays with markings for the drone and the target position.

[[Image of Kinect overlay]]

DISCUSSION

Here, we'll discuss what could have been done differently to improve upon the design. If we experience any major setbacks that are not alleviated in time, this is the section in which we explain what it would require to produce the desired results – such as using other type of drones for the usage example.

CONCLUSIONS

Here, we conclude upon the results and suggest future work.

ACKNOWLEDGEMENTS

Here, we acknowledge the work and assistance that we utilized in order to write this project.

REFERENCES

1. Axel Bürkle, Florian Segor, Matthias Kollmann, *Towards Autonomous Micro UAV Swarms*. http://link.springer.com/article/10.1007%2Fs10846-010-9492-x, 2010 (Accessed Oct. 2013).

2. Sima Mitra, *Autonomous Quadcopter Docking System*. http://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2012to2013/ssm92/ssm92_report_201305171020.pdf, 2013 (Accessed Oct. 2013).

3. Erding Altug, James P. Ostrowski, Robert Mahony, *Control of a Quadrorotor Helicopter Using Visual Feedback*. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1013341, 2002 (Accessed Oct. 2013).

4. Guillaume Ducard, Raffaello D'Andrea, *Autonomous Quadrotor Flight Using a vision System And Accomodating Frames Misalignment*. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05196224, 2009 (Accessed Oct. 2013).

5. Boston University Intelligent Mechatronics Lab, *Autonomous Quadcopter Charging Station*. http://www.bu.edu/iml/2012/08/02/221/, 2012 (Accessed Oct. 2013).

6. Aly Abidali, Jibran Ahmed Shakir Ahmed, Irmantas Burba, Pourshid Jan Fani, George Kowfie, Kazimierz Wojewoda, *SolarCopter*. http://www.qmul.ac.uk/media/qmnews/items/91251.html, 2013 (Accessed Nov. 2013).