Mobile System Programming

Spring 2014

Traffic Viewer Final report

Group 6

Christian Cardin - Ruben Guerrero - Wille Keltikangas - Elmeri Poikolainen - Rafael Andrade Lott

1. Evaluation of the results

Our system was an Android application that utilizes the Parrot AR Drone 2.0 to detect objects and follows them as they move. Our goal was to produce this functional Android application that will control the drone to track an object.

The scope of this project was to produce an Android application that tracks an object, from the video retrieved by the drone, and detects its movement using OpenCV library. The application sends commands to the drone according to the movement of the object being tracked. This project does not include a functionality to manually pilot the drone or to fly the drone in any other way.

What we achieved is an Android program that tracks an object on the screen based on the color of the object. The program uses OpenCV library for android and uses basic library functions to locate the middle point of the seen colors. The middle point is drawn as a blue line on the video feed. Since the integration to controlling of the drone was partially implemented, hypothetical use of the image tracking is described. Drone captures the pictures from the bottom view camera and it is passed on to the image tracking. The idea is to track the middle point of the object and based on that the program would calculate a direction for the drone and send it back to drone control.

The learning goal for this project was to know how to develop Android applications and external libraries such as Parrot AR Drone SDK and OpenCV.

This project achieved some of the learning goals. Since the group was divided between 2 groups - machine vision and controlling of the drone - the learning varied as such. Both teams got experience from Android development and the other team from OpenCV and drone control team from Parrot AR Drone SDK.

Image recognition with OpenCV was one side result produced by the elaboration of this project. Since our project was to create at app that uses image recognition and nobody in the team had experience in this field, self-study of the subject was required. Another result was the ability to develop using Android NDK (Native Development Kit) and to develop for Parrot AR Drone 2.0. However, not all members of the can say they learn image recognition with OpenCV since we split in two teams, as previously said.

In a nutshell, these results are in the learning part to develop this project, but clearly it was a disadvantage to not knowing anything of them before this project and a lot of time was expended in learning the technologies. Of course, Android NDK and Parrot AR Drone API was something that is expected to be unknown for us. We think that in order to make a properly functional application that utilizes image tracking, at least one member of the team should know the basics of it or had taken a course about it.

The result we obtain wasn't the one we wanted . As previously said, we couldn't fully implement the tracking system in the drone due to the complexity of the system and all the time needed for knowing the technologies.

We have an image tracking application that works really well, but due to time issues we couldn't merge the drone controlling part and this app.

Another problem we had was the complexity of Parrot AR Drone 2.0. When you get the API, you get examples of applications that utilize the drone. There's an Android example (used as base for our project) that is highly complex for our Android skill level. At first, we didn't wanted to use this example as base because of that and we search for external APIs for the drone in Java. We actually found several, JavaDrone being the most notable, that works great for piloting the drone and was very simple to use and understand. The problem was that this API was for Parrot AR Drone 1.0 and the video used in the first version of the drone uses another format, so JavaDrone couldn't retrieve the video of our drone and this is crucial for our project. We reject that API and return to the example wasting precious time.

Our next difficulty was to understand and run the app. First it required several steps in order to compile it. For some reason unknown for us, it can only be compile in a Linux 32 bit machine. After successfully compiling, executing and testing the example the next step was to understand the code. Here we face a wall due to the lack of comments or documentation in the code. It is one thing to have a complex and highly couple example and having a complex and highly couple example without comments. Variable names weren't as descriptive as we wanted and the documentation pdf was mostly about how does the drone works and how to control the drone in a low level, not the one desired. Desperates, we end up doing thing at trial and error by deleting code and seeing what happen.

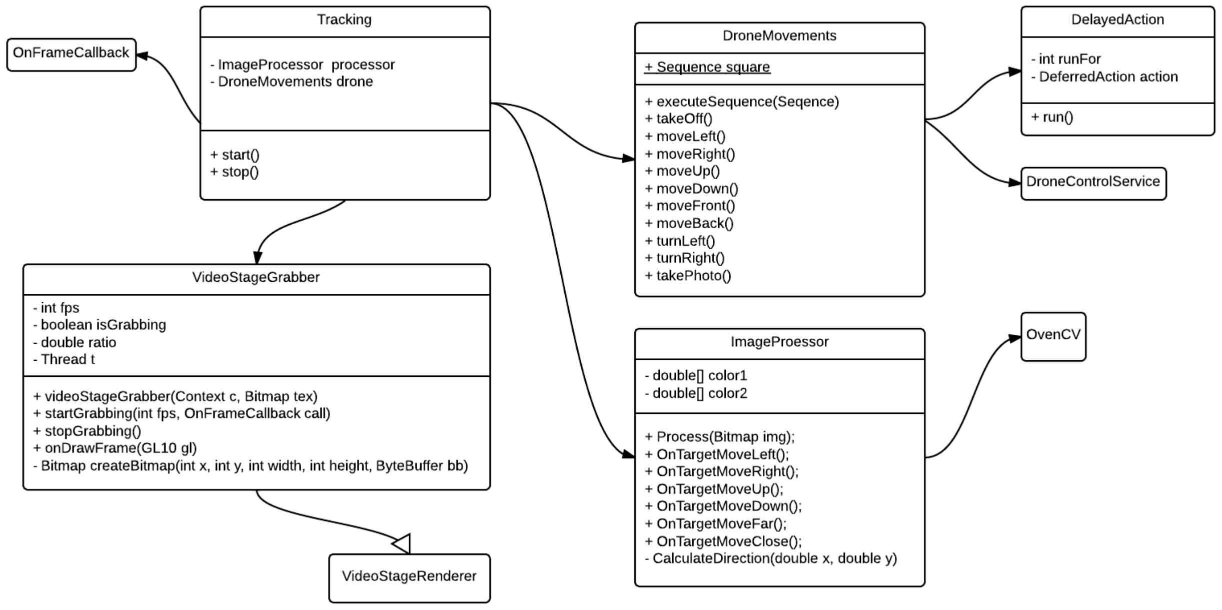
The last problem of this project was the availability of us to do testings. We mostly coded separates and had only meeting to talk about the project and how is it going. With this model, only one person could test the application with the drone at any time. If someone wanted to test something, he must first contact the person who had the drone at the time and go for it.

Application Architecture

The following picture shows the core of the Tracking application. The strategy adopted for the assignment was to adapt the components already available into the AR.Drone SDK for Android.

Inside the SDK package there is an exhaustive example application which lets the user to control the Parrot and record media like photos and videos. Unfortunately there is not a proper documentation available online, and the source code is not commented adequately.

Firstly, we inspected the Android code to recognize relevant components suitable to our needs, and then understand how we could adapt them for adding the image recognition feature directly inside the provided application.



- **DroneMovements:** It's a wrapper around the *DroneControlService* component to send movement commands to the drone. It extends the basic functionalities by adding the support for the automated execution of predefined movement sequences. Starting from a list of actions and the duration for each of them, a user can build his own custom movements that the Drone will execute.
- **DelayedAction:** helper class for delay the execution of a function. Useful to schedule chains of actions to execute in a predefined time with a certain duration.
- **ImageProcessor:** using the OpenCV library for Android, this component is able to analyze a bitmap and detect a colored target inside of it. The color is specified in the class constructor. After the elaboration the position of the object in the screen is detected and, if it exceeded the screen boundaries, the adjustment functions will be called in order

to align the camera with the target again. The elaboration is computed in a separate thread for preventing bottlenecks and slowdowns in the main UI and video rendering

- **VideoStageGrabber:** Derives from the *VideoStageRenderer*, a component used to render the application scene on the screen. Most part of the video elaboration is done by the native code using the Android OpenGL ES, which is difficult to inspect and change since there is no documentation. This component intercepts the object containing the texture to render on the screen before it is actually rendered, making it available to the rest part of the application, in particular to the ImageProcessor class. However, since the OpenGL object is binded in the context where it was created, the VideoStageGrabber must convert it into another object (for example a Bitmap) in order to make it available to the ImageProcessor, which can elaborate it in a separate thread.
- **Tracking:** Acts like a Facade, hiding the complexity of the other components. The start() method creates the bindings between *DroneMovement, ImageProcessor* and *VideoStageGrabber* by customizing the behaviour of the abstract classes and assigning callbacks.

2. Metrics

We used multiples programs to be able to develop effectively. For the group communication, files sharing and meeting schedule, we used a program called Flowdock.
For creating the documentation of Project Plan and Final Report, the program used was the Google Drive, where all group members, independently from the OS used, could contribute to the documentation. For development, some used the Eclipse IDE, others tried the new Android Studio.

The Parrot AR Drone has an API, on which we used the version 2.0 for this project, we also used the OpenCV library for the video capturing and recognition. We have found an Android Wrap controller for the Parrot AR Drone 2.0, on which we used as initial code for development. Unfortunately, the documentation for that code was not very helpful at all.

At the beginning, our quality metrics were low, since we didn’t have enough knowledge in the Drone API or on how the system works. After getting familiar with the system and the program, we start to increase our metrics, trying to customize the code for our goals. Unfortunately, Android Wrap controller did not have a good documentation or comments, and that made the understanding of the code more complicated as previously said.

3. Work practices and tools

Eclipse

Eclipse is the currently recommended IDE for android development. It worked fine for the basic stuff, but setting up the NDK was pretty difficult. Major problems was that since the code need to be compiled in a Linux 32-bit machine, most of us used a VM for this and the performance of the VM was really bad, hence eclipse crashed sometimes.

Flowdock

Flowdock is a new team collaboration application for desktop, mobile and web. There was a lot of discussion about which medium we would use to stay in touch with each others. In the end, we decided to try with Flowdock. We used it to send messages, share information and schedule meetings. All in all, it worked ok. We were pretty active at first, but later on participation was low. This may be because not everyone had it as a mobile app and they didn’t check the web app so often. In retrospect a program that has constant push notifications on the phone would have been more useful. Flowdock sends push notifications only when you have a 1-on-1 with someone, not in the main thread where all team members participate so it required that each one remember to check it every now and then.

Google Drive

We used Google Drive to create our project plan, final report and any other deliverables. We also used Google Drive spreadsheet to keep track of our working hours.

OpenCV android library

OpenCV is a library of computer vision related convenience functions. We used the Java library for Android. It was really good, but at some point we realised that it would not be fast enough for our use cases and we would have to run native code (C/C++) on Android.

Android SDK

We used the Android SDK to create the android applications for our project and to install any android related tooling to our computers.

Android Studio

While most of us used Eclipse as the IDE, some tried the new Android Studio as well. Android Studio was still very much WIP and trying to get things to work with it was really hard. Android studio uses gradle as it’s build tool and it also includes gradle specific files in the project directory. For this reason, trying to share Android Studio projects with Eclipse users was complicated, because it was messy and they couldn't easily build them.

Android Emulator

The Android SDK included an emulator for testing android applications without using a physical device. The emulator did what it promised to do, but the problem was, it worked really slowly. Even turning on the hardware acceleration did little to fix that. This forced us to almost solely use our test phone for application development, but because we only had one, we couldn't work that fast. Using the emulator with the drone was impossible, since it took forever to load.

Android NDK

There were many times, when we found that the Android Java API was not enough for us. Trying to work with the NDK was challenging, because the support for it was very lacking. Especially when working with Android Studio and on an operating system that wasn't a Linux. The native C/C++ development pretty much required us to either run Linux or some kind of emulated Unix environment (cygwin).

4. Preliminary business model

We were given very loose requirements for the application. As such we lack the necessary information to properly speculate on the business models.

Business model (STOF)

Service domain

e.g. Value proposition, Target group

One model, the *Value Proposition Builder* for creating a value proposition states six stages to the analysis:[3]

1. Market: for which market is the value proposition being created?
2. Value experience or customer experience: what does the market value most? The effectiveness of the value proposition depends on gathering real customer, prospect or employee feedback.

3. Offering: which products or services are being offered?
4. Benefits: what are the benefits the market will derive from the product or service?
5. Alternatives and differentiation: what alternative options does the market have to the product or service?
6. Proof: what evidence is there to substantiate your value proposition?

Technology domain

e. g. Service delivery system

Our application is an Android application which uses Parrot AR Drone 2.0 as hardware. If published, it will be deliver in Google Play.

Finance domain

e.g. Revenue model

Revenue model is highly dependant on the value proposition which was not clear. Possible revenue models would providing a drone tracking service or selling the tracking program as a separate application on Android market(s).

Organization domain

e.g. Division roles, network strategy

Organization domain is not applicable in this context. There is manufacturer of the Parrot AR Drones that supplies the physical device that performs the flying and capturing the video feed. Another organization is ours which develops the program and sells/provides it for the customers.

Hypothetical deployment plan

Hypothetical deployment plan is difficult to write since our business model is not clear.