Drone-Megaddon: A Real-Time Strategy Interface for Managing Drone Swarms

Sairam Krishnan
Department of Electrical and Computer Engineering
Carnegie Mellon University
sbkrishn@andrew.cmu.edu

Daniel Ting
Department of Electrical and Computer Engineering
Carnegie Mellon University
danielti@andrew.cmu.edu

ABSTRACT

Neural sensors, Oculus Rift, mobile devices - to date, drone researchers have utilized a variety of different media to control a single drone. However, to the best of our knowledge, there are currently no adequate solutions for dealing with multiple drones at a time. So, in this paper, we describe Drone-Megaddon, an Android-based real-time strategy interface patterned after the hit RTS game StarCraft II, which facilitates high-level management of a drone swarm.

Keywords

Wireless Sensor Networks, Cyber-Physical System, Quadrocopter, Drones, RTS, Swarm, Zigbee, StarCraft II, User Interface

1. INTRODUCTION

In the past decade, the space of drone applications has expanded tremendously, encompassing tasks such as search-and-rescue or even concerted music performance [1]. Swarm intelligence in particular has been touted as an area of strong potential, inspired by biological phenomena such as insect swarming. Despite this surge of interest, few interfaces support multi-drone control.

To address this problem, we look to another, very different area of computing for inspiration - video game development. Real-time strategy (RTS) game engines feature efficient top-down control of up to hundreds of units at a time. Advanced AI pathfinding algorithms have been instituted for collision avoidance, and by using the simple "select-execute" paradigm of RTS interfaces, gamers have reached performance levels in excess of 300 actions-per-minute (APM) [2]. We see four key advantages to using an RTS interface for coordination of multiple agents:

- High-level display and quick selection of agents
- Intuitive waypoint setting and support for potential pathfinding algorithms
- Clear indication of unit status through health bars and other per-unit graphics
- Fast action execution using command bar

These features, coupled with the familiarity of this interface, makes this an attractive model for a drone control UI.

This paper presents Drone-Megaddon, an RTS-style drone interface that meshes together the real-life control of physical drones with a virtual interface inspired by the RTS game Starcraft II. It is a "cyberphysical" system in the purest sense of the word. Because of the physical component, this fusion comes with a high emphasis on backend processing, which we discuss at length in this paper.

The rest of this paper is organized as follows. Section 2 provides a brief summary of existing applications for drone control. In Section 3, we provide an overview of this system's hardware and software components. Afterwards, in Section 4, we describe the system's architecture and end-to-end flow-of-control; specifically, we discuss the major components and how they work together. Section 5 follows with a detailed description of the four diverse software architectures within this system. Finally, in Section 6, we consider our app's limitations and possible future work.

2. RELATED WORK

To our knowledge, the control of multiple quadrocopter-size drones through an RTS-style user interface has not been previously implemented. A large roadblock to controlling multiple drones from a single platform has been the issue of opening multiple connections. Some drone enthusiasts have surmounted this problem, [3, 4] but as

CPS'14, December 3–5, 2013, Pittsburgh, PA.

of yet, none have taken the next step of creating user interfaces for multi-drone management. At the opposite end of the spectrum, several corporations and startups have created mobile tablet or phone apps geared towards flight planning for single drones.

One such project was developed by Sensefly, a subsidiary of Parrot. Geared towards agricultural mapping, the eBee Ag drone was commanded by an application with support for takeoff/land commands, return to static home loctions, and complex waypoint settings (including circular row traversal for maximum coverage). The application was highly developed, but did not have fully dynamic control, as paths had to be first preplanned and then executed, albeit with support for pause/resume [5].

Another similar interface was designed by the creators of the popular DJI Phantom drone, titled simply "Ground Station". Ground Station, developed for the iPhone and DJI Phantom Vision 2, focused primarily on setting up trajectories for scenic camera capture. As such, it set waypoints on a Google Maps-style backdrop, accepting parameters such as speed and altitude. After waypoints were set, a "GO" command could be issued and the drone would make its circuit, passing real-time control of the camera angling to the operator [6].

Even more intriguing, and perhaps frightening, was the Ballista system developed by DreamHammer, a recent startup based in Los Angeles [7]. This game-like interface enabled control of multiple military-class drones for tasks such as fire-fighting, search, and potentially, guided missile attack. Although it had support for real-time actioning, it appeared to rely mainly on preplanned mission schedules. The complexity of commands offered meant that users would likely need to click through several options before being able to execute an action.

The preceding interfaces illustrate well the potential of a fully-fleshed out drone UI interface. However, where they fall short is real-time control - enabling the user to specify any movement at any time. By adopting the model of a real-time strategy interface, Drone-Megaddon lays the framework for responsive, fluid action issuing.

3. COMPONENTS

3.1 Hardware

- 1 Samsung Galaxy S5 (OTG-enabled)
- 1 USB-to-USB cable (compatible with Galaxy S5)
- 2 Firefly nodes
- 1 Piksi GPS Module
- 1 AR Parrot Drone

3.2 Software

- Android OS 4.4.2, KitKat
- Google Maps API v2
- Physicaloid, a USB-Serial Library for Android
- Nano-RK
- Drone-RK

4. ARCHITECTURE

4.1 Connections

- The drone and the Android device are connected to their respective Firefly nodes by a USB-serial connection.
- The Piksi GPS module is connected to the drone's Firefly by a UART connection.
- The two Firefly nodes wirelessly communicate with each other via Zigbee.

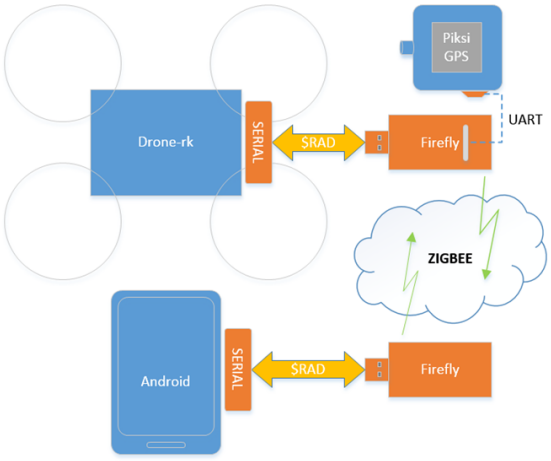

Figure 1: Architecture Diagram

4.2 Android device sends radio command to drone.

The user can specify various commands through the Android RTS interface. Here, we shall discuss the most fundamental command, FLY. The user can select 1 or more drones on the Google Map and tap on a specific point on the map to command them to move there. Consequently, the application extracts the target coordinates and applies a random offset to them for each drone, so that the drones don't all end up colliding at the destination. In the event that there is only one drone, then the application simply leaves the target coordinates as is. Afterwards, the application packages the coordinates into a $RADFLY command, which is written to the serial connection between the device and the Firefly node. The Firefly node duly transmits the $RADFLY command via ZigBee to the drone's Firefly.

4.3 Drone receives radio command from the Android device.

The Firefly attached to the drone receives the radio message from the Android device and forwards it to the drone. On the drone, a receiving thread reads the incoming information into a buffer, identifies its remote origin, and executes the appropriate command using the received parameters. For example, in the case of $RADFLY, the receiving thread uses $RADFLY, the first parameter in the incoming data, to identify that this message is a radio command. It then extracts the GPS coordinates and the altitude and forwards the information to the appropriate drone-rk library method for moving the drone to a specified target.

4.4 Drone receives GPS coordinates from the Piksi GPS module.

The Piksi GPS module regularly sends the latest measured GPS coordinates to the drone through the Firefly node. Consequently, the receiving thread mentioned in the previous subsection updates an internal variable tracking the drone's current location. This variable is used as a part of the drone's control logic when it is moving towards a waypoint.

4.5 Drone transmits GPS coordinates to the Android device.

The internal GPS variable mentioned above is also used to transmit the drone's latest GPS coordinates to the Android device. The Android device uses this information to track the drone's progress towards the waypoint. If the drone has successfully reached the waypoint, then the device displays a toast message indicating successful completion of a previous $RADFLY command. Note that the Android device also receives the altitude and battery information along with the GPS coordinates, which is used to update other parts of the UI.

5. DESIGN AND IMPLEMENTATION

5.1 Android: The Communication Layer

Our Android app consists of a single Activity that creates a CommunicationServer and MapService upon creation. The CommunicationServer handles all tasks related to serial communication with the Firefly while the MapService handles all UI-related tasks. Figure 2 shows the high-level structure of this Android Activity.

The CommunicationServer spawns 2 threads: a Transmitter for sending commands to the drone and a SerialCommunicationReader for receiving commands from the drone.

The former polls a queue of TxCommands within the CommunicationServer every 100 milliseconds; if a TxCommand is in the queue, the Transmitter writes the command to the Firefly, which then transmits the message to the drone. Buttons in the UI bar and UI-event listeners in MapService queue TxCommands to induce the drone to behave accordingly; for example, placing a waypoint on the map queues a FlyCommand on the CommunicationServer's TxCommand queue to get the drone to fly to that waypoint.

The latter continuously monitors incoming messages from the drone (via the USB-serial connection to the Firefly), passes them on after construction to the RxCommandFactory, and queues the resulting command on the MapService's RxCommand queue. The MapService contains a thread handler that polls the RxCommand queue every 100 milliseconds; if the queue is not empty, the first command in the queue is popped and executed to induce the appropriate UI response.

The biggest advantage of this structure is modularity - the serial communication and UI tasks are highly decoupled, as are the TxCommands and RxCommands (transmission and receive commands). The phone currently only receives GPSCommands; however, in the future, if we wanted to add another type of RxCommand, we would simply create a class extending RxCommand and specify what MapService UI method it should invoke (what UI action to perform). Similarly, if we want a new type of TxCommand, we would simply create a new child class of TxCommand that can construct a radio message for the drone using some arbitrary input parameters. Finally, future changes in the UI (such as moving from the Google Maps API or changing event-listeners) will not cause us to revisit the communication layer.

5.2 Drone-RK

Originally, we attempted to handle all communication between the drone and its attached Firefly using the Serial-Line-Interface-Protocol (SLIP). SLIP functions by using special START, END, and ESC delimiter characters, processing them in what is essentially a finite state machine. However, we were unable to resolve the numerous issues that we encountered with SLIP, especially due to the lack of documentation. The protocol also proved to be overly complex for the purposes of our application, so we have transitioned to our own ASCII protocol, described below.

Our ASCII protocol consists of comma-delimited messages, where the first field indicates the message type and command. If the data is just GPS information from the Piksi, then the first field will be $GPGGA (defined by the Piksi's manufacturer). Otherwise, if the data is a radio message received from the phone's Firefly, then the first field contains the prefix $RAD, followed by three characters that specify the command type. For exam-

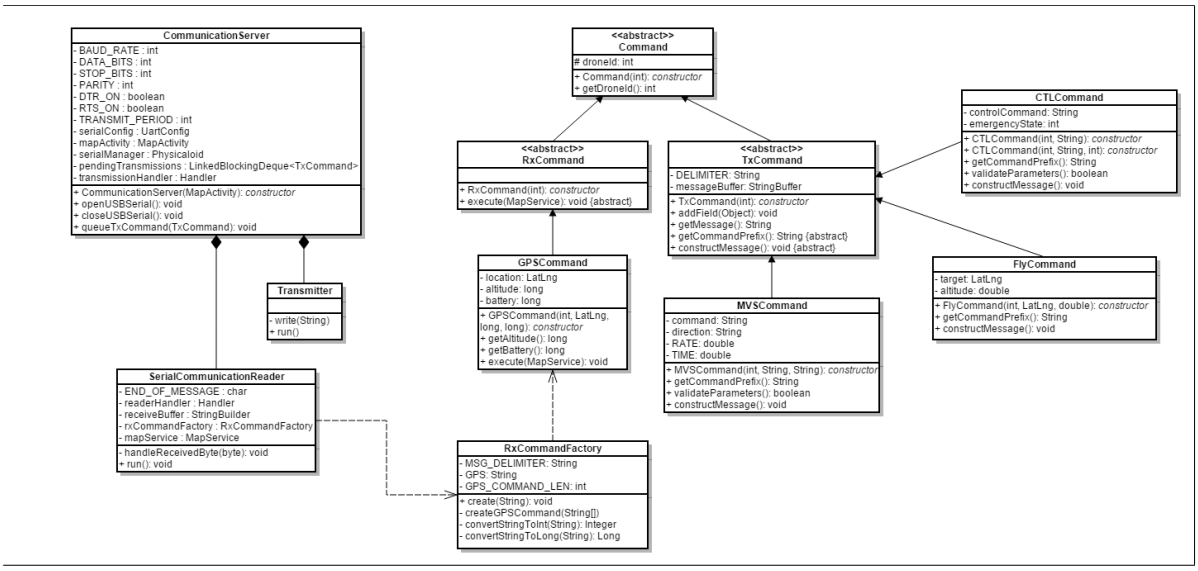

Figure 2: UML Diagram of Android Application's Communication Layer

ple, if the phone transmitted a FlyCommand, then the drone would receive $RADFLY in the first field. Immediately following the prefix is the drone id field, which enables the Android device to specify which drone to control and the drone to disclose its identity to the Android platform. Subsequent fields are message-specific, and drone commands originating from the Android platform are grouped according to their input parameters.

As of now, one drone-originated message type is supported, $RADGPS. It includes latitude and longitude, converted from fixed-point floats to strings. Since the battery capacity and altitude information are updated along with position on the Android UI, they are included in the payload to reduce message-passing overhead.

Drone-directed commands originating from the Android platform are grouped according to their input parameters. $RADCTL commands are control messages that take no parameters. Under $RADCTL, mission-critical takeoff, landing, and emergency cutoff commands are specified. The $RADFLY command instructs the drone to move to a target specified by GPS coordinates. $RADMVS bundles together move and spin commands, which take the direction, a rate at which to execute the command, and how long the movement should execute. MOVE commands the drone to move in four directions on the xy coordinate plane, as well as up and down the z-axis. SPIN accepts either clockwise (right) or counter-clockwise rotation in the xy-plane. Lastly, $RADHOV is used to make the drone hover for a specified number of milliseconds.

After the message payload has been transmitted, the transmitter signals the end of transmission by sending a special end byte of 255.

The full list of supported radio messages is shown below, followed by packet structure in Figure 3:

1. *$RADGPS*
   - GPS messages received by the Android device from the drone.
2. *$RADFLY*
   - command drone to fly to the specified waypoint.
3. *$RADCTL*
   - command to drone to takeoff, land, or enter/exit emergency state. To enable emergency, specify EMERGENCY for the third field and 1 for the last field. To disable emergency, specify EMERGENCY for the third field and 0 for the last field.
4. *$RADMVS*
   - command the drone to move/spin in a direction.

To complicate matters slightly, there is an extra component to the messages transmitted from the Firefly to the drone - a 3-byte header containing header version and a sequence number. Every time the drone sends a message to the attached Firefly, it expects an "ACK" message in response so that it can verify successful message delivery. If the Firefly receives the serial message for forwarding over Zigbee, it sends an "ACK" containing 'Z', followed by the sequence number. It then discards the header and sends the payload over Zigbee. If the "ACK" is not received on the drone side, further communication is aborted. The ack message consists of a sequence number; the drone checks for this sequence number within an internal acknowledgement buffer. If it finds the number, then communication can continue. If it doesn't find the sequence number, then the drone knows that it has drifted out of sync with the Firefly, so messages have been dropped and the link's reliability

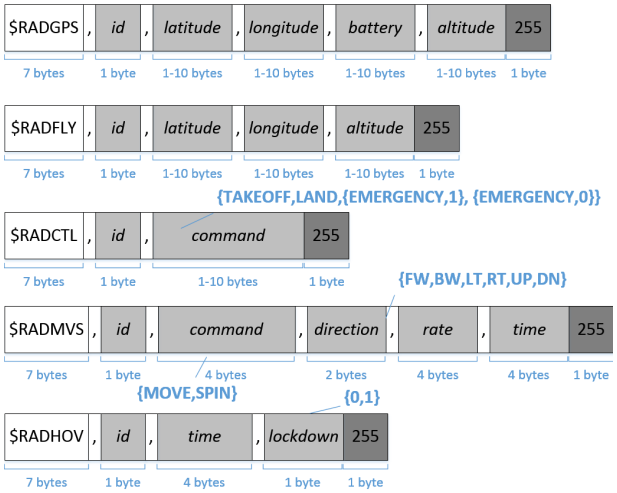

Figure 3: $RAD Protocol Packet Structure

has been compromised. Essentially, the acknowledgement buffer is a way of enforcing a tolerance range - as long as the sequence number falls within that range, there are supposedly no issues.

5.3 Drone Firefly's Nano-RK

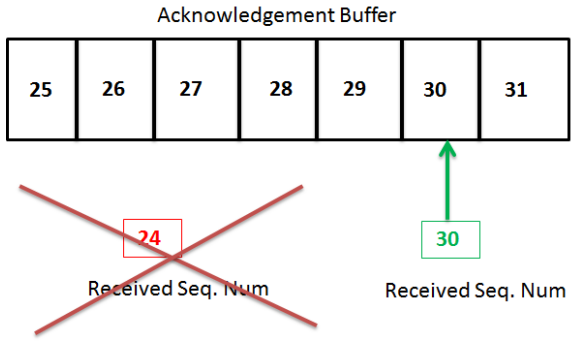

Figure 4: How Ack-ing Between the Firefly and Drone Works

The drone Firefly's Nano-RK firmware spawns three threads - one to receive radio messages from the phone's Firefly and transmit them via USB-Serial to the drone, one to send the drone's messages to the phone's Firefly, and one to forward the Piksi's GPS coordinates to the drone via the USB-Serial connection. These three tasks are called Task_RfReceive, Task_RfSend, and TaskGPS respectively. Note that Task_RfSend has to send an "ACK" back to the drone to acknowledge successful receipt of its message; this ack message consists of a sequence number between 0 and 254 inclusive. As a part of this process, the Firefly maintains its own sequence number counter, which will roll over to 0 after 254. Note that 255 is the terminating byte for each message, which is why we only iterate up to 254.

5.4 Android Device Firefly's Nano-RK

The Android device Firefly's Nano-RK is very similar to the drone's; the only differences are that there is no TaskGPS function and that the Firefly does not need to send an "ACK" message to the Android device.

6. CONCLUSION

In this paper, we presented the first-ever RTS-style drone control interface that supports command of multiple drones, over Zigbee radio. Drone-Megaddon is a 2D, top-down control system that grants users speed, flexibility, and reliability. Unlike existing drone apps for aerial camera shooting, the Drone-Megaddon system is a general framework, modularly extensible to many potential applications. We hope that the features in this work will lay the groundwork for other systems that will further blur the lines between RTS games and drone swarm control.

6.1 Limitations

Currently, Drone-Megaddon's biggest limitation is that multiple selected drones can be commanded to fly to a single waypoint; a future enhancement would involve adding a small (possibly random) offset to the target location for each drone and adjusting each drone's velocity.

Another limitation of our application is that it does not indicate whether a particular drone has crashed mid-flight; the drone's icon will simply be shown at the last received GPS coordinate on the map. One way to mitigate this would be to compare the drone's current GPS coordinates to the user-specified waypoint and trigger an error message if there is a mismatch for more than N seconds. It is worth noting that it might also be useful to show the drone operator's location on the map so that he/she knows which direction to search for a downed drone.

Lastly, unlike the $GPGGA message supplied by the Piksi, our protocol does not employ an NMEA checksum to verify message integrity. The current scheme assumes that the channel is mostly free of noise, corresponding to a low packet error rate. For more robustness, a checksum could be added without much difficulty to enable rejection of corrupted packets.

6.2 Future Work

Besides addressing the aforementioned limitations, we think that it might be advantageous to make the UI's flight controls more intuitive by connecting them to the Android device's sensors. For example, the user currently needs to click on a button in the UI bar to command the drone to spin left. Instead, it would be more

interesting to spin the drone based on the direction that the user rotated the Android device, which could be obtained from the device's gyroscope.

For better diagnostics, message management, and security purposes, timestamping seems to be a logical step forward for the current radio protocol. With just a small increase in overhead, a timestamp field could be appended to each message, enabling receivers on both ends of the connection to make decisions based on freshness or ordering of messages received.

Now that the infrastructure is in place, we could also explore how this application can be used to coordinate a drone-delivery service. For instance, a single operator can now manage a drone delivery fleet for a neighborhood rather than a single drone alone. This raises some interesting challenges in terms of CRM (customer-relationship management) and GPS connectivity.

6.3 Acknowledgments

The authors would like to thank Dr. Anthony Rowe for the original concept behind Drone-Megaddon, and the 18-848 TAs for their practical lab tutorials on hardware/software integration. We would also like to extend a special thanks to Luis Pinto for his invaluable help with the codebase, hardware boards, and communication debugging.

7. REFERENCES

[1] Heater, Brian. Please enjoy this video of dancing drones. http://www.engadget.com/2014/01/08/dancing-drone/, 2014. [Accessed: 2014-12-15].

[2] Wong, Kevin. StarCraft 2 and the quest for the highest APM. http://www.joystiq.com/2014/10/24/starcraft-2-and-the-quest-for-the-highest-apm/?ncid=rss_truncated, 2014. [Accessed: 2014-12-15].

[3] Backus, John. Programming Multiple Parrot A.R. Drones on One Network With Node.js. http://drones.johnback.us/blog/2013/02/03/programming-multiple-parrot-a-dot-r-drones-on-one-network-with-node-dot-js/, 2013. [Accessed: 2014-12-14].

[4] Vaughan, Richard. AutonomyLab. https://github.com/AutonomyLab/ardrone_autonomy, 2012. [Accessed: 2014-12-14].

[5] Backus, John. senseFly: eBee Ag. https://www.sensefly.com/home.html, 2014. [Accessed: 2014-12-14].

[6] Phantom 2 Vision+ | DJI. https://www.sensefly.com/home.html, 2014. [Accessed: 2014-12-14].

[7] Ballista - DreamHammer. http://www.dreamhammer.com/ballista/#ourstory/uc-1-1, 2014. [Accessed: 2014-12-14].