How to Develop an Intelligent Autonomous Drone

Introduction
Playing with small civilian drones is fun. Building them is even better! For those of you who are interested in developing your own "smart" drone, this article provides easy directions for making an intelligent, autonomous drone using an Android* smartphone, OpenCV*, C++, and Java*. And this is just the starting point. Once you've mastered these instructions, other programs are available to help you further refine your drone. Visit the Intel® Software Academic Program [1] for more information about the Intel® OpenCourseWare for self-study.

Materials and Methods

Autonomous and Smart?
For a drone to fly autonomously, all the necessary sensors, processing power, and communication chips must be built-in. While that may not seem like much, it actually eliminates a lot of currently existing commercial civilian drones available to the general public.

Did you see the drone commercial created by Lexus and KMEL Robotics [2] featuring swarms of small ultra-precise drones? Their movements were choreographed by positioning them in space within the room where they were operated, thanks to powerful sensors positioned all around the room. You probably also know that drones can be navigated using GPS, as drones have received a lot of attention recently. GPS is handy and very simple to access since it is digital, which is why it is used by planes to navigate during the cruise phase at high altitude. But the accuracy of ±2.5 m and an important latency won't allow a drone to deliver a pizza to your doorstep because 2.5 m too close and the drone would hit your house, or 2.5 m too far to the left or right, and it would crash as well. Yes, these drones have independent navigation, but they are not very "smart."

To be called “smart,” your drone must have enough embedded processing capabilities to, for example, capture a video and analyze in real time such targets as QR codes (easy), shapes, or movements (difficult). You can even measure volumes and rebuild a space in real time as was done with the MIT UAV [3]. However, all of these “upgrades” require a powerful processor plus sensors like accelerometers and GPS and possibly the ability to communicate over a 3G/4G network from time to time. You'd probably also like all that to be light weight, easy to program, and powered by a good battery. In the end, we are left with a very powerful mobile phone that can, in turn, power a drone's flight capabilities.

We recently developed a drone remote control device using an Android-based smartphone, in our case the ZTE V975 Geek, with an Intel® Atom™ processor Z2580. Using Android makes it easy to develop the software and to share code between a PC and the smartphone. Even native libraries like Intel® Integrated Performance Primitives (Intel® IPP) [4] or OpenCV [5] can be used on both intel android smartphones and PCs. So there is no need to reinvent the wheel—the smartphone has all the necessary components: camera, GPS, accelerometer, and 3G.
Motor Control

Once the choice of the embedded computer was made, we were ready to connect the motors. We selected the Pololu Maestro\textsuperscript{*} servo controller, which costs about €5, connects via USB, and even has Bluetooth\textsuperscript{*} with an optional Bluetooth-serial interface. This card is used to control standard servo motors. Whether you’re directing a quadcopter or another type of drone, just replace the radio control part with an Android-based smartphone and this Pololu card to make it an autonomous smart drone. The recipe is infinitely reusable. Simple, right?

With a few lines of code and using an Android USB standard package, we can control the servos and thus control the movements of the aircraft. With a few more lines of code, we can access the GPS, take pictures, and send them via 3G. At the software level, working with Android allows you to rapidly evolve your design.

Call \texttt{controlTransfer} from \texttt{UsbDeviceConnection}:

```java
import android.hardware.usb.UsbDeviceConnection;
// …
private UsbDeviceConnection connection;
// …
connection.controlTransfer(0x40, command, value, channel, null, 0, 5000);
```

The card allows you to move a servo determining the target position, velocity, and acceleration, which is all that is needed for smooth movement. The argument "command" can be one of these three values:

```java
public static final int USB_SET_POSITION = 0x85;
public static final int USB_SET_SPEED = 0x87;
public static final int USB_SET_ACCELERATION = 0x89;
```

You have to choose the corresponding value and target the right servo with "channel." The complete source code and configuration of USB access in the app manifest are included in the ZIP file [1].

The Special Case of Quadcopters

So far so good. The hardware is plug-n-play, the code is simple, and everything is done in Android. But the development of embedded systems retains some specific qualities, as we are about to see with quadcopters. There is a major difference in understanding multirotor drones and simpler models like remote control cars or planes. If you drive an RC electric car, all you need is an Electronic Speed Control
(ESC) servo for the acceleration and another servo for the direction. But multirotors need a permanent balancing engine to maintain the desired position. Luckily, quadcopters come with their own stabilization card. Instead of connecting directly to the Pololu card and the four ESC motors of the quadcopter and having to write complex stabilization software in C/C++ or Java on Android, it is much easier to connect the stabilization card to the Pololu card and let this stabilization card manage the four engines. Everything else you need to manage can be done using simple Java commands like +/- altitude, +/- speed, +/- inclination, and +/- direction. We chose to do everything on an Android-based smartphone because it has a powerful processor, but the balancing of engines is left to a small dedicated card costing only a few euros. As a developer, it is good for you to know about this card and why we decided to use it, but you will also appreciate the fact that it requires only an initial calibration and, once that is done, you can forget it.

**Conclusion of the First Stage**

In the end of the initial stage of developing the autonomous quadcopter, the hardware chain is:

- mobile phone <> micro USB-USB host adapter <> USB-mini USB cable <> Pololu Maestro card <> 4 JR cables <> stabilization card <> JR cables <> ESC <> UAV engines

By comparison, the hardware chain for simpler drones is:

- mobile phone <> micro USB-USB host adapter <> USB-mini USB cable <> Pololu Maestro card <> JR cables <> ESC <> UAV engines

As a bonus, you can also control other servos, such as a 2-channel direct 3D camera attached to the quadcopter. Or, if you want to, you can control the flaps, landing gear, etc. Pololu Maestro cards can control from 6 to 24 motors, which is a lot more than is actually needed for this project but does provide great flexibility.

So far, we have discussed the preferred software and hardware that make up the components for our embedded project. In the next section, you will see how to develop image analysis software to make this a smart drone.
Computer Vision
While we've established that our drone needs good embedded processing capacity to be intelligent and autonomous, what is still missing is the code that takes advantage of this potential. So, let's give "eyes" to our drone!

Why?
Some drones can use a GPS to navigate, but the accuracy and latency of GPS doesn't allow precise trajectories between buildings, people, or trees, for example. Other drones can fly with precision and dexterity in the laboratory, but they get their positioning cues from sophisticated cameras and sensors around the room. This same technique cannot be scaled for outdoors.

In real conditions, it is essential that the drone be able to "see," for example, to recognize and track markers or visually identify humans. So what remains is to be able to capture images from Android and analyze them with OpenCV.

How?
OpenCV is an open source library of programming functions for image analysis and is the basis of numerous computer vision and virtual reality projects. Originally developed by Intel, OpenCV is now available for many hardware and OS platforms. You can develop your code on a PC and deploy it on servers, smartphones, or Internet of Things (IoT) platforms.

To practice, we will first try to recognize a simple mark such as a circle and navigate using the smartphone so that we are in front of this circle at a fixed distance. Imagine that the drone is able to navigate by GPS to the global position of a runway (about 3 m). It must be able to recognize marks on the ground or pavement in order to position itself above the landing position precisely, at a fixed height. The drone will then land on the marked spot to within a few centimeters. To simplify the test, we will post the flight controls on the screen, and you will simulate the movement of the drone by moving the mobile by hand.

Navigating a circle with a smartphone

Native Java Project+
OpenCV is not part of the libraries available directly from Java on Android. It is a native library typically used from C++, so you need to use the Android NDK. The image capture and display part will be in Java, and the communication between Java and C++ will use JNI. We need to install the Android NDK, Android SDK, create the above-referenced "Circles" project, add a C/ C++ component, and change the project settings for using the OpenCV library, which are captured in the following Eclipse screen shots:
Setting the variable NDK_ROOT project C / C++

C/C++ project parser settings
At the end, our project will have:

Java main file « Src/MainActivity.java »

XML layout file « Res/layout/activity_main.xml » and the manifest

Two Makefiles « Jni/Android.mk » and « Jni/Application.mk »

cpp Code « Jni/ComputerVision_jni.cpp » and header « Jni/ComputerVision_jni.h »

**Hardware Architecture**

Unlike Java, C++ must be compiled for a specific processor. This adjustment is made by simply editing the variable `APP_ABI` in `Application.mk` file. For our Intel Atom processor-based smartphone, the correct value is "x86"; the NDK will do the rest.

**Deployment**

OpenCV is a library used by countless Android apps, and each app may use a different version of the library. As an app developer you could bundle the required version of OpenCV with your app, but there's a better method: use a dependency manager called "OpenCV Manager." It's an Android app that detects when you need OpenCV, what version, and/or help with installation, and then loads OpenCV. Your app will have to establish a connection with OpenCV Manager, and the rest will happen "automagically."
Interaction C++/Java and Algorithm
We want to detect circles in OpenCV, determine the center and radius, and then display orders to the operator of the smartphone to achieve a perfectly centered, good-sized circle. The following Java code retrieves the image from the camera with the Java API for Android and makes a call to C++ by JNI attaching a pointer to the image in memory. The C++ code then performs image processing to detect circles. Java is called back to display the detected circles and comments on the picture.

Remote Operation
During testing, I moved the phone in front of a printed sheet of paper. To simulate the final position of the drone, I picked a distance to see how wide it appeared according to the optics of my phone's camera. Example: I drew a 10 cm circle on the sheet of paper, I placed my phone 20 cm from the sheet of paper, and the circle appears to be 300 px wide. That's how wide it has to appear to be the correct distance. If the circle is too big, I will move backward; if too small, I will move forward. In the end, the drone will be at a precise distance in the circle.

Our first test case is very simple: 1 circle = 1 distance. But you can also go further and use several concentric circles: the large circle can be easily detected from a long distance, the middle circles will be used when the drone is too close to see the largest circle, and the drone will finally land on the smallest inner circle. If circles are not precise enough, you can also use more complex shapes like arrows.

Center the circle on the image of the smartphone to simulate the center positioning of the drone just above the landing mark. This type of control is actually simple and, when combined with other information such as color, GPS position, and perhaps depth, it can be very easy to implement.

Java Code
```java
... // capture images from the camera import org.opencv.Android.CameraBridgeViewBase;
// load OpenCV native dependency import org.opencv.Android.OpenCVLoader;
...
public void onResume()
{
    super.onResume();
    // OpenCV loading with a callback // non typical code specific to OpenCV OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_6, this, mLoaderCallback);
}
...
// once the OpenCV manager link established,
// we can load the dynamic library System.loadLibrary("jni_part");
...
```

C++ Code
```c++
... // typical for JNI : Java class method name // pointer to RGB image as argument JNIEXPORT int JNICALL Java_com_example_circles_MainActivity_process
```
(JNIEnv *jenv, jobject obj, jlong addrGray, jlong addrRgba)

... // Get the bitmap from pointer
Mat& mRgb = *(Mat*)addrRgba;
// blur, required before detecting circles
medianBlur(mGr,mGr,5);
// OpenCV detection — Hough transformation
HoughCircles(mGr, //grayscale input image
              *circles, //output vector
              CV_HOUGH_GRADIENT, //detection method to use
              4,  //inverse ratio of the accumulator resolution to the image
              mGr.rows/8,  //min distance between centers of detected circles
              220,  //higher threshold of the two passed intern canny edge detector
              200, //accumulator threshold 100
              20, //min radius
              mGr.cols/8 //max radius
          );

Performance and Next Steps
Install the OpenCV Manager from Google Play* [6] and your app APK file from Eclipse* [7]. Start your app and let it guide you through the set up; it will detect circles in the field of vision and guide you to center the circle with a predefined diameter.

Detecting circles is very basic. It’s a good idea to start with a plain background and high contrast for easy visibility. Simple computer vision functions work with simple shapes, just like toddlers play with simple shapes: circles, squares, stars—all geometric primitives defined as vectors. But advanced computer vision functions can handle any photographic input, not only simple geometric primitives. Such libraries are used in augmented reality apps; they can detect the presence of any photo in another photo, even if distorted. Example: given the photo of a food package, an app will detect the position and 3D orientation of the package in a larger photo. Most computer vision apps work by moving an object in front of a fixed mobile phone. The object is detected and can be augmented by 3D objects, as depicted in this video: http://www.dailymotion.com/video/xco8xm_la-realite-augmente-par-total-imme tech [8].

We don’t need our drone to play with food packaging, of course. But the same library can be used differently, still with a fixed photographic pattern to detect, but using a moving mobile phone on a drone. Example: Once I give the drone the photo of the landing site, the drone will be able to detect it, understand very precisely where it is in space compared to the pattern, and prepare a visual approach. Same library, different use case.

To control the drone, simply replace the directions displayed by sending the commands “power,” “roll,” “pitch,” and “yaw” to the drone that carries the camera. Fortunately, orders to pilot the drone and the printed commands match exactly so the migration from your indoor manual tests to the outdoor flights will be easy.

On our test smartphone, we capture and process an image every 8 hundredths of a second, 12.5 frames per second. This proves that it is possible to simply add computer “vision” to a drone with a limited
development time/processor/energy budget, which is reasonable and realistic for a small drone. And everything is computed independently without deporting a single calculation.

It is possible to go a lot further. OpenCV is an open source library and widely portable. Plus, Intel IPP [5] replaces some low level OpenCV calls and accelerates your code by injecting routines highly optimized for Intel processors. You can keep your code portable and your dependency on an Open Source library while enjoying the optional benefits of Intel IPP performance.

Finally, using an Intel processor-based Android smartphone to control a drone is a solution that combines performance, endurance, and ease of development. This first project gives you the option to migrate to another OS or a more powerful hardware platform if your smartphone is no longer sufficient or you wish to move to larger drones. Again, check out the other software programs [1], which show other available software development instructions in the realm of drones and robotics.

About the Authors
Authors: Paul Guermonprez, Nicolas Vailliet, Cedric Andreolli
Link: http://intel-software-academic-program.com/pages/courses#drones
Contact: paul.guermonprez@intel.com
Team: Intel Software Academic Program for EMEA-Russia, Paris, France

References and Resources
Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRES OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents that have an order number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or going to: http://www.intel.com/design/literature.htm

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel, the Intel logo, and Atom are trademarks of Intel Corporation in the U.S. and/or other countries.

Copyright ©2014 Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.