HAWKS

Hovering Aerocraft with Kinetic-Motion Sensing

Eric Chiang

A-Tion Zhuo Li

Amy Lin

Byron Lutz

Junda Zhu

# 2. Abstract

HAWKS is a smartphone application that allows a user to control the movement of a drone by moving the device itself. The finished application, developed for Android, uses data from the device gyroscope and accelerometer to detect tilt or acceleration. The device calculates the direction in which the drone should move and sends the command over a Bluetooth connection to a laptop. The laptop then sends commands directly to the drone over 802.11. Instead of using the gyroscope and accelerometer, commands for takeoff, landing, and up/down movement are initiated by buttons on the app UI and are sent to the drone in the same way.

# 3. Introduction

## 3.1 The idea

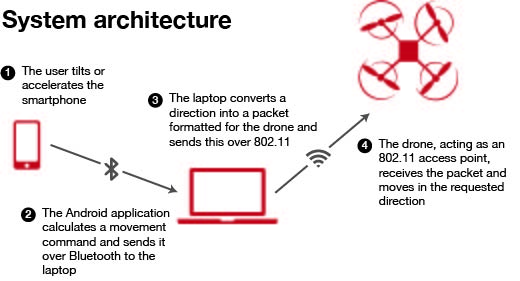
Even though there is already a selection of smartphone apps that control a drone’s flight movement, they all used some sort of on-screen UI (buttons, sliders, etc.) to control the basic movement. HAWKS was built to be more intuitive than that, converting a user’s smartphone into something akin to a joystick.

When the phone is tilted in a particular direction, the drone should move in that direction. This is much easier (and more fun) than using buttons on a touchscreen.

Once this control system is made, applications can be built to use this new type of input.

Besides making the drone movement work properly, another major goal of HAWKS was to implement and correctly use wireless communication protocols effectively.

## 3.2 Overall architecture



To accomplish the goals, the system was designed with a laptop relaying the phone’s commands to the drone itself. This increased the importance of wireless communication in the project and allowed the drone to be controlled from both the laptop or the phone.

This increased complexity, making the system much less portable and more difficult to set up, as the smartphone will have to stay within a Bluetooth communication range of the laptop and the laptop will have to stay within an 802.11 communication range of the drone. To follow the drone’s movement across a few hundred meters, for instance, the laptop and the phone will both have to be moved without breaking any part of the chain.

During the final discussion, points were raised about the architectural choice of using Bluetooth. Because the laptop has to communicate with the drone and the Android device simultaneously, two channels of communication are needed. These two channels were provided by having the Android device send commands to the laptop via Bluetooth then having the laptop send out the commands via WiFi.

## 3.3 Drone

Because of the modularity of the Android application, laptop software, and drone command processing, HAWKS can easily be implemented for a different type of drone, as long as accepts movement commands as up/down, left/right, and forward/backward.

For the initial demo of HAWKS, a Parrot AR Drone 2.0 was used. This drone was selected primarily because it was easily available. It also has a feature set that makes it appealing to this project:

* It functions as an 802.11b/g/n access point.
* It has an emergency-stop feature that will prevent damage to the drone and prevent it from flying away if the software malfunctions.
* It accepts a variety of simple commands (movement, takeoff/land, flip, etc.) without much extra programming.
* Interfacing with the drone is relatively easy, with an open SDK released by the manufacturer.
* Though not used in this project, the drone has a high-definition, front-facing camera and a standard-definition, ground-facing camera.

## 3.4 Controlling the drone

The laptop connects to the drone using Node.js and an open-source library that properly formats communications with the drone.

After getting a requested direction or command from the smartphone, the Node.js application sends the appropriate command to the drone. The drone acts like a state machine, retaining all commands until either (a) a “stop” command is issued to clear all commands or (b) conflicting commands are issued. In the latter case, the last-received command “wins” and is executed.

To simplify the implementation, the “speed” of all commands is fixed. This means that the amount the phone is tilted or accelerated does not change how fast the drone actually moves, it only issues a move command. (This would be similar to operating a car with a binary control, like a lightswitch, instead of a continuous control, like the gas pedal.)

The commands available to send are summarized in the table below.

|  |  |
| --- | --- |
| **Command** | **Description** |
| Take off | Start motors and fly up to an altitude of about 1 meter. |
| Land | Ignore further commands, slowly lower to the ground, and stop motors. |
| Front | Move forward |
| Back | Move backward |
| Left | Move left |
| Right | Move right |
| Counter clockwise | Rotate counterclockwise |
| Clockwise | Rotate clockwise |
| Up | Increase altitude |
| Down | Decrease altitude |
| Stop | Clear all commands that are currently being issued and wait for further instructions |

## 3.5 Bluetooth design

Bluetooth was chosen to communicate between the smartphone and the laptop because a user would be expected to stay close to the laptop. In addition, Bluetooth interfaces are common on both smartphones and laptops, ruling out a more obscure interface like ZigBee. Though WiFi Direct is also very protocol and has a longer range, most laptops only have one 802.11 card and thus can only connect to one WiFi network at a time. Since WiFi is needed to connect to the drone itself, another wireless interface was needed.

The actual Bluetooth communication has two components: a sender and a receiver. For the sake of simplicity, both were written in Java since Java is the preferred language for Android development.

The Android device is configured to connect over Bluetooth to the MAC address of the laptop. Then, whenever a command is determined from a button press or movement, a command is sent over this connection. To interface with the smartphone’s Bluetooth interface, the operating system’s default API is used (android.bluetooth.\*).

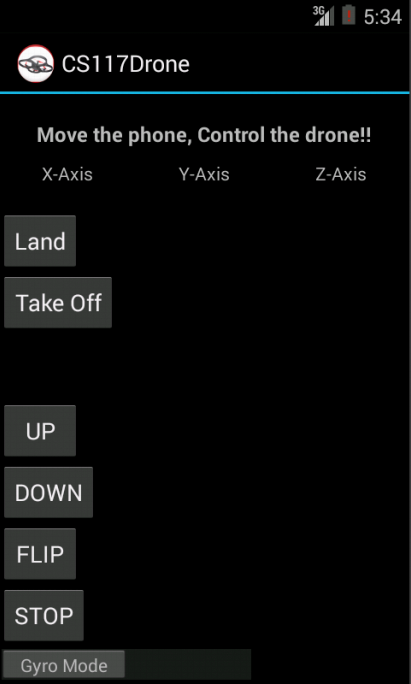
On the laptop side, a Bluetooth server is created and waits for a connection from the smartphone. An open-source library, BlueCove, is used to run and manage the Bluetooth server. This library makes the HAWKS code portable across major operating systems and allows the code to interact with Bluetooth with a Java-style stream.

Integrating the Bluetooth and drone control layers was a challenge — the Java application needs to send its output to the Node.js server. Because Node.js won’t accept this input from a pipe, a local port (8888) was made. When the Bluetooth server receives a connection, instead of just printing it to standard output, it prints it to the port. When the Node.js application listens on this port, it can successfully receive and process the command.

## 3.6 Android app

The Android app was one of the most complex parts of the system. It must present a UI on the screen, process gyroscope/accelerometer sensor data, connect to the laptop over Bluetooth, then send the appropriate command.

### 3.6.1 User interface



The interface provides all the controls needed for moving the drone. When the app is run on a phone with a gyroscope/accelerometer (the screenshot above is from an emulator without motion sensors), the direct readings from the sensor will appear below the “X-Axis,” “Y-Axis,” and “Z-Axis” labels. This made debugging easier and gives a hint as to how the raw data is being processed.

When the smartphone detects a movement, it displays “LEFT,” “RIGHT,” “FORWARD,” or “BACK” on the screen so the user can know if the phone interpreted the movement correctly.

To flip the drone, the user can physically flip the device or can press the “FLIP” button.

### 3.6.2 Processing sensor data

One of the biggest challenges of developing the app was interpreting the gyroscope/accelerometer data. At the bottom of the UI, there is a switch that can change the device from “Gyro Mode” to “Accel Mode.” The former uses data from the device’s gyroscope to detect movement, while the latter uses data from the device’s accelerometer to do the same.

Theoretically, both can have a similar effect. A gyroscope measures rotation while an accelerometer measures acceleration. With a constant gravity vector (like on Earth), the acceleration due to gravity can be used to determine rotation of the device.

In HAWKS, however, they are implemented differently. In Gyro Mode, the raw gyroscope data is used to directly determine the rotation of the device. In Accel Mode, however, only the accelerations along the x and y axes are taken into account. This means to issue a recognizable movement command, the device must be *accelerated* in the desired direction. This can be accomplished by thrusting the phone in the desired direction.

Because stopping the phone after an acceleration will produce a negative acceleration, the app waits for a negative acceleration and ignores it.

Operating the device in Accel Mode is not very intuitive, so the decision was made to build Gyro Mode and make it the default. In gyro mode, the device detects which way it has been tilted and determines one of the four appropriate movement commands (or no movement). Rotating the device back to its initial orientation (with the screen facing up, perpendicular to gravity) does *not* produce a reverse rotation. This simplified the implementation and the user interaction.

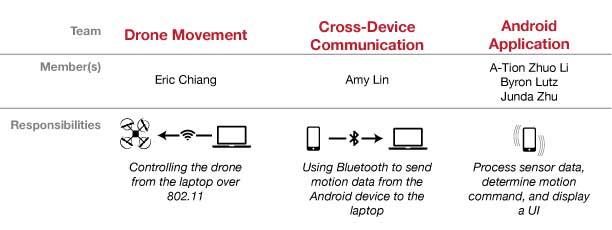
Gyro Mode does have a drawback, though. In Accel Mode, it is intuitive that, to move the drone up or down, the user can accelerate the phone up or down. With the gyroscope, however, there is no intuitive way to move the drone up or down just by tilting the device. Thus, up and down buttons were added to the touchscreen so the user can manually issue these directions while in Gyro Mode.

There were two major challenges in implementing this sensor processing: (1) ignoring noise and (2) implementing the flip command. A noise threshold was set for both Gyro Mode and Accel Mode: movements within this threshold would be ignored. Finding this value so that the device was sensitive *enough* but didn’t require excessive rotation/acceleration was a challenge and required testing.

To implement flipping the drone by flipping the device, a timer had to be set that kept track of the different commands that were issued to make sure incorrect commands were not sent to the laptop during a flip.

# 4. Contribution

The group was divided into three teams: Drone Movement, Cross Device Communication, and Android Application. These roles are described in the figure below.



## 4.1 Eric Chiang

Eric functioned as the project manager: he set up group meetings and assigned people to different roles. As the owner of the AR Drone, he presented the drone as an option to use for the project idea and worked closely with it on the drone movement team.

He was also in charge of designing the overall system architecture. This includes figuring out how the phone communicates with the laptop while the laptop’s only WiFi interface is occupied by the communication with the drone. Eventually Bluetooth came as a natural solution.

From a technical perspective, he implemented the Node.js application that reads commands and sends directions to the drone. Direction sending was built with a framework called NodeCopter, an open source project that leverages the event driven programming nature of Node.js. Since commands to the drone are sent via simple JSON packets, it is easy to use an event listener to efficiently respond to command requests from the smartphone then send command packets to the drone.

In addition, Eric presented the initial project idea presentation.

## 4.2 A-Tion Zhuo Li

Working on the Android app, Zhuo implemented the algorithm to process gyroscope data. He worked on the Android UI and implemented all gyroscope-related algorithms including noise removal, direction determination, and flip detection.

Zhuo also integrated the Android app, combining the sensor-related code and the Bluetooth communication code.

## 4.2 Amy Lin

Amy built the Bluetooth connection between the phone and laptop. Using Java, she used the Android Bluetooth interface to send data and used an open-source Bluetooth library to receive data. She then processed the data and output it in a format that the drone control application would accept.

Amy also presented the midpoint presentation.

## 4.2 Byron Lutz

Byron worked on retrieving the sensor data from the Android device in a format that can be parsed. He researched the differences between accelerometer and gyroscope data to determine which would be more appropriate to use. In addition, he worked on a UI overhaul that ultimately did not make it into the final project.

Byron wrote the final HAWKS report.

## 4.2 Junda Zhu

Junda implemented the accelerometer mode on the Android application. He acquired the accelerometer data from the phone, removed noise by setting a threshold value acquired by manual testing results, and designed two algorithms that convert accelerometer data into direction commands. The first algorithm calculates the change in consecutive accelerometer readings to determine acceleration in a particular direction, and the second algorithm directly interprets accelerometer readings for directions.

In addition, Junda implemented the initial project code and UI for other teammates to work on and designed the app icon.



# 5. Conclusion

HAWKS successfully implemented a motion-based, wireless controller to move a flying drone. Though this application works as-is, it can be used as a base for various improvements and applications.

## 5.1 Areas to improve

### 5.1.1 Non-cardinal directions

Right now, HAWKS only senses four directions: forward, backward, left, and right. If a user wants to move the drone in a direction other than these four cardinal directions, he/she must issue separate commands to compose their desired direction out of these four directions.

To improve the application, the device can be made more aware of the actual orientation of the device. For instance, of the top right corner of the device is tilted downward, the drone should move in a forward-right direction automatically instead of choosing a single direction based on which axis has a larger tilt.

### 5.1.2 Improved user experience

Because this project focused more on making the wireless communication mechanisms — Bluetooth and 802.11 — work, not much energy was devoted to the user experience. If this app were to be released to the public, a better visualization of the sensor data should be added (instead of just stating “LEFT”, “RIGHT”, “FORWARD”, or “BACK”).

In addition, the other buttons on the UI should be improved, calling attention to the land button. Finally, the touchscreen should be disabled when a flip is detected, making sure the user cannot press any of the buttons when flipping the device.

### 5.1.3 Drone stabilization

As mentioned in section 3.3, the drone has two cameras: a high-definition, front-facing camera and a standard-definition, ground-facing camera. The ground camera can be used to stabilize the drone so that when no commands are issued, the drone does not move.

This requires two major improvements: (1) image processing and (2) bidirectional communication. As of now, communication only exists one way: the device tells the drone what to do but gets no feedback.

Since battery and CPU usage are a priority on the mobile device, the laptop should retrieve a feed of the bottom-facing camera and assist in drone stabilization without complicating the Android app implementation.

## 5.2 Future applications

### 5.2.1 Multiplayer games

The type of gyroscope-based drone control demonstrated in HAWKS is fun to use and easy to learn, making it a very good candidate for an input method for a game.

Two smartphones can be connected to two separate drones. After creating an objective (for instance, racing, taking a photo of the other drone, or bumping into the other drone), the “loser’s” drone can be disabled for something like a timeout period.

This would be very interesting, requiring communication between two separate drone-control systems to negotiate a points system and a game start/end time among other things.

### 5.2.2 Collision prevention

Collision detection is a completely separate facet, modifying the drone’s internal logic instead of interacting with the laptop or smartphone. The drone itself would have to be modified to install sensors on the sides to automatically “bounce back” before even colliding with a wall.

# 6. References

### 6.1 Image attributions

Smartphone icon designed by George Agpoon from the thenounproject.com.

Bluetooth icon designed by Thomas Le Bas from the thenounproject.com.

Quadcopter icon designed by Nithin Nanthikkara from the thenounproject.com.

Laptop designed by Edward Boatman from the thenounproject.com.

### 6.2 Referenced code

A library by Felix Geisendörfer allowed us to easily communicate with the drone using Node.js. The library is available with the MIT license at https://github.com/felixge/node-ar-drone.

BlueCove is a library used to work with the Bluetooth interface on the laptop. It is available with the GPL license at http://bluecove.org/.