FIUBAAR

Realidad Aumentada en ANDROID

Documentación del proyecto de Trabajo Profesional de Ingeniería en Informática FIUBAAR.

2017

Esteban Guillardoy (P.81597) sagarboy@gmail.com & Alejandro E. Aguilar (P.73935) alezek@gmail.com

Facultad de Ingeniería de la Universidad de Buenos Aires

17/02/2017

Tabla de contenido

[Arquitectura de la Aplicación 5](#_Toc475121435)

[Introducción 5](#_Toc475121436)

[Arquitectura de Android 5](#_Toc475121437)

[Model-View-Presenter 6](#_Toc475121438)

[Performance 6](#_Toc475121439)

[Diagrama de paquetes 8](#_Toc475121440)

[Diagramas de Clases 11](#_Toc475121441)

[Clase MainApplication 11](#_Toc475121442)

[Clase mainactivity 12](#_Toc475121443)

[Clase maincontroller 13](#_Toc475121444)

[Clase objectar 14](#_Toc475121445)

[Clase marker 15](#_Toc475121446)

[Clase markerdetector 16](#_Toc475121447)

[Clase nativehandtrackingdetector 17](#_Toc475121448)

[Clase handdebugview 18](#_Toc475121449)

[Clase vuforiafragment 19](#_Toc475121450)

[Clase vuforiarenderer 20](#_Toc475121451)

[Clases del paquete engine3d: Base3dFragment y modelfragment 21](#_Toc475121452)

[Clases del paquete rest: RestClient y download3dmodelasynctask 22](#_Toc475121453)

[Diagrama de Despliegue 23](#_Toc475121454)

[Diseño de la Interfaz de Usuario 24](#_Toc475121455)

[Modelo de datos 25](#_Toc475121456)

[Conclusión 26](#_Toc475121457)

[Setup Entorno de Desarrollo 27](#_Toc475121458)

[Objetivo 27](#_Toc475121459)

[Setup 27](#_Toc475121460)

[Requerimientos 27](#_Toc475121461)

[Configuración 27](#_Toc475121462)

[Alternativa 28](#_Toc475121463)

[Definición de variables de entorno 28](#_Toc475121464)

[Dispositivos Android 28](#_Toc475121465)

[Herramientas adicionales 29](#_Toc475121466)

[Notas Adicionales 29](#_Toc475121467)

[Buenas Prácticas 30](#_Toc475121468)

[Comentarios 30](#_Toc475121469)

[Desarrollo en Android 30](#_Toc475121470)

[Repositorio 30](#_Toc475121471)

[Uso de git en el proyecto 30](#_Toc475121472)

[Referencias 31](#_Toc475121473)

[Investigación de Markers 32](#_Toc475121474)

[Introducción 32](#_Toc475121475)

[Estado del arte 32](#_Toc475121476)

[Requerimientos FiubaAR 32](#_Toc475121477)

[Elección del marker núcleo 33](#_Toc475121478)

[Tracking del marker 33](#_Toc475121479)

[Definición de marker 33](#_Toc475121480)

[Documentación adicional útil 34](#_Toc475121481)

[Investigación Detección de Markers 36](#_Toc475121482)

[Introducción 36](#_Toc475121483)

[Implementaciones analizadas 36](#_Toc475121484)

[Documentación útil 37](#_Toc475121485)

[Testing de implementaciones 37](#_Toc475121486)

[Aruco 37](#_Toc475121487)

[ARma 39](#_Toc475121488)

[Implementation of a Java Framework for Marker Based Detection in Augmented Reality 39](#_Toc475121489)

[Marker Detection for Augmented Reality Applications 39](#_Toc475121490)

[OpenCV Square Tracking Android JNI 40](#_Toc475121491)

[Vuforia 40](#_Toc475121492)

[Conclusiones 40](#_Toc475121493)

[Investigación Detección de código QR 41](#_Toc475121494)

[Introducción 41](#_Toc475121495)

[Implementaciones encontradas 41](#_Toc475121496)

[Documentación útil 41](#_Toc475121497)

[Testing de implementaciones 42](#_Toc475121498)

[Conclusiones 42](#_Toc475121499)

[Investigación Engine 3D Android 42](#_Toc475121500)

[Introducción 42](#_Toc475121501)

[Implementaciones encontradas 42](#_Toc475121502)

[Documentación útil 43](#_Toc475121503)

[Conclusiones 43](#_Toc475121504)

[Investigación Clientes REST Android 43](#_Toc475121505)

[Introducción 43](#_Toc475121506)

[Implementaciones encontradas 43](#_Toc475121507)

[Documentación útil 44](#_Toc475121508)

[Testing de implementaciones 44](#_Toc475121509)

[Conclusiones 44](#_Toc475121510)

[Investigación Hand & Finger Tracking 45](#_Toc475121511)

[Introducción 45](#_Toc475121512)

[Implementaciones encontradas 45](#_Toc475121513)

[Testing de implementaciones 46](#_Toc475121514)

[Conclusiones 46](#_Toc475121515)

[Investigación sobre detección de gestos 47](#_Toc475121516)

[Ideas de Implementación 49](#_Toc475121517)

[Links 50](#_Toc475121518)

[Burndown Charts por Sprint 52](#_Toc475121519)

[Sprint 1 52](#_Toc475121520)

[Sprint 2 52](#_Toc475121521)

[Sprint 3 53](#_Toc475121522)

[Sprint 4 53](#_Toc475121523)

[Sprint 5 54](#_Toc475121524)

[Sprint 6 54](#_Toc475121525)

[Calendario 55](#_Toc475121526)

[Replanificación N°1 - Al inicio del Sprint N°3, 20/Octubre/2014. 55](#_Toc475121527)

[Replanificación N°2 - Al inicio del Sprint N°4, 26/Enero/2015. 55](#_Toc475121528)

[Replanificación N°3 - Al inicio del Sprint N°5, 26/Enero/2015. 55](#_Toc475121529)

[**Sprint N°1** (137 HH) Inicio: 04/08/2014 Finalización: 07/09/2014 55](#_Toc475121530)

[**Sprint N°2** (144HH) Inicio: 08/09/2014 Finalización: 19/10/2014 55](#_Toc475121531)

[**Sprint N°3** (156HH) Inicio: 15/12/2014 Finalización: 25/01/2015 56](#_Toc475121532)

[**Sprint N°4** (152HH) Inicio: 03/08/2015 Finalización: 13/09/2015 56](#_Toc475121533)

[**Sprint N°5** (163HH) Inicio: 26/09/2016 Finalización: 06/11/2016 56](#_Toc475121534)

[**Sprint N°6** (151HH) Inicio: 07/11/2016 Finalización: 18/12/2016 57](#_Toc475121535)

[**Sprint N°7** (154HH) Inicio: 02/01/2017 Finalización: 12/02/2017 57](#_Toc475121536)

# Arquitectura de la Aplicación

## Introducción

La documentación oficial de Android explica los conceptos principales de una aplicación

<http://developer.android.com/guide/components/fundamentals.html>

Desde un inicio, se desea integrar en una aplicación básica todos los conceptos que probablemente serán utilizados en futuras versiones de la aplicación. Es por ello que en la versión de la aplicación correspondiente al Sprint 1, vamos a integrar los siguientes componentes:

OpenCV

Engine 3D (al menos uno a modo de prueba inicial)

Implementación de código nativo vía JNI (Como referencia visitar <http://developer.android.com/training/articles/perf-jni.html>)

Servicios locales en background

Multithreading (en los servicios vía ThreadPoolExecutor)

Varias vistas en forma de Fragments

Dada que esta es la primera vez que los integrantes participan del desarrollo de una aplicación para la plataforma Android, se decide investigar a cerca de los modelos de arquitectura más comunes y los patrones que se utilizan.

Debe tenerse en cuenta que el presente trabajo implica el desarrollo de dos aplicaciones: FIUBAAR Client App y FIUBAAR Server App. La complejidad se encuentra en la aplicación cliente y allí es donde se aplican los mayores esfuerzos, por lo que la documentación de la arquitectura está centrada también esta aplicación, mostrando detalles de la Server App cuando se lo considera necesario o conveniente.

## Arquitectura de Android

También algo importante a entender es la arquitectura general de la plataforma Android:

<http://www.tutorialspoint.com/android/android_architecture.htm>

Android es básicamente un sistema operativo Linux y posee un componente principal relacionado a las aplicaciones llamado Dalvik VM. Cada aplicación es ejecutada en un proceso y este proceso posee una instancia de una DVM. Es sin embargo importante mencionar que DVM evolucionó y, a partir de la versión 4.4 Kit Kat de Android, su reemplazo lleva el nombre de ART: Android Run Time. ART incorpora compilación AOT (Ahead-of-Time) y un recolector de basura (GC, Garbage Collector) mejorado. AOT implica que lo que se almacena y ejecuta en nuestros dispositivos ya no es Bytecode (o mejor dicho, archivos .dex), sino código nativo, lo cual deriva en una mejora en la velocidad, que particularmente se nota en el menor tiempo de inicio de las aplicaciones. Dalvik funcionaba con compilación JIT (Just-in-Time), de la cual se esperaba que compilara -y optimizara- el código al tiempo que lo leía (on-the-fly), con la contrapartida de que esto sucedía mientras ejecutábamos la aplicación, posiblemente afectando en forma negativa el desempeño. AOT hace lo que JIT, pero una única vez, al momento de instalar la aplicación. Como mencionamos, está presente a partir de la versión 4.4 de Android, aunque no por defecto (hay que habilitarlo desde *Settings* -- >  *Developer Options* -- > *Select Runtime*-- > *Use ART*).

## Model-View-Presenter

En general, todo gira alrededor del patrón Model-View-Controller (MVC). Si bien la plataforma Android no implementa por defecto el patrón MVC

(<http://stackoverflow.com/questions/2925054/mvc-pattern-in-android>), es posible implementarlo de forma muy coherente:

<http://www.therealjoshua.com/2012/07/android-architecture-part-10-the-activity-revisited/>

<http://mindtherobot.com/blog/675/android-architecture-message-based-mvc/>

<http://www.thinkmind.org/download.php?articleid=patterns_2013_1_20_70039>

Si pensamos un poco más estrictamente, el patrón que en realidad se utilizará es Model-View-Presenter (<http://stackoverflow.com/questions/19996963/difference-between-asp-net-mvc-and-mvp-are-they-both-same>)

MVP es un derivado (o en algunos casos considerado un subgrupo de MVC) que aplica mejor a la lógica que queremos utilizar, donde el modelo no notificará directamente a las vistas sino que los controladores centralizan todo y son estos los que actualizan/notifican a las vistas.

Determinamos entonces que nuestra aplicación hará uso de MVP.

## Performance

Como dijimos, cada aplicación es ejecutada en un proceso y este proceso posee una instancia de una DVM. <http://developer.android.com/guide/components/processes-and-threads.html>

Debido a esto, debemos considerar desde un principio que si bien se construirá inicialmente una aplicación básica que emplee OpenCV para utilizar características simples como capturar video de una cámara, capturar imágenes, aplicar filtros de colores, etc., esta deberá evolucionar hacia una aplicación que realice procesamiento intensivo cuando se logre desarrollar toda la funcionalidad comprometida, por lo que sin dudas requerirá un muy buen manejo de la performance.

Por ello debemos contemplar la opción de implementar servicios que se ejecutarán en background, con múltiples threads para realizar diferentes tareas fuera del thread principal de UI. Es posible que los servicios ejecuten en un mismo proceso de la aplicación o en otros procesos externos. Dependiendo de la forma en la que los servicios funcionen, se deberán utilizar diferentes formas de comunicación, tal como lo muestra la documentación oficial:

<http://developer.android.com/guide/components/services.html>

<http://developer.android.com/reference/android/app/Service.html>

Encontramos mucha información sobre cómo ejecutar servicios en otros procesos y cómo realizar la comunicación entre servicios y actividades.

<http://stackoverflow.com/questions/4083756/android-service-process-vs-not>

<http://stackoverflow.com/questions/2463175/how-to-have-android-service-communicate-with-activity>

<http://stackoverflow.com/questions/4300291/example-communication-between-activity-and-service-using-messaging>

<http://android-developers.blogspot.com.ar/2010/04/multitasking-android-way.html>

<http://mindtherobot.com/blog/37/android-architecture-tutorial-developing-an-app-with-a-background-service-using-ipc/>

<http://www.techotopia.com/index.php/Android_Remote_Bound_Services_%E2%80%93_A_Worked_Example>

<http://stackoverflow.com/questions/15524280/service-vs-intent-service>

Algo que es frecuentemente destacado en múltiples sitios (foros, blogs, guías, etc.), es que utilizar servicios en diferentes procesos y comunicar entre ellos vía IPC es costoso, con lo cual para poder tener mejor performance la mejor alternativa en nuestro caso es utilizar servicios locales.

Se realizó una investigación inicial sobre la mejora de performance de OpenCV de manera de maximizar el valor de FPS (Frames Por Segundo) que se pueden obtener en Android.

Una de las alternativas planteadas para mejorar la performance y FPS requiere hacer uso directo de la API de cámara de Android en lugar de hacer uso de OpenCV. Esto permitiría obtener los previews de cada frame evitando parte del overhead agregado por el uso de OpenCV y varios callbacks y listeners asociados.

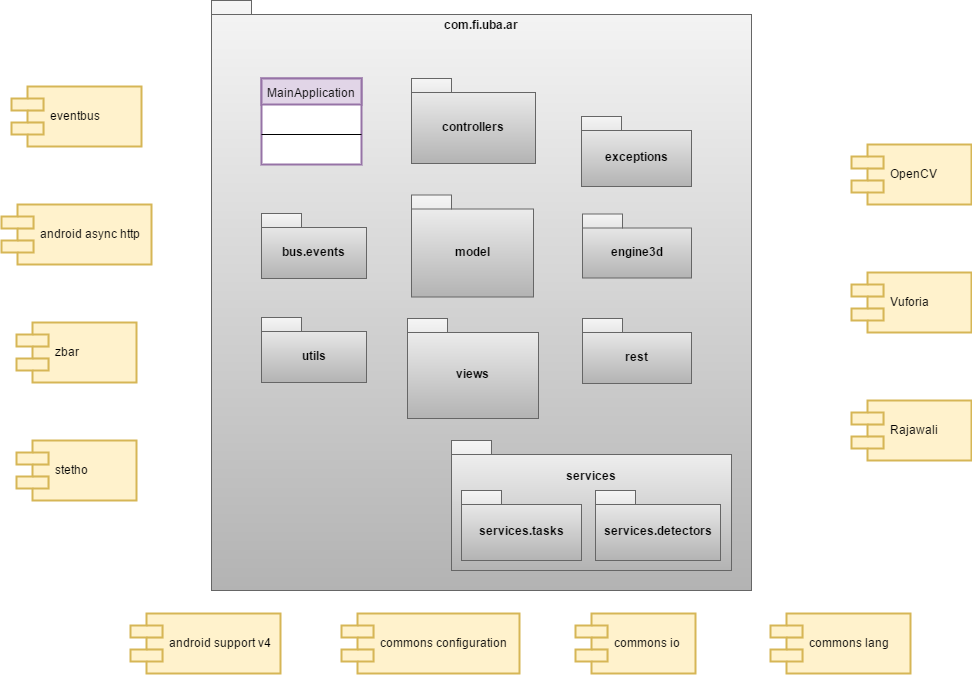
En esta aplicación básica entonces a modo de prueba se implementará esta propuesta junto con una versión que utilice la cámara mediante clases de OpenCV para así poder realizar tests de performance.

# 

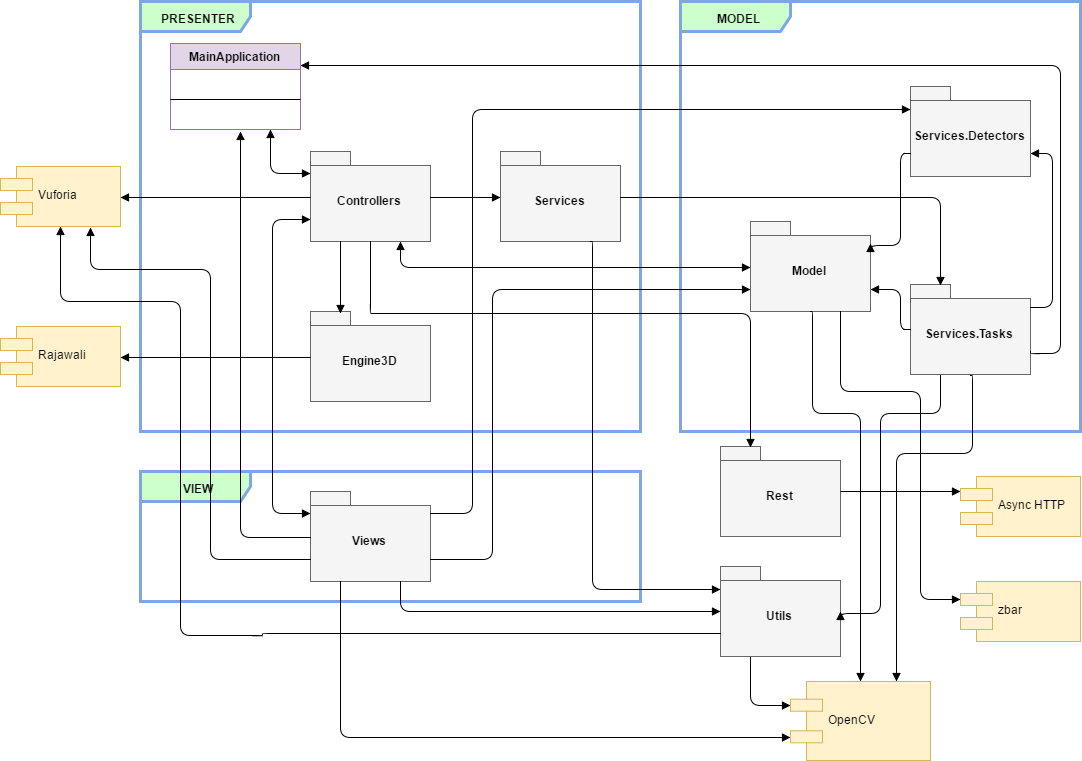
# 

## Diagrama de paquetes

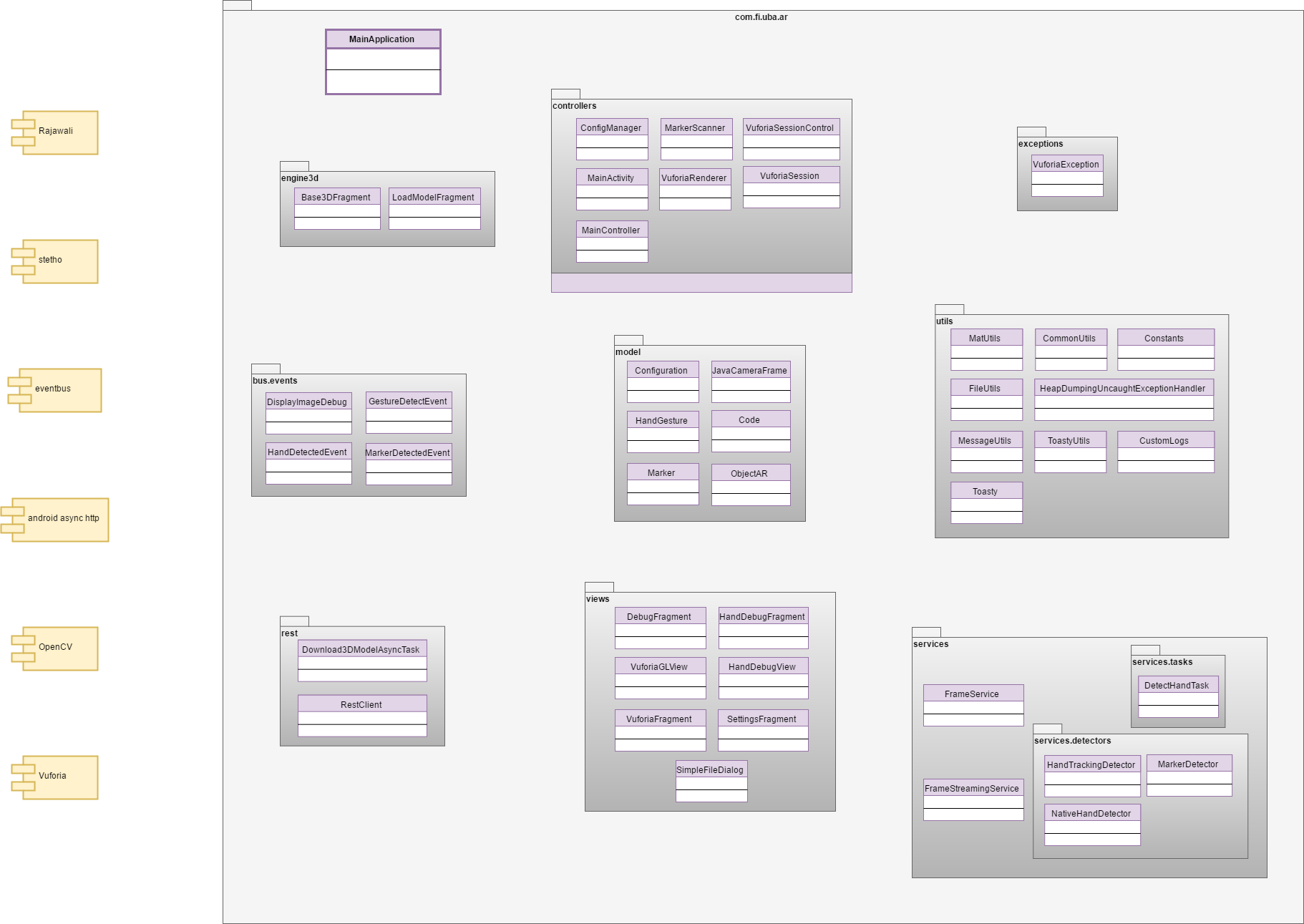
El código de la aplicación cliente se distribuye en los siguientes paquetes y librerías.



A continuación, el diagrama de paquetes muestra las asociaciones entre paquetes y librerías, destacándose la implementación del patrón MVP. También se identifican las librerías de mayor relevancia.



El siguiente diagrama explicita las clases contenidas dentro de cada paquete.



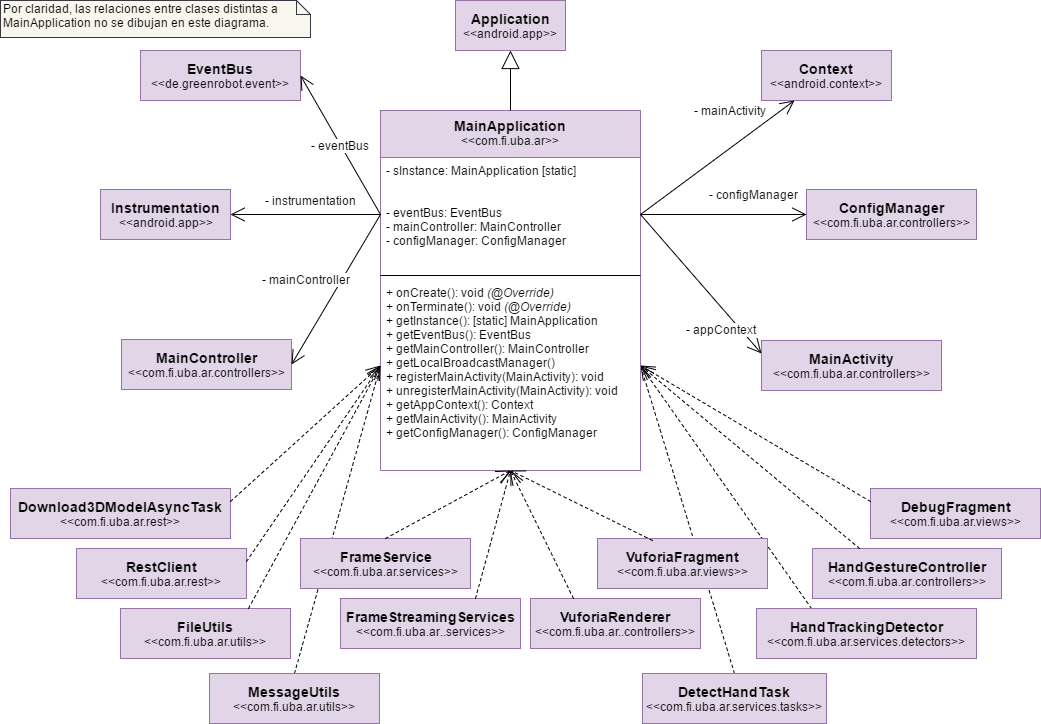
# 

# 

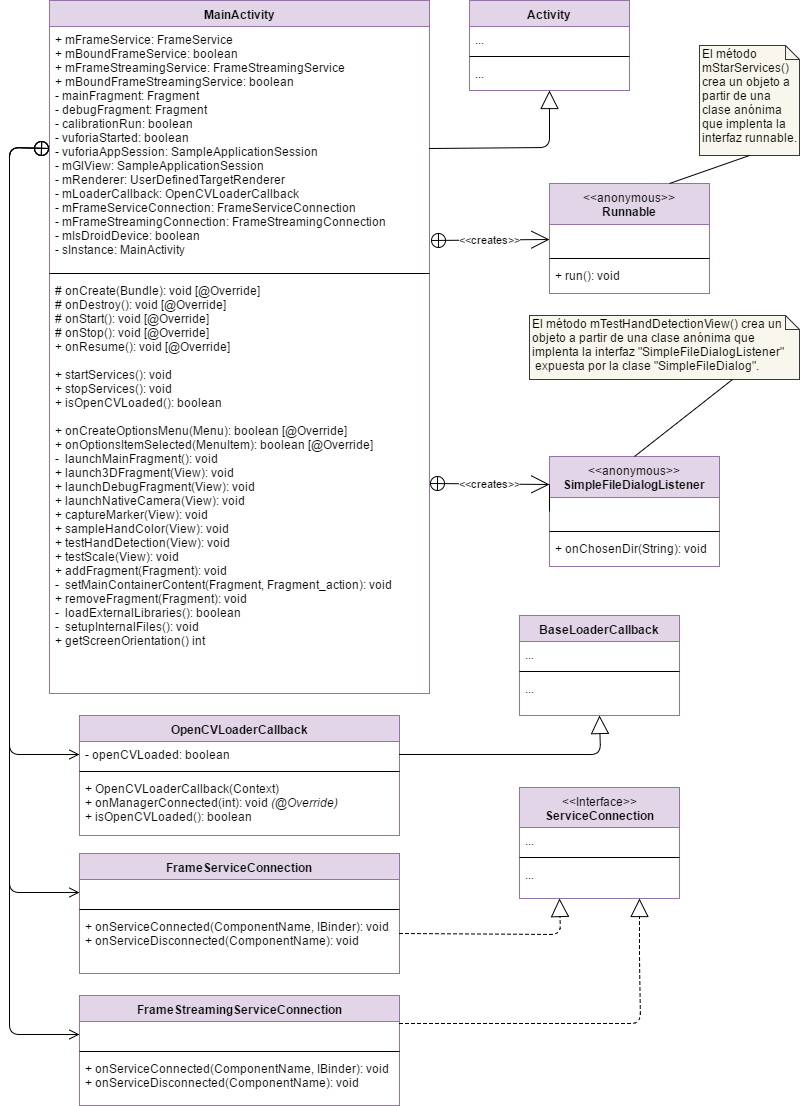
## Diagramas de Clases

Se muestran a partir de este punto, numerosos diagramas de clase que intentan resaltan los componentes de cada una de las más importantes, así como también las clases con las que interactúan y aquellas que la referencian.

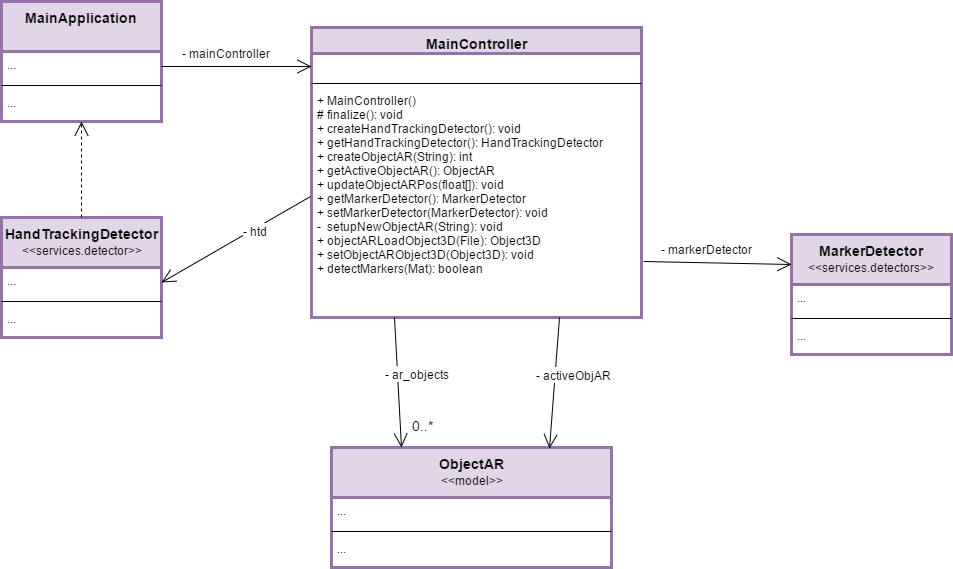
### Clase MainApplication



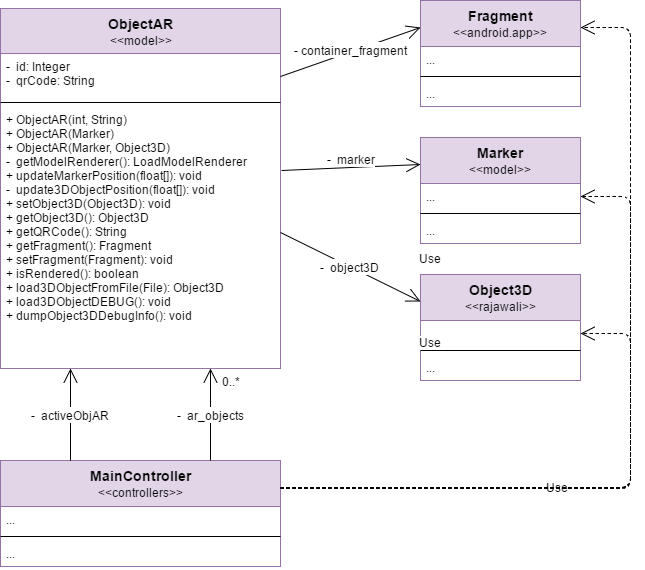
### Clase mainactivity



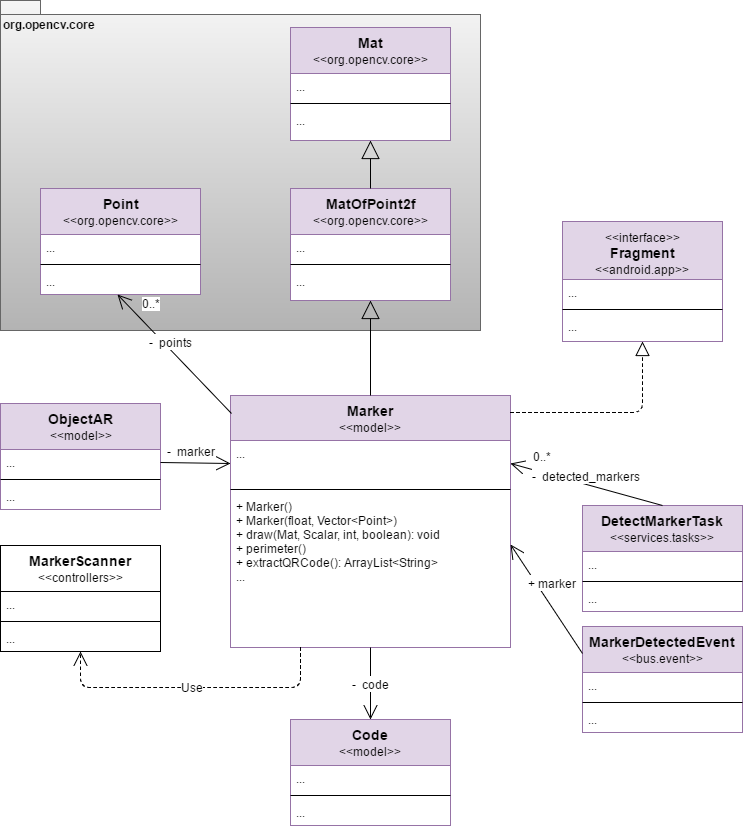
### Clase maincontroller



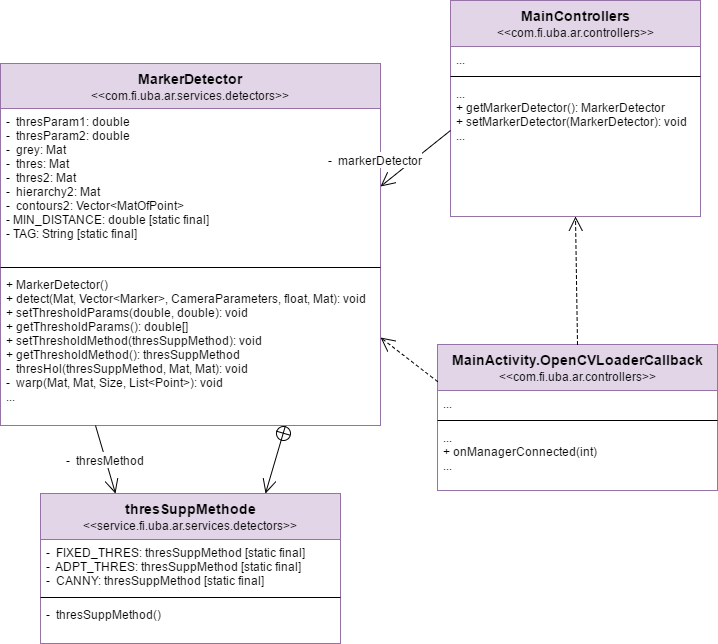
### Clase objectar



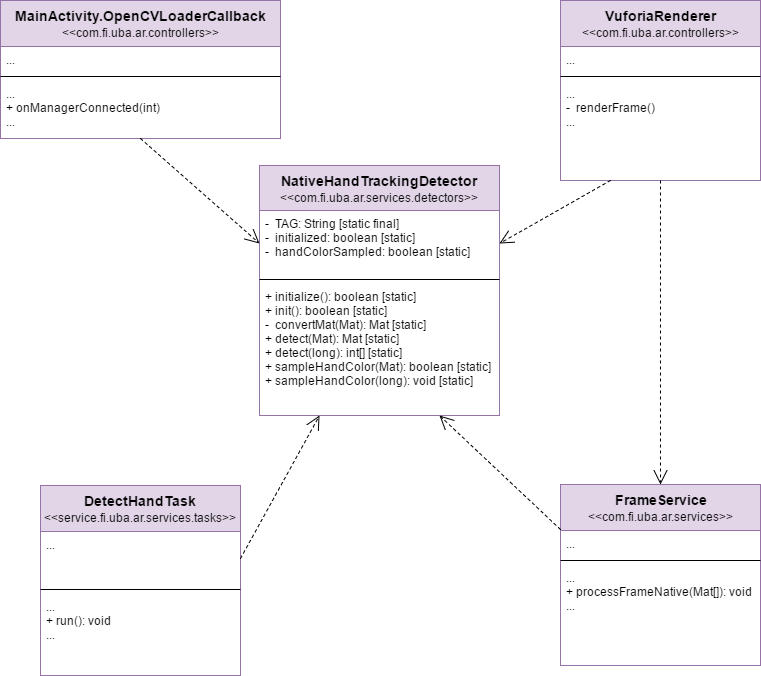
### Clase marker



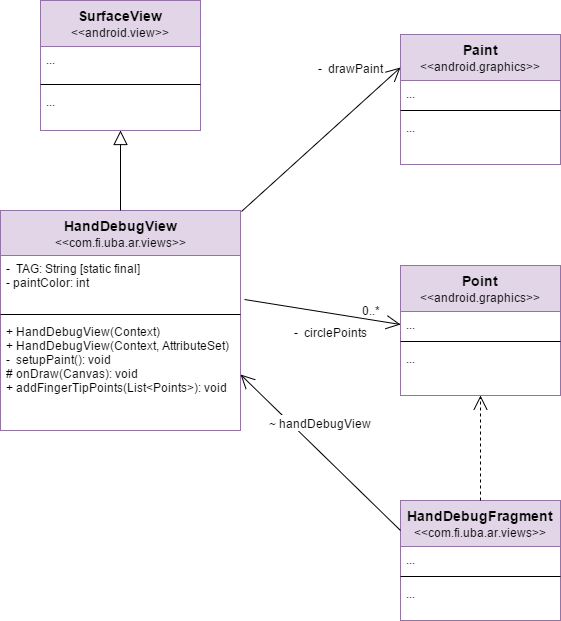
### Clase markerdetector



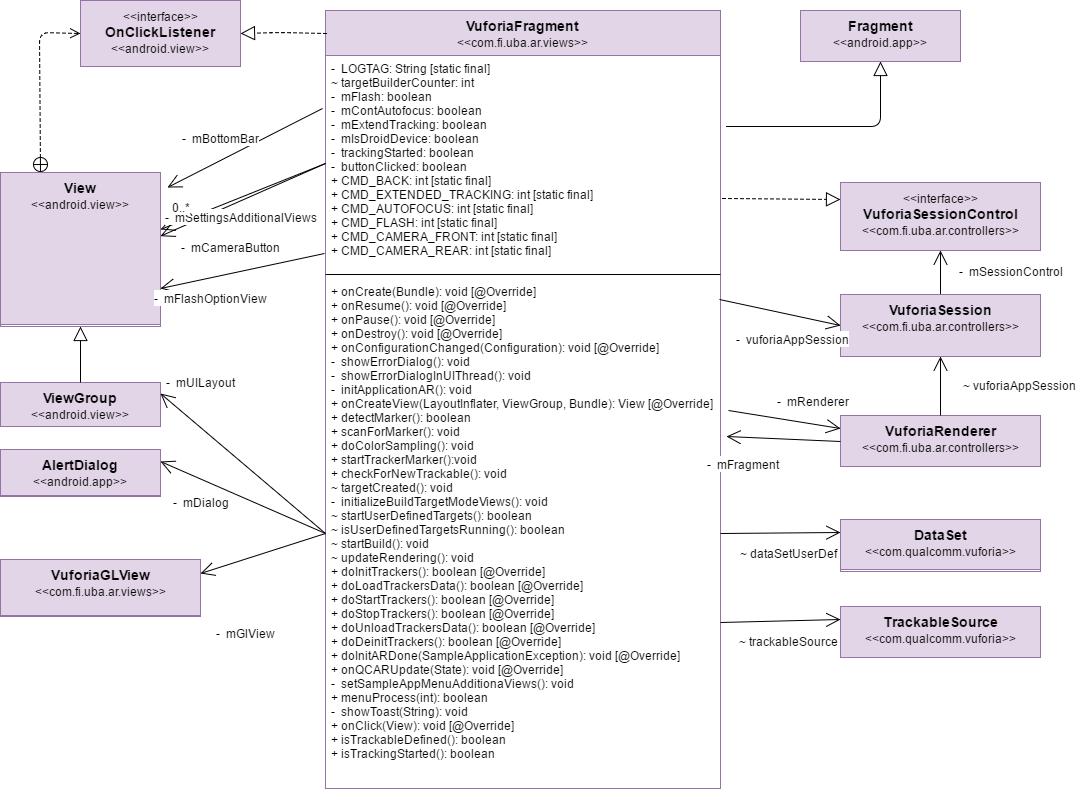
### Clase nativehandtrackingdetector



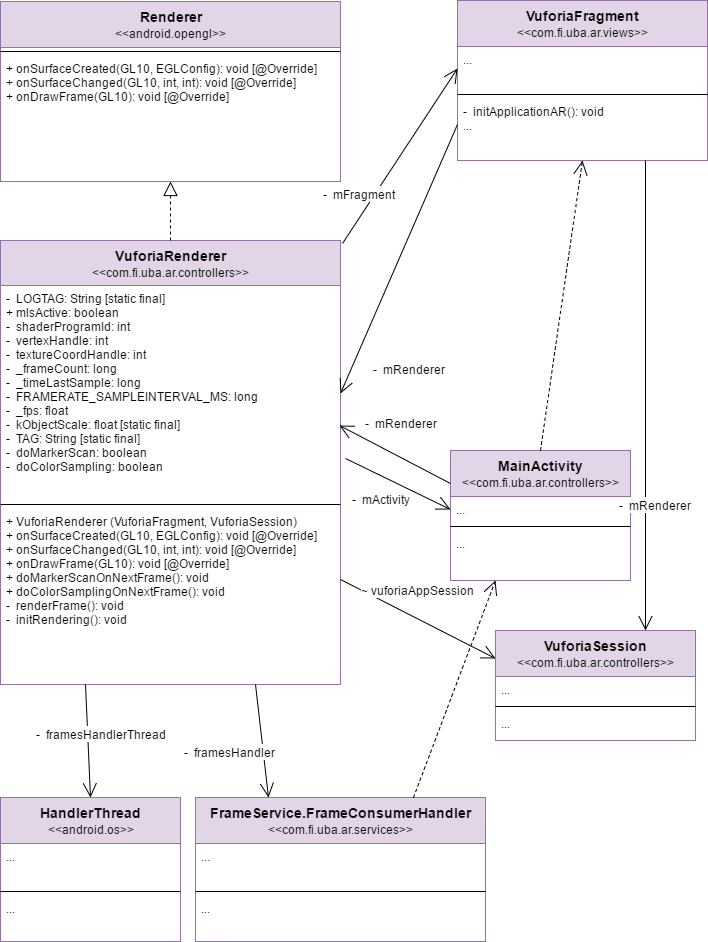
### Clase handdebugview



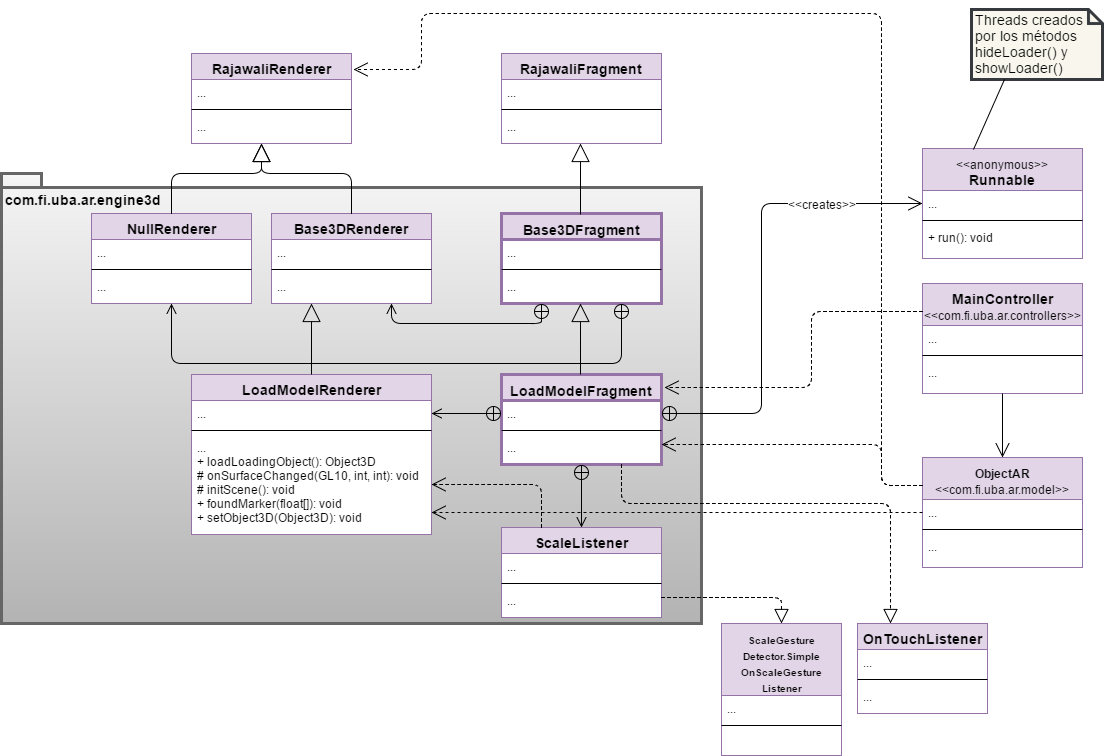
### Clase vuforiafragment



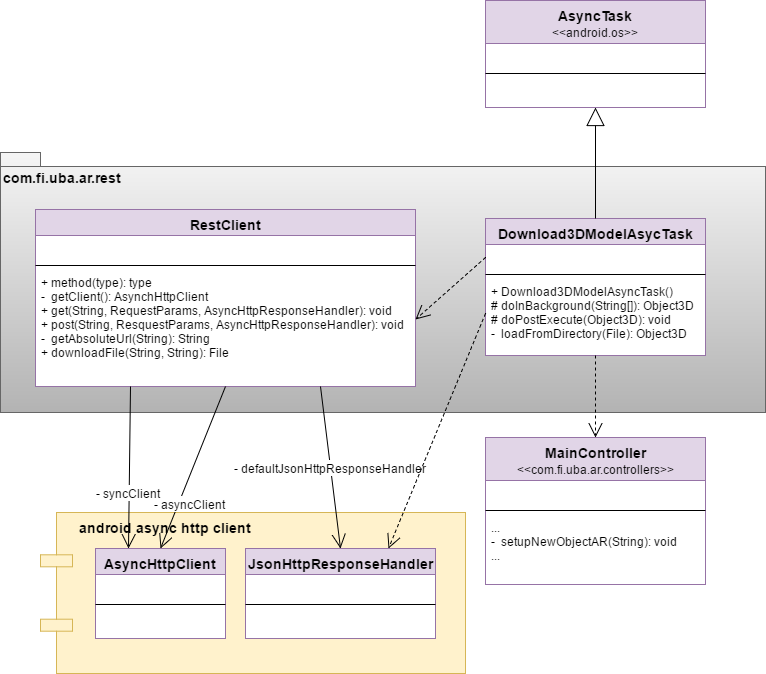
### Clase vuforiarenderer



### Clases del paquete engine3d: Base3dFragment y modelfragment

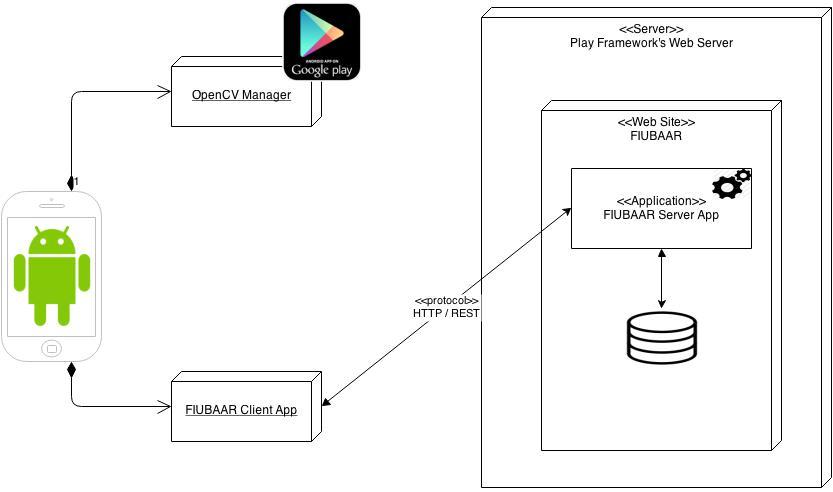


### Clases del paquete rest: RestClient y download3dmodelasynctask



Por otra parte, es necesario señalar que estos diagramas representan a la Aplicación Cliente de FIUBAAR. Esta es la que representa el mayor esfuerzo de desarrollo, y por esta razón se muestra con todo detalle. Para entender la Aplicación FIUBAAR veamos el diagrama de despliegue.

## Diagrama de Despliegue

La aplicación FIUBAAR Cliente es la única de las desarrolladas en el presente proyecto que el usuario final conocerá, y deberá descargarla desde Google Play. Para que funcione, deberá además descargar OpenCV Manager, también desde Google Play. Ambas aplicaciones “vivirán” en el dispositivo móvil del usuario. Sin embargo, el proyecto contempla la construcción de otra aplicación, la FIUBAAR Server App, que es aquella desde la cual la aplicación cliente descargará los modelos destinados a “aumentar la realidad”, dependiendo del identificador reconocido en el marker.

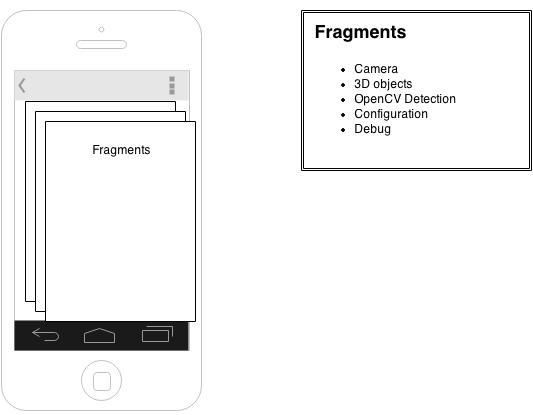
La aplicación server “vivirá” en un servidor remoto, y la comunicación entre ambas -siempre iniciada por el cliente- se realizará mediante servicios REST sobre HTTP. Nos valdremos del Framework Play para implementar esta aplicación.

# 

# 

## Diseño de la Interfaz de Usuario

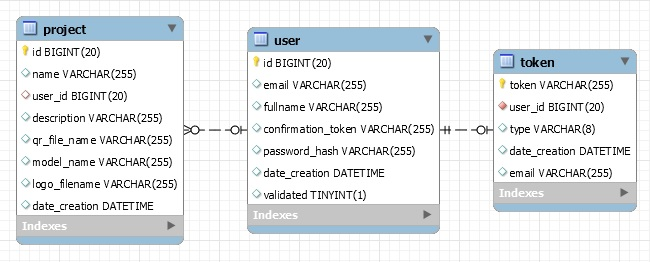
Es interesante mostrar que para la interacción con el usuario se ha recurrido a la utilización de “fragments”, lo cuales representan vistas que se superponen entre sí. De este modo, un fragment mostrará la imagen tal cual es registrada por la cámara (NativeCameraFragment), en otro mostraremos los markers reconocidos resaltados (OpenCVCameraFragment) y en un tercer fragment cobrarán vida los objetos 3d que aumentarán la realidad (LoadModelFragment). De esta forma hemos mantenido aislados entre sí, tres conceptos que forman parte de la aplicación pero que no deben entrelazarse en la vista. Algo así como separación de incumbencias (*separation of concerns*) aplicado a reducir el acoplamiento entre componentes de la vista.



## Modelo de datos

El modelo de datos en esta instancia está enfocado principalmente a reunir los datos de cada usuario y relacionarlos con los múltiples proyectos sobre los que puede accionar como propietario.

Los modelos 3D a visualizarse no forman parte de la base de datos propiamente dicha, sino que son archivos .zip separados, los cuales son referenciados por la BD a partir de su nombre en el sistema de archivos (*file* system).



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tabla | **user** | | | |
| Guarda la lista de usuarios del sistema, habilitados a operar o a la espera de validación. | | | | |
| Columnas |  | | | |
| id | PK | bigint(20) | Id del usuario. | |
| email | Único | varchar(255) | e-mail del usuario. | |
| fullname |  | varchar(255) | Nombre completo del usuario. | |
| confirmation\_token |  | varchar(255) | Token enviado a su dirección de correo electrónico -integrado en un hipervínculo- para verificar la autenticidad de su e-mail. | |
| password\_hash |  | varchar(255) | Hash obtenido a partir del password suministrado por el usuario. | |
| date\_creation |  | datetime | Fecha y hora de la validación del usuario. | |
| validated |  | tinyint(1) | En '1' si el usuario validó su dirección de correo electrónico. | |
| Relaciones | | | | |
| Nombre de la relación | Tabla FK | Tabla PK | Join | Card. |
| fk\_project\_user\_1 | project | user | user\_id = id | 0..\* |
| fk\_token\_user\_1 | token | user | user\_id = id | 0..1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tabla | **project** | | | |
| Guarda la información de todos los proyectos -con sus características o referencias a sus componentes- para cada usuario. | | | | |
| Columnas |  | | | |
| id | PK | bigint(20) | Id del proyecto. | |
| name | Único | varchar(255) | Nombre del proyecto. | |
| user\_id | FK | bigint(20) | FK con tabla *<<user>>*. | |
| description |  | varchar(255) | Descripción del proyecto. | |
| qr\_file\_name |  | varchar(255) | Ruta del código QR creado para el proyecto. | |
| model\_name |  | varchar(255) | Ruta al archivo .zip que contiene el modelo 3D a recrear en la aplicación móvil. | |
| logo\_filename |  | varchar(255) | Ruta al logo del proyecto. | |
| date\_creation |  | datetime | Fecha y hora de la creación del proyecto. | |
| Relaciones | | | | |
| Nombre de la relación | Tabla FK | Tabla PK | Join | Card. |
| fk\_user\_project\_1 | user | project | user\_id = id | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tabla | **token** | | | |
| Contiene información necesaria para la validación de la dirección de e-mail del usuario. | | | | |
| Columnas |  | | | |
| token | PK | varchar(255) | Id del token. | |
| user\_id | FK | bigint(20) | FK con tabla <<user>>. | |
| type |  | varchar(8) | Sin uso. | |
| date\_creation |  | datetime | Fecha y hora de creación del token. | |
| email |  | varchar(255) | E-mail al que fue enviado. | |
| Relaciones | | | | |
| Nombre de la relación | Tabla FK | Tabla PK | Join | Card. |
| fk\_user\_token\_1 | user | token | user\_id = id | 1 |

## Conclusión

La arquitectura básica fue definida como se ha expuesto, pero aún estará sujeta a refinamientos que puedan considerarse convenientes durante la evolución del proyecto, siempre en busca de mejorar la performance en actividades que son intensivas en cuanto al consumo de potencia de procesamiento y que a la vez requieren una respuesta sumamente veloz a efectos de no afectar negativamente la percepción de calidad por parte del usuario.

# Setup Entorno de Desarrollo

## Objetivo

El presente documento representa una guía práctica para que un programador pueda configurar completamente un entorno de desarrollo que le permita trabajar en el proyecto de la aplicación FiubaAR.

El documento apunta a cubrir las plataformas Linux y Windows ya que esto es indistinto para el resultado final. Todas las herramientas necesarias se encuentran disponibles para ambas plataformas, lo que permite a cada programador elegir aquella en la que encuentre mayor comodidad.

## Setup

### Requerimientos

Java JDK (version 7.x)

<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>

Android ADT Bundle (contiene Android SDK)

<http://developer.android.com/sdk/index.html>

Android Native Development Kit (NDK)

<https://developer.android.com/tools/sdk/ndk/index.html>

OpenCV for Android - (android-sdk)

<http://sourceforge.net/projects/opencvlibrary/files/opencv-android/>

Activator

<https://www.typesafe.com/activator/download>

### Configuración

Se debe proceder a instalar en primer lugar Java JDK.

Luego se puede descomprimir el archivo de Android ADT Bundle siguiendo los pasos descriptos en la guía: <http://developer.android.com/sdk/installing/index.html>

Descomprimir Android NDK

Iniciar Android ADT y seguir los pasos descriptos en la siguiente guía de OpenCV para importar los proyectos OpenCV y compilar la librería que luego deberemos importar en nuestros proyectos: <http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html>

Al compilar el proyecto OpenCV obtendremos el archivo /OpenCV-2.4.8-android-sdk/sdk/java/bin/opencv library - 2.4.8.jar (o la versión de OpenCV descargada).

La librería OpenCV generada se deberá importar como dependencia en los proyectos de Android que luego se creen.

ARUCo: ver item “Investigación Detección de Markers” en la documentación de FIUBAAR, allí se muestra el sitio desde el cual se descarga el código de ArUCO y cómo debemos manejarlo para incorporarlo a nuestro proyecto.

Rajawali: 3DEngine. <https://github.com/MasDennis/Rajawali>

### Alternativa

Una vez instalado el Java JDK 7 o superior, existe una alternativa que facilita enormemente el armado del entorno de desarrollo. Se trata del Tegra Android Development Pack (TADP), el cual descarga de Internet e instala las herramientas de desarrollo necesarias y las configura. Adicionalmente contiene otras herramientas que no usaremos en este proyecto, pero la sencillez de uso sugiere al TADP como una opción muy recomendable si no hay limitaciones en recursos. En su versión actual contiene, además de muchas librerías y herramientas, lo siguiente:

* Android SDK r23.0.2
* Android APIs
* Android NDK r9d
* Android Build Tools r19.1
* Android Platform Tools r20
* Android Support Library r20
* Google USB Driver r10
* JDK 1.6.0\_45
* Cygwin 1.7.29
* Eclipse 4.3, CDT 8.2.0, ADT 23.0.2
* Apache Ant 1.8.2

Esta es la opción recomendada para una instalación desde cero, según propone:

<http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/android_dev_intro.html#android-dev-intro>

No obstante, para el desarrollo del presente trabajo, hemos instalado cada uno de los componentes por separado, tal como se describió anteriormente.

### Definición de variables de entorno

El proyecto hará uso de algunas variables de entorno para la compilación y referencia de ciertas librerías. Por lo tanto, se deberán definir las siguientes variables de entorno en el sistema operativo

OPENCV\_HOME="path al proyecto /OpenCV-2.4.9-android-sdk"

NDKROOT="path al directorio de android-ndk"

ACTIVATOR=”path al directorio de Activator”

En Windows se pueden definir siguiendo los pasos aquí expuestos: <http://www.computerhope.com/issues/ch000549.htm>

En Linux se puede ejecutar en una terminal

export OPENCV\_HOME="<path>/OpenCV-2.4.9-android-sdk"

export NDKROOT="<path>/android-ndk-r9"

Es conveniente también editar el archivo $HOME/.bashrc y agregar al final la definición de estas mismas variables para que se definan automáticamente en cada reinicio de sistema.

## Dispositivos Android

Es posible ejecutar las aplicaciones desarrolladas tanto en un emulador como en un dispositivo real.

Es preferible realizar la ejecución de la aplicación en un dispositivo real ya que para el procesamiento de imágenes se necesita contar con una buena velocidad en términos de cuadros por segundo (*FPS, Frame Per Second*), que se consigue con una cámara real y no así tanto con el emulador utilizando una webcam.

Para poder conectar un dispositivo real y hacer uso del mismo desde el IDE, es necesario seguir los pasos que indica la documentación oficial de Android:

<http://developer.android.com/tools/device.html>

Se indica allí como configurar dispositivos tanto para Linux como Windows (donde es necesario instalar un driver USB)

## Herramientas adicionales

Bitbucket (Repositorio GIT privado, Issue tracking, Wiki y Team management): Registrar una cuenta en el sitio <https://bitbucket.org> y pedir una invitación al repositorio del equipo FiubaAR vía email a [proyecto.fiubaar@gmail.com](mailto:proyecto.fiubaar@gmail.com)

Cliente para repositorio GIT: Se puede utilizar cualquier cliente para GIT ya sea mediante línea de comando o algún otro con interfaz gráfica. En particular, es conveniente el uso del plugin de eclipse para GIT, EGit (<http://www.eclipse.org/egit/>). Para Windows se recomienda usar msysgit (<http://msysgit.github.io/>).

Aplicación para diagramas UML: Descargar Modelio desde su sitio <http://www.modelio.org/>. Esta herramienta está basada en Eclipse, así que simplemente se descomprime y se ejecuta. Otra opción consiste en recurrir a una herramienta web como <https://www.draw.io/>

Gradle (build & dependency management): Android ADT Bundle ya posee integrado lo necesario para hacer uso de Gradle. Más información sobre la herramienta se puede encontrar en <http://tools.android.com/tech-docs/new-build-system/user-guide>

GenyMotion (<https://cloud.genymotion.com/page/doc/>): un emulador alternativo mucho más veloz que el suministrado por Google. Está basado en virtual box y tiene disponible un plug-in para Eclipse que nos permite iniciar los dispositivos desde allí. Está basado en arquitectura x86, por lo que resulta necesario compilar el código nativo también para x86.

## Notas Adicionales

Si bien en el sitio oficial de Android existen varias referencias a la herramienta Android Studio (que actualmente se encuentra en Beta), no podemos hacer uso completo de la misma ya que aún no cuenta con soporte para Android NDK.

Android NDK es necesario para poder compilar código nativo en C/C++ en los casos en los que se desea hacer uso de librerías de más bajo nivel con interfaces JNI. Este tipo de situación en el que será necesario realizar implementación en C/C++ lo encontraremos en el caso de la detección de manos y gestos, ya que para dicha funcionalidad será fundamental la performance y velocidad en el procesamiento de frames.

## Buenas Prácticas

Los programadores que formen parte del proyecto deberán en lo posible seguir buenas prácticas en todos los aspectos comprendidos en la implementación de la aplicación.

Se describen a continuación algunas de las más importantes.

### Comentarios

Agregar comentarios en el código en aquellos lugares en los que se realicen operaciones complejas. Se recomienda ser consciente y considerado con la cantidad de comentarios y no sobrepasarse con esto. [[9]](http://javarevisited.blogspot.com.ar/2011/08/code-comments-java-best-practices.html)[[10]](http://www.hongkiat.com/blog/source-code-comment-styling-tips/)[[11]](http://java.dzone.com/articles/5-best-practices-commenting)

Hacer uso de JavaDoc en las clases y métodos. Esto permitirá luego generar automáticamente la documentación de la aplicación y API’s. Los IDE’s en uso auto-completan el template de JavaDoc si se comienza a comentar sobre un método o clase con /\*\* y se presiona ENTER. [[10]](http://stackoverflow.com/questions/1777175/how-can-i-generate-javadoc-comments-in-eclipse?rq=1)[[11]](http://www.jetbrains.com/idea/features/javadoc.html#link6)

### Desarrollo en Android

<http://developer.android.com/guide/practices/index.html>

<http://developer.android.com/training/index.html> (contiene buenas prácticas para algunos cuantos aspectos del desarrollo para Android)

<http://www.slideshare.net/retomeier/being-epic-best-practices-for-building-android-apps>

<http://forum.xda-developers.com/showthread.php?t=2635275>

## Repositorio

Una vez aceptada la invitación para formar parte del team FiubaAR en BitBucket (<https://bitbucket.org/fiubaar>) se tendrá acceso al repositorio git.

Existen varios “workflows” con los cuales se trabaja comúnmente con git, siento “gitflow” (<https://www.atlassian.com/git/workflows#!workflow-gitflow>) uno de los más conocidos.

A pesar de que “gitflow” presenta un esquema de trabajo interesante y que puede aplicarse a cualquier desarrollo, no se hace uso de este workflow estrictamente, sino que se plantea algo diferente.

El workflow de trabajo para el presente proyecto utilizará un branch “master” default inicial, pero se creara un branch nuevo por cada sprint (llamados “sprintN” donde N será el número de sprint).

No se harán “merge” de estos branches ya que cada sprint tendrá un conjunto de entregables y una versión intermedia de la aplicación.

Solo al final del proyecto, una vez que se cuente con la versión final completa de la aplicación, se tendrá en el branch “master”, la release final de todo lo necesario.

Para más información sobre cómo utilizar GIT en general se puede recurrir a los siguientes tutoriales:

<https://www.atlassian.com/git/tutorial>

<https://confluence.atlassian.com/display/BITBUCKET/Using+Git+branches>

### Uso de git en el proyecto

Se debe crear un archivo .gitignore con el contenido indicado en <https://www.gitignore.io/api/android,java,c++,eclipse> de modo de no contaminar el repositorio con archivos que no deben incluirse en él. Así mismo, y de acuerdo a lo expuesto en <https://help.github.com/articles/ignoring-files> , es necesario saber que una vez que se haya aplicado un check out bajando archivos que debieron estar en el .gitignore, deberá ejecutarse el siguiente comando para que estas reglas se apliquen también a ellos:

$ git rm --cached

## Referencias

[1] <https://developer.android.com/about/dashboards/index.html>

# Investigación de Markers

## Introducción

Se conoce como “marker” en Realidad Aumentada a una imagen fija, probablemente con algún patrón simple y fácilmente distinguible que la aplicación puede reconocer o detectar.

El tipo de marker comúnmente utilizado por la mayoría de las aplicaciones de realidad aumentada está basado en una imagen en blanco y negro para mayor facilidad de detección debido al contraste de colores y con una imagen geométrica simple que generalmente es un cuadrado.

También existen aplicaciones de realidad aumentada “markerless”, que no hace uso de un marker sino que realizan tracking de cualquier tipo de imagen que no posee características demasiado restrictivas o con patrones fijos.

## Estado del arte

En general, las aplicaciones de realidad aumentada poseen una cantidad limitada y fija de markers que son capaces de reconocer. Dichos markers no poseen información adicional y consisten simplemente en una imagen fija que la aplicación puede reconocer fácilmente.

Ejemplo de esto son los markers utilizados por frameworks como ARToolkit

<http://www.artoolworks.com/support/library/Creating_and_training_new_ARToolKit_markers>

Existen herramientas que permiten generar nuevos markers con los patrones que uno desee

<http://flash.tarotaro.org/blog/2008/12/14/artoolkit-marker-generator-online-released/>

<http://www.arined.org/?page_id=6>

## Requerimientos FiubaAR

En el caso de la aplicación FiubaAR es necesario que aquel marker que se utilice pueda contener información reconocible en su patrón.

Es por esto que dicho marker deberá estar compuesto por dos elementos:

Patrón fijo y reconocible, que llamaremos *marker contenedor* o *patrón exterior*

Código 2D que pueda contener información, que llamaremos *marker núcleo* o *patrón interior*.

Existe hoy en día una aplicación de realidad aumentada que utiliza un marker con el concepto aquí expuesto. Esta aplicación es Zappar (<http://www.zappar.com/>) y utiliza un marker que llaman Zapcode (<https://zapcode.it/>)

Se puede apreciar que en el caso del Zapcode, los componentes del marker se encuentran invertidos ya que el patrón interior es fijo y es el patrón exterior el que cuenta con la información adicional dinámica. Su página explica claramente el funcionamiento <https://zapcode.it/how-zapcodes-work/> y presentan una comparación con los códigos QR, explicando porque los zapcodes son -a su criterio- mejores <https://zapcode.it/zapcodes-vs-qr/>

Para poder definir la forma más conveniente de componer el marker completo que utilizará la aplicación FiubaAR deberemos analizar los diferentes tipo de códigos de barras 2D.

<http://en.wikipedia.org/wiki/Barcode>

<http://en.wikipedia.org/wiki/Mobile_tagging>

## Elección del marker núcleo

Es conocido que el código QR es uno de los más difundidos y comúnmente reconocido por el público en general. Esto lo convierte en potencialmente el tipo de código más conveniente para la implementación, ya que podría ser fácilmente incluido en un patrón cuadrado de alto contraste tales como los que utiliza ARToolkit y además facilitaría el desarrollo el hecho de que existen ya librerías capaces de proveer su reconocimiento, como así también el generarlos.

## Tracking del marker

La aplicación deberá poder realizar tracking del marker para determinar la posición y luego proyectar allí un objeto 3D. Es importante notar que el reconocimiento del marker núcleo (código QR) es solo necesario realizarlo una única vez para obtener la información necesaria que indica desde donde obtener el modelo de objeto 3D.

Esto entonces implica que solo será necesario realizar tracking del marker contenedor (patrón fijo exterior) lo que simplificaría notoriamente el algoritmo utilizado.

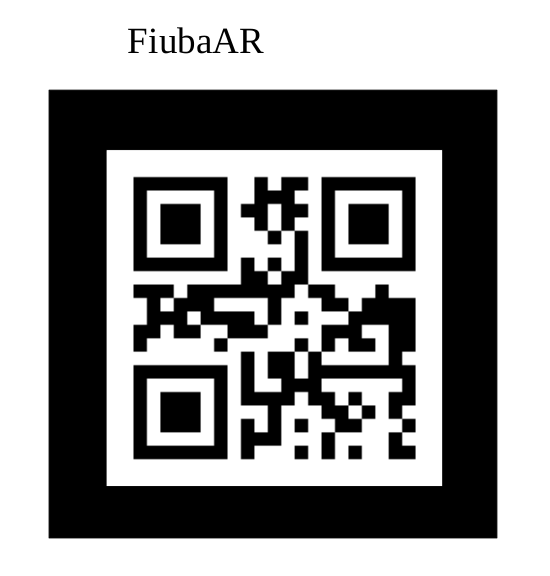
## Definición de marker

Tal como se ha explicado anteriormente el marker completo está compuesto por un contenedor y un núcleo.

No existen medidas estrictamente necesarias para el contenedor, y la única restricción es que posea una forma rectangular con un borde de ancho considerable de color negro. Cabe aclarar que no hace falta que ambos lados posean la misma medida para convertirlo en un cuadrado perfecto.

El núcleo será un código QR que debe estar separado por al menos algunos milímetros del borde del borde contenedor para facilitar la lectura.

A continuación se presenta un marker completo a modo de ejemplo:



## Documentación adicional útil

<http://qrlive.bluegouda.com/>

<https://github.com/zxing/zxing> (librería que reconoce varios tipos de código y posee una implementación para Android)

<http://dbis.eprints.uni-ulm.de/986/1/BA_Hartmann_13.pdf> tesis, cuyo código está disponible (<https://github.com/xamino/bachelor_framework_git>), en donde se implementa detección de marker que si bien no es QR, se implementa el reconocimiento.

<http://en.wikipedia.org/wiki/Fiduciary_marker>

Comparación de AR con marker y sin marker. Menciona otros formatos además de QR

<http://researchguides.dartmouth.edu/content.php?pid=227212&sid=1891183>

<http://stackoverflow.com/questions/22743213/detect-the-size-of-a-qr-code-in-python-using-opencv-and-zbar>

<http://stackoverflow.com/questions/22145084/scanning-qr-codes-using-opencv-and-zxing-for-android>

<http://karanbalkar.com/2013/12/tutorial-65-implement-barcode-scanner-using-zxing-in-android/>

<http://huaizhiyazi.wordpress.com/2014/04/15/can-you-give-a-list-of-popular-android-libraries-that-are-commonly-used-to-build-applications/>

<http://dsynflo.blogspot.com.ar/2010/03/2d-barcodes-form-data-matrix-to-qr-code.html>

Se puede generar dinámicamente una imagen QR con Google charts con

[http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=|1&chl=TEXTO\_ACA](http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=%7C1&chl=TEXTO_ACA)

<http://blog.ayoungprogrammer.com/2013/07/tutorial-scanning-barcodes-qr-codes.html>

<http://blog.ayoungprogrammer.com/2014/04/real-time-qr-code-bar-code-detection.html>

<http://www.blackdogfoundry.com/blog/zbar-bar-code-qr-code-reader-android/>

<https://github.com/dm77/ZBarScanner>

# Investigación Detección de Markers

## Introducción

Existen varias implementaciones para realizar detección de Markers utilizando OpenCV.

La idea es realizar una recolección de información acerca de cada una de las implementaciones disponibles y luego testear cada una de ellas para verificar su funcionamiento y arquitectura.

## Implementaciones analizadas

**Aruco**: a minimal library for Augmented Reality applications based on OpenCV.

**Sitio:** <http://www.uco.es/investiga/grupos/ava/node/26>

Esto es una librería en C++ que parece presentar la posibilidad de realizar una compilación para plataforma Android.

AndroidARMarkerDetection: Android port of Aruco

**Sitio:** <https://github.com/dszafir/AndroidARMarkerDetection>

Parece ser un port directo de la clase MarkerDetector de Aruco a una implementación en Java utilizando OpenCV

**ARma library:** Pattern tracking for Augmented Reality

**Sitio:** <http://xanthippi.ceid.upatras.gr/people/evangelidis/arma/>

Librería en C++ que presenta una implementación más simple que Aruco

Implementation of a Java Framework for Marker Based Detection in Augmented Reality

**Sitio:** <http://dbis.eprints.uni-ulm.de/986/1/BA_Hartmann_13.pdf>

Proyecto final de tesis que implementa una aplicación para Android que realiza detección y tracking de markers para realidad aumentada. La implementación es con OpenCV y puramente en Java. La implementación del proyecto se encuentra en

<https://github.com/xamino/bachelor_framework_git>

Marker Detection for Augmented Reality Applications

**Sitio:** <http://www.infi.nl/blog/view/id/56/Marker_Detection_for_Augmented_Reality_Applications>

Presenta la implementación y explicación de un algoritmo para detectar markers cuadrados.

OpenCV Square Tracking Android JNI

**Sitio:** <https://github.com/victorkp/OpenCV-Square-Tracking-Android-JNI>

Presenta una aplicación para Android que realiza la detección de cuadrados con OpenCV cuya implementación está realizada con JNI.

## Documentación útil

Además de las implementaciones antes mencionadas encontramos artículos que proveen información útil adicional.

<http://iplimage.com/blog/cv-img-tec-black-white-marker-detection/>

Este artículo presenta la idea básica detrás de un algoritmo elemental para realizar la detección de markers con forma cuadrada.

<http://qtandopencv.blogspot.com.ar/2013/10/perspective-correction-for.html>

El artículo explica cómo realizar una corrección de perspectiva en una imagen. Esto es esencial en el momento en el que se detecta un marker y se debe analizar la información dentro del mismo. Si el marker no se encuentra de frente a la cámara y existe inclinación, esta debe compensarse para poder leer los datos.

<http://opencv-code.com/tutorials/automatic-perspective-correction-for-quadrilateral-objects/>

Este artículo también explica cómo realizar la corrección de perspectiva, con una clara explicación del los pasos del algoritmo utilizado.

## Testing de implementaciones

### Aruco

Compila directamente con todos los valores por defecto de proyecto descargado desde <http://sourceforge.net/projects/aruco/files/1.2.5/>

Ejecutando la aplicación de testing “aruco\_test live” que toma los frames desde una webcam podemos ver que funciona detectando los markers rápidamente siendo estable sin presentar crashes. Al detectar markers se puede ver que la librería tiene información de tracking como si fueran algún estilo de coordenadas y detecta los markers con algunos ángulos muy interesantes respecto a la cámara que realiza la captura.

Inicialmente, no logramos compilar el código para la plataforma Android y pero se estimó que quizás solo fuera cuestión de modificar archivos CMake.

Esta librería parece ser una muy buena opción estable para detectar y realizar tracking de markers, por lo tanto consideramos que debíamos invertir más tiempo en lograr realizar la compilación para Android o, de no obtenerse esto, al menos una adaptación para Android con solo lo mínimo indispensable.

Para compilar para Android hay que bajar una versión actualizada de android-cmake para ndk r9 desde <https://github.com/taka-no-me/android-cmake> ya que vemos algunos errores como esto <https://code.google.com/p/android-cmake/issues/detail?id=15>

Además, fue necesario hacer lo siguiente:

|  |
| --- |
| export ANDROID\_NDK=/opt/android-ndk-r9  export CMAKE\_C\_COMPILER=/opt/android-ndk-r9/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-x86\_64/bin/arm-linux-androideabi-gcc  export CMAKE\_CXX\_COMPILER=/opt/android-ndk-r9/toolchains/arm-linux-androideabi-4.8/prebuilt/linux-x86\_64/bin/arm-linux-androideabi-g++  sudo cp /opt/OpenCV-2.4.9-android-sdk/sdk/native/jni/OpenCVModules\_armeabi.cmake /usr/local/share/OpenCV/  sudo chmod 644 /usr/local/share/OpenCV/OpenCVModules\_armeabi.cmake  cd /aruco-1.2.5/android  # Por defecto el cmake nos genera todo para compilar aruco como librería estática (extensión .a)  # pero es más fácil si compilamos como librería shared (extensión .so)  # Así al momento de importarla en nuestra implementación en android todo es más simple  # Para esto hay que modificar el archivo CMakeCache.android.initial.cmake  # y buscar donde define la variable BUILD\_SHARED\_LIBS y especificarla en ON, quedando así:  # set(BUILD\_SHARED\_LIBS ON CACHE BOOL "" )  # Además, modificamos el script cmake\_android\_armeabi.sh para incluir en la línea de cmake el siguiente parámetro:  # -DOpenCV\_DIR=/opt/OpenCV-2.4.9-android-sdk/sdk/native/jni  sh ./scripts/cmake\_android\_armeabi.sh  # hay que modificar algunos flags porque quedaron mal los includes  # buscamos el archivo build\_armeabi/src/CMakeFiles/aruco.dir/flags.make  # -I/usr/sdk/native/jni/include se modifica por -I/opt/OpenCV-2.4.9-android-sdk/sdk/native/jni/include  # Una vez modificado eso hacemos:  cd build\_armeabi/  make |

Esto va a compilar una librería en /aruco-1.2.5/android/build\_armeabi/libs/armeab/libaruco.so que deberíamos poder utilizar en alguna implementación con JNI en nuestra aplicación Android.

Más tarde, también comprobamos que es posible compilar directamente con android NDK si se agregan todos los archivos fuente (cpp y hpp) de aruco en un proyecto y se configura correctamente el archivo Android.mk.

Algo interesante a destacar de esta librería, es que presenta la posibilidad de extenderla para detectar markers con un formato diferente al que Aruco maneja por defecto.

No existe suficiente documentación sobre la librería excepto lo explicado en la página web de la librería, pero el código presenta gran cantidad de comentarios e información documentando las clases, métodos y funciones.

Se puede ver en el archivo *aruco/markerdetector.h* que el método setMakerDetectorFunction de la clase MarkerDetector puede ser utilizado para registrar una nueva función callback que realice la detección de un marker diferente. Esta callback debe respetar una cierta estructura donde recibirá como parámetro una imagen que contendrá un “potencial” marker. Estos potenciales markers serán regiones rectangulares que posean el color negro.

Esto simplifica en gran medida el trabajo necesario para la detección y tracking de los markers definidos previamente para la aplicación FiubaAR.

Encontramos una página que realiza una evaluación técnica de la librería

<http://iplimage.com/blog/aruco-technical-evaluation/>

### ARma

Para poder compilar tuvimos que manualmente crear el archivo CMakeLists.txt con la configuración necesaria para compilar los archivos cpp.

El código descargado de su sitio para OpenCV 2.4.3 no compila directamente, presentando varios errores. Se realizaron las correcciones necesarias en el código, en algunos include, tipos de datos y se logró compilar la librería correctamente.

Realizando pruebas con la aplicación “ARma\_demo” se puede apreciar que la aplicación no es nada estable prácticamente, causando un crash que cierra la aplicación apenas un marker es detectado.

El crash indica lo siguiente:

OpenCV Error: Assertion failed (CV\_IS\_MAT(objectPoints) && CV\_IS\_MAT(imagePoints) && CV\_IS\_MAT(A) && CV\_IS\_MAT(rvec) && CV\_IS\_MAT(tvec)) in cvFindExtrinsicCameraParams2, file /opencv-2.4.8/modules/calib3d/src/calibration.cpp, line 1177

terminate called after throwing an instance of 'cv::Exception'

what(): /opencv-2.4.8/modules/calib3d/src/calibration.cpp:1177: error: (-215) CV\_IS\_MAT(objectPoints) && CV\_IS\_MAT(imagePoints) && CV\_IS\_MAT(A) && CV\_IS\_MAT(rvec) && CV\_IS\_MAT(tvec) in function cvFindExtrinsicCameraParams2

Quizás estos threads ayuden a acomodar el código para no tener problemas con los asserts que causan el crash

<http://answers.opencv.org/question/3023/solvepnp-assertion-failed/>

<https://github.com/kylemcdonald/ofxFaceTracker/issues/15>

### Implementation of a Java Framework for Marker Based Detection in Augmented Reality

El proyecto compila correctamente para Android 4.x con OpenCV 2.4.8 y 2.4.9. La aplicación ejecuta y no dio errores de ejecución, pero aún así no hubo ninguna muestra de que la aplicación detectará marker alguno o mostrara cambio alguno en la ejecución. Quizás no se estaba realizando el testing de manera correcta.

### Marker Detection for Augmented Reality Applications

Para compilar tuvimos que crear manualmente el archivo CMakeLists.txt con la configuración necesaria para compilar los archivos cpp. Además, tuvimos que acomodar varias directivas de #include para utilizar los headers correctos de OpenCV.

Luego de los cambios todo compila correctamente y el algoritmo parece funcionar para detectar los cuadrados con bordes de markers. No se experimentaron crashes con tests simples, pero parece que se debería adaptar bastante el código fuente para obtener la funcionalidad deseada para tracking y no solo detección.

### OpenCV Square Tracking Android JNI

Utilizamos el código de este proyecto para la primera versión básica de la aplicación cliente FiubaAR del sprint 1 donde se pudo comprobar que la detección básica funciona. Aun así, esto solo indica si un cuadrado fue detectado o no. No se realizaron suficientes pruebas con respecto a los ángulos soportados respecto de la cámara.

### Vuforia

Utilizando los ejemplos provistos en el portal de desarrollo de Vuforia (https://developer.vuforia.com/downloads/samples) se pudo adaptar el código correspondiente a “User Defined Targets” para utilizar el port de Aruco para Android de modo de validar que la imagen cumpla con el borde exterior del marker y que contiene un código QR válido en el centro, previo a generar la imagen que se utiliza para el tracking.

Es importante notar que se realizaron pruebas con las versiones del SDK 3.0.9 y 4.2.3.

Si bien la versión 4.x del SDK contiene mejoras y avances, también presenta algunos cambios que no resultaron convenientes para el presente trabajo, entre ellos la inclusión de una marca de agua en la pantalla (<http://developer.vuforia.com/library/articles/FAQ/Watermark> ), el requerimiento de una clave para la aplicación generada en el portal de desarrollo (<http://developer.vuforia.com/library/articles/Training/Vuforia-License-Manager> ) y el cambio de algunas clases que implican un pequeño refactoring del código. Debido a esto se decide hacer uso de la versión 3.0.9 del SDK.

## Conclusiones

Es claro -dados los resultados de las pruebas realizadas- que la librería “Aruco” presenta la implementación más estable que encontramos en proyectos libres y/o opensource.

Adicionalmente, se pudo incorporar el código fuente de Aruco a un proyecto android y compilar con Android NDK sin problemas y sin necesidad de utilizar cmake con todas las modificaciones antes mencionadas. Todo esto sumado a la posibilidad simple de definir un nuevo callback para detectar markers con otro patrón, no dejan dudas de que Aruco es la mejor alternativa disponible a integrar en el proyecto para realizar la detección de markers.

**Revisión Sprint4**: Luego de realizar el test de Vuforia utilizando los “User Defined Targets” se pudo comprobar que simplifica abismalmente la implementación del tracking de markers y la ubicación de objetos 3D pero se limita el soporte solo a plataformas ARM no pudiendo ejecutar en x86 (que de todas formas no es lo más común). La utilización de un SDK como este que presenta gran madurez elimina varios problemas que se presentaron cuando se hacía uso de la implementación de Aruco como librería nativa y así también aumenta en gran medida la estabilidad de la aplicación en general.

# Investigación Detección de código QR

## Introducción

Habiendo realizado la elección de utilizar la librería Aruco para la detección de markers, debemos ahora investigar las posibilidades existentes de detección de código QR.

El primer requisito a cumplir es que la implementación deberá ser en C/C++ para poder aprovechar la velocidad de la implementación nativa mediante JNI, así como para facilitar la integración con Aruco.

## Implementaciones encontradas

**Nombre**: ZBar

**Sitio:** <http://zbar.sourceforge.net/>

**Nombre**: ZXing

**Sitio:** <https://github.com/zxing/zxing/>

**Nombre**: CodeProject Simple QRCode Library

**Sitio:** <http://www.codeproject.com/Articles/593591/Simple-C-Cplusplus-QRCode-Library>

**Nombre**: Quirc

**Sitio:** <https://github.com/dlbeer/quirc>

**Nombre**: QRCode

**Sitio:** <http://qrcode.sourceforge.jp/index.html.en>

## Documentación útil

http://dsynflo.blogspot.com.ar/2010/06/libdecodeqr-libdecodeqr-is-cc-library.html

<https://www.openhub.net/p/zxing/rss_articles?page=5>

<http://stackoverflow.com/questions/17973641/how-to-decode-data-using-zxing-c>

<http://wiki.ssrrsummerschool.org/doku.php?id=robocup2012:qrcode-cppexample>

<http://www.pixcl.com/QR_Code_Decoder.htm>

<http://www.pixcl.com/libdecodeqr_instability.htm>

<https://conra.dk/2013/01/06/a-quicker-qr-code-scanner.html>

<http://stackoverflow.com/questions/18638319/zbar-memory-leak-on-ios>

<http://developer.nokia.com/community/wiki/Qr_Decoder_in_Qt>

## Testing de implementaciones

<http://www.blackdogfoundry.com/blog/zbar-bar-code-qr-code-reader-android/>

ZBar da como resultado problemas de heap corruption (memory leaks quizás)

Se hizo un branch en el repositorio probando quirc -que funcionó- y también dio errores de heap corruption (parecen ser free() de lugares ya liberados).

Se sospecha que podría ser un problema en aruco relacionado al callback, cuando este no es el standard de aruco.

## Conclusiones

Por las referencias y pruebas, tanto ZBar como Quirc son buenas opciones, pero su adopción definitiva dependerá de si puede resolverse o no el problema de memoria que ambas presentan, tal vez no por sí mismas sino por inclusión dentro del código de la aplicación en desarrollo.

En este momento la decisión puede ser revisada e inclinarnos por la utilización de otra de las opciones mencionadas.

# Investigación Engine 3D Android

## Introducción

La aplicación FiubaAR Cliente debe poder realizar carga y rendering de modelos 3D. Para ello se hará uso de un engine 3D para plataforma Android.

## Implementaciones encontradas

**Nombre**: jPCT 3D Engine

**Sitio:** <http://www.jpct.net/jpct-ae/index.html>

**Nombre**: Rajawali

**Sitio:** <https://github.com/MasDennis/Rajawali>

**Nombre**: min3D

**Sitio:** <https://code.google.com/p/min3d/>

**Nombre**: jMonkeyEngine

**Sitio:** <http://jmonkeyengine.org/>

**Nombre**: libGDX

**Sitio:** <http://libgdx.badlogicgames.com/>

## Documentación útil

A continuación se presenta una lista de links con información útil y ejemplos.

<https://github.com/MasDennis/Rajawali/wiki>

Integración de Rajawali con Vuforia SDK <https://github.com/MasDennis/RajawaliVuforia>

Integración de jPCT con Vuforia SDK <http://www.jpct.net/wiki/index.php/Integrating_JPCT-AE_with_Vuforia>

<https://github.com/sidneibjunior/vuforia-jpct>

jMonkeyEngine Android Integration

<http://hub.jmonkeyengine.org/wiki/doku.php/jme3:android>

<http://hub.jmonkeyengine.org/wiki/doku.php/jme3:eclipse_jme3_android_jnindk>

## Conclusiones

De todas las implementaciones encontradas, se encontró que Rajawali es estable, compiló en el primer intento sin necesidad de introducir modificaciones en el código, es ampliamente utilizada (con numerosas menciones a través de distintos sitios) y es parte de muchos otros proyectos. Además, está ampliamente probada la integración con Vuforia, por lo que se considera la mejor opción para el proyecto en curso.

# Investigación Clientes REST Android

## Introducción

La aplicación FiubaAR Cliente debe poder conectarse con el server mediante la invocación de servicios REST. Para ello se hará uso de alguna librería que resuelva la comunicación HTTP y consumo de servicios REST, que nos permitirá solo enfocarnos en la implementación de la lógica de consumo de los datos obtenidos.

## Implementaciones encontradas

**Nombre**: Android Asynchronous Http Client

**Sitio:** <http://loopj.com/android-async-http/>

**Nombre**: Retrofit

**Sitio:** <http://square.github.io/retrofit/>

**Nombre**: Spring for Android

**Sitio:** <http://projects.spring.io/spring-android/>

**Nombre**: Volley

**Sitio:** <https://android.googlesource.com/platform/frameworks/volley/>

**Nombre**: RESTDroid

**Sitio:** <https://github.com/PCreations/RESTDroid>

**Nombre**: RoboSpice

**Sitio:** <https://github.com/stephanenicolas/robospice>

**Nombre**: Simple rest API client library

**Sitio:** <https://github.com/darko1002001/android-rest-client>

## Documentación útil

A continuación se presenta una lista de links con información útil y ejemplos de cómo realizar requests vía HTTP o utilizar algunas de las librerías encontradas.

<https://www.youtube.com/watch?v=xHXn3Kg2IQE>

<http://developer.android.com/training/volley/index.html>

<http://spring.io/guides/gs/consuming-rest-android/>

<http://pivotallabs.com/using-retrofit-rest-client-android-applications/>

<https://raw.githubusercontent.com/stephanenicolas/robospice/master/gfx/RoboSpice-InfoGraphics.png>

## Testing de implementaciones

Los primeros resultados encontrados al buscar librerías para consumir servicios REST en Android fueron Android Asynchronous Http Client y Retrofit.

Android Asynchronous Http Client parece ser utilizada por varias aplicaciones muy conocidas tal como se puede ver en su página web y en el reporte de AppBrain (http://www.appbrain.com/stats/libraries/details/loopj\_asynchttpclient/android-asynchronous-http-client).

Se debe destacar que ambas librerías son muy simples de utilizar, proveen toda la funcionalidad necesaria y además no presentan problemas al consumir una variada selección de servicios REST pero la diferencia principal es que Retrofit no permite por su cuenta realizar requests de manera asincrónica y para ello se debe utilizar la librería RoboSpice que contiene un módulo para integrar con varias de las librerías REST encontradas.

Spring es un framework muy completo, pero también suele ser demasiado “pesado” ya que incorpora mucha funcionalidad que no es necesaria en esta aplicación en particular. Debido a esto descartamos Spring for Android.

La librería RESTDroid se encuentra en un release alpha en busca de colaboradores con lo que no nos provee suficiente confianza de que, llegado el caso de encontrarnos con un problema, encontremos la documentación o ayuda necesaria.

## Conclusiones

Android Asynchronous Http Client se presenta como la mejor alternativa ya que el carácter asincrónico es muy importante para nuestra aplicación ya que se realizan tareas intensivas constantemente y es necesario poder ejecutar en paralelo diferentes acciones. Además, habiendo sido esta librería elegida para importantes desarrollos como los casos de Pinterest, Instagram, MercadoLibre, etc. aporta mayor peso aún al momento de la elección.

# Investigación Hand & Finger Tracking

## Introducción

La aplicación FiubaAR Cliente debe realizar reconocimiento y seguimiento de manos y dedos para así poder también detectar una cierta cantidad de gestos que permitirán al usuario interactuar con los objetos 3D.

## Implementaciones encontradas

**Implementación N° 01**

**Sitio:** <http://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/>

**Implementación N° 02**

**Sitio:** <http://eaglesky.github.io/blog/2014/03/27/color-based-hand-gesture-recognition-on-android/>

Esta implementación parece ser la mejor alternativa: compila y ejecuta. El problema es que es muy lenta, a menos que se la configure con una resolución bien chica, de otro modo no se ve para nada como el video de demo, y hay una demora gigante en el video.

Está basada en la anterior (Implementación N° 01) ya que lo menciona en el post y cita el paper, pero esta portado todo a java y no lo hace con la implementación nativa en JNI. JNI está solo siendo usado para la librería de entrenamiento “SVM”.

Se estima que esta implementación podría ganar velocidad si usara código nativo completamente.

**Implementación N° 03**

**Sitio:** <http://s-ln.in/2013/04/18/hand-tracking-and-gesture-detection-opencv/>

**Implementación N° 04**

**Sitio:** <https://www.dropbox.com/sh/t22mblcu02xf7dt/yNwsZY1aBW>

Desarrollo de características interesantes realizado por un estudiante de Japón.

**Implementación N° 05**

**Sitio:** <https://github.com/browny/hand-tracking>

**Implementación N° 06**

**Sitio:** <https://github.com/iphkwan/high-hand>

**Implementación N° 07**

**Sitio:** <https://github.com/Param-Uttarwar/SimpleHandTracking-openCV>

**Implementación N° 08**

**Sitio:** <https://github.com/royshil/HHParticleFilter> (no compiló)

**Implementación N° 09**

**Sitio:** <https://github.com/felipeue/Hand_tracking>

**Implementación N° 10**

**Sitio:** <https://github.com/minghuam/simple-hand-tracking>

**Implementación N° 11**

**Sitio:** <https://github.com/richhiey1996/Hand-Tracking-and-Segmentation>

**Implementación N° 12**

**Sitio:** <https://github.com/AhmedRiahi/hands-tracking>

## Testing de implementaciones

Todas las implementaciones que se han probado están basadas en OpenCV e implementadas mayormente en C y solo unas pocas en Java para la plataforma Android. Esto es útil, ya que la intención es poder realizar la implementación en Android de forma nativa para poder ganar en performance.

En su mayoría, los proyectos encontrados son aplicaciones de escritorio, con lo cual se intentaron compilar los mismos y ejecutar para poder así validar el funcionamiento de cada una más allá de los videos de demostración que se encontraron para algunos de ellos.

## Conclusiones

De momento, y por las razones expuestas en el análisis de la implementación, se avanzará en el uso y mejora de la implementación N°2.

# Investigación sobre detección de gestos

Como mencionamos, hemos seleccionado dos implementaciones de hand & finger tracking:

1.- <https://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/> (nativa)

2.- <https://eaglesky.github.io/blog/2014/03/27/color-based-hand-gesture-recognition-on-android/> (java)

De ambas podemos obtener información sobre la cantidad y posición de dedos en cada frame de video resultante.

Sin embargo, esto no es suficiente para “detectar gestos” porque para ello necesitamos un continuo de varios frames, a partir de los cuales analizaremos cambios en la posición y cantidad de dedos.

Actualmente el FrameService está reuniendo una determinada cantidad de frames (configurable desde com.fi.uba.ar.services.FrameService.frameGroupSize ) en una cola y recién en ese momento se hace la detección de mano de cada una de ellas de forma individual.

Debemos destacar que, después de realizar pruebas sobre las implementaciones de detección de manos y dedos, se pudo determinar que la utilización de un servicio que procese de forma asincrónica los gestos agrega una demora demasiado grande, lo que impide la consecución de un comportamiento aceptable, ni cercano a tiempo real. La demora (*delay*) que se obtuvo en diferentes dispositivos -con diferentes características de hardware- mostró ser algo inaceptable para la interacción del usuario con la aplicación. Debido a esto, se decide optar por una implementación que realice la detección frame por frame.

Una idea inicial de cómo podríamos detectar gestos es tener una cola de datos detectados para cada frame y mantener esa cola como un historial con una cantidad fija en la cual ingresan datos nuevos y los viejos se van descartando (FIFO) y donde usemos una “ventana” que correspondería a lo que aceptamos como duración de un gesto. De esta ventana tendríamos que ver si hay datos que indiquen el posible inicio de un gesto.

Los gestos a detectar -indicados en nuestra propuesta- serían:

1. Si la presencia de una mano realizando un gesto de pinzado delante de la cámara fuera reconocida, la aplicación cliente aplicará un zoom in/out sobre la animación.
2. Si la presencia de una mano abierta girando delante de la cámara fuera reconocida, la aplicación cliente aplicará una rotación sobre la animación.
3. Si la presencia de una mano realizando un gesto de “cerrar en forma de puño” delante de la cámara fuera reconocida, la aplicación cliente hará desaparecer la animación (concepto de cerrar).
4. Si la presencia de una mano cerrada con sólo el dedo índice extendido delante de la cámara fuera reconocida, la aplicación cliente interpretará la existencia de un “puntero” con el que se podrán realizar “clicks” sobre la animación.
5. Si la presencia de una mano cerrada con dos dedos extendidos (índice y medio) delante de la cámara fuera reconocida, la aplicación cliente interpretará la existencia de un “puntero” que sujeta y puede arrastrar la animación sobre la pantalla.

Según la investigación realizada, existen tesis dedicadas por completo a la detección de gestos, ya que esto es por sí solo un problema bastante complejo de resolver. Ciertas implementaciones proponen utilizar redes neuronales para entrenar y luego reconocer gestos. Este tipo de sistemas superan la complejidad que se le intenta dar al presente proyecto.

En este proyecto, implementaremos una forma simplificada de detectar un gesto que consistirá en el reconocimiento de las siguientes sucesiones de estados (se debe considerar que Vuforia podría entregar aproximadamente 30 FPS –Cuadros por segundo o *Frames Per Second*-, pero asumiremos que existen dispositivos en los cuales la performance dista de ser óptima, ubicándose en promedio en torno a los 20 FPS):

1. INICIO\_GESTO: comprende al menos 2 (y no más de 5) frames consecutivos
2. SECUENCIA\_GESTO: comprende al menos 15 (y no más de 100) frames consecutivos. Aquí se está imponiendo que un gesto no puede tardar más de 5 segundos, aunque podría el usuario decidir mover un objeto alrededor de la pantalla durante dicho tiempo, en cuyo caso luego de 5 segundos se dejará de procesar el gesto y se iniciará la detección nuevamente.
3. FINAL\_GESTO: ídem INICIO\_GESTO, comprende al menos 2 (y no más de 5) frames consecutivos

En cada uno de estos estados se debe encontrar en los frames procesados una cantidad particular de dedos, con ciertas características y posiciones.

Es la intención del equipo, hacer una detección de gestos básica para demostrar el uso que se le puede dar a la aplicación en conjunto con la realidad aumentada dejando claro que la implementación podría ser mejorada o reemplazada en futuras versiones, logrando así mejor o mayor funcionalidad.

A continuación se definen los estados que componen cada uno de los gestos antes descriptos:

1. Pinzado
   1. INICIO\_GESTO: Debe haber solo 2 dedos donde la longitud de uno de ellos debe ser menor (asumimos que sería el pulgar) y además el ángulo entre los dedos debe ser superior a un valor a definir. Ambos dedos deben estar en la misma posición en todos los frames.
   2. SECUENCIA\_GESTO: Debe haber solo 2 dedos (pulgar e índice) en un ángulo diferente al INICIO\_GESTO. Dicho ángulo debe ir cambiando.
   3. FINAL\_GESTO: Se debe detectar una cantidad diferente a 2 dedos.
2. Mano abierta girando

2.1 INICIO\_GESTO: Deben estar presentes (detectados) los 5 dedos (palma abierta completa) en una misma posición durante todos los frames.

2.2 SECUENCIA\_GESTO: deben estar presentes los 5 dedos, pero los dedos 0 y 5 (los extremos) deben haber cambiado de posición.

* 1. FINAL\_GESTO: se debe detectar una cantidad de dedos diferente a 5.

1. Cerrar puño

3.1 INICIO\_GESTO: Deben estar presentes (detectados) los 5 dedos (palma abierta completa).

3.2 SECUENCIA\_GESTO: Deben detectarse entre 1 y 5 dedos.

* 1. FINAL\_GESTO: Se deben detectar 0 dedos

1. Dedo índice click

4.1 INICIO\_GESTO: Debe haber un único dedo (sería conveniente intentar identificar si es el índice, quizás usando datos del círculo o bounding rect o quizás convex hull). En todo caso funcionará igual siempre y cuando haya un único dedo. El dedo debe estar posicionado sobre el objeto 3D.

4.2 SECUENCIA\_GESTO: Se debe continuar detectando un único dedo pero en diferente posición. El gesto de click corresponde a que el dedo debe estar en una posición, luego subir solo en el eje vertical y volver a bajar hasta la posición inicial (aproximadamente).

4.3 FINAL\_GESTO: se deben detectar 0 dedos o al menos más de 1 único dedo.

1. Dos dedos puntero

5.1 INICIO\_GESTO: Deben haber presentes solo dos dedos sobre el objeto 3D, de similar longitud y un ángulo entre ellos pequeño. Esto nos obliga –probablemente- a no poder tener los dos dedos juntos porque el algoritmo no los detectaría como dos, sino como uno único. Esto implica que resulta necesaria una separación entre ambos dedos.

5.2 SECUENCIA\_GESTO: Se debe continuar detectado solo 2 dedos y en diferentes posiciones. Se tomará un punto entre los 2 dedos como posición a la cual se debe mover el objeto 3D.

5.3 FINAL\_GESTO: se deben detectar alguna cantidad diferente a 2 dedos.

## Ideas de Implementación

Android posee formas de detectar gestos del usuario basados en lo que se toque en la pantalla a partir de sus *“touch events”* (eventos de contacto). Con esto ya se puede realizar detección de pinchado por ejemplo.

Existe abundante documentación acerca de cómo utilizar la detección de gestos:

<https://developer.android.com/training/gestures/detector.html>

<https://developer.android.com/training/gestures/scale.html>

<https://developer.android.com/training/gestures/multi.html>

<https://developer.android.com/reference/android/view/GestureDetector.html>

<https://developer.android.com/reference/android/view/MotionEvent.html>

Algo que podría proveernos con la implementación lista de los gestos posiblemente sería la idea de generar *“touch events”* basados en las posiciones donde detectamos los dedos, como si en realidad los lugares donde aparecen los dedos en la cámara fueran puntos “tocados” en la pantalla del dispositivo.

Para ciertos casos esto resolvería la lógica directamente, por ejemplo en el caso de “pinchado” (hacer zoom en un objeto, que en Android se conoce como “scale”). Pero para otros gestos habría que hacer algún tipo de traducción, ya que los clicks (llamados *“tap”* en los gestos de android) no se pueden lograr solo marcando un *touch event* y para enviar un *“tap”* nosotros deberíamos detectar cierto movimiento que consideremos “click”/”tap”.

Para el caso de girar el objeto, nuestro gesto implica una palma completa (5 dedos) girando, pero en el dispositivo girar implica mantener dos dedos continuamente haciendo touch y girando.

Entonces se presenta la situación donde debemos tener algo que funcione como “traductor” de nuestros gestos a gestos de android.

Quizás como caso inicial el tema del pinchado (zoom) podemos hacerlo directamente con touch y ver si funciona directamente.

Otra alternativa sería utilizar Robotium para generar eventos tipo click.

## Links

<https://depts.washington.edu/madlab/proj/dollar/index.html>

<https://github.com/asweigart/moosegesture>

<https://stackoverflow.com/questions/5867059/android-how-to-create-a-motionevent>

<https://stackoverflow.com/questions/4396059/how-to-simulate-a-touch-event-in-android>

<https://developer.android.com/training/gestures/scale.html>

<https://stackoverflow.com/questions/5250290/why-doesnt-this-motionevent-simulation-work>

<https://stackoverflow.com/questions/5240287/how-to-send-synthesized-motionevent-through-the-system>

[https://groups.google.com/forum/#!searchin/android-platform/simulate$20motionevent|sort:relevance/android-platform/e1mL3YT2uU8/LdkCesv4rxUJ](https://groups.google.com/forum/#!searchin/android-platform/simulate$20motionevent%7Csort:relevance/android-platform/e1mL3YT2uU8/LdkCesv4rxUJ)

<https://github.com/RobotiumTech/robotium/blob/b69dcf740baabec48c91999e523377faef79682e/robotium-solo/src/main/java/com/robotium/solo/Solo.java>

<https://www.softteco.com/blog/touch-hold-swipe-release-gesture-simulation-in-android-unit-test/>

<http://alvinalexander.com/java/jwarehouse/android/test-runner/src/android/test/TouchUtils.java.shtml>

<http://blogs.steeplesoft.com/posts/2013/simulating-swipes-in-your-android-tests.html>

<https://nandhanthiravia.blogspot.com.ar/2014/01/inject-multi-touch-event.html>

<http://www.pocketmagic.net/injecting-events-programatically-on-android/>

<http://www.vogella.com/tutorials/AndroidTouch/article.html>  < artículo destacado.

<https://stackoverflow.com/questions/29573026/dispatchtouchevent-in-fragment-in-android>

<https://github.com/Almeros/android-gesture-detectors>

<https://github.com/pwittchen/gesture>

<https://github.com/nisrulz/sensey>

<https://github.com/codepath/android_guides/wiki/Gestures-and-Touch-Events>

<https://medium.com/@ali.muzaffar/android-detecting-a-pinch-gesture-64a0a0ed4b41>

<https://blahti.wordpress.com/2011/01/17/moving-views-part-2/>

<https://stackoverflow.com/questions/11523423/how-to-generate-zoom-pinch-gesture-for-testing-for-android>

<https://stackoverflow.com/questions/3637044/generating-multitouch-motionevents-for-testing>

<https://stackoverflow.com/questions/39582808/android-motionevent-pointer-index-confusion>

# Burndown Charts por Sprint

## Sprint 1

## Sprint 2

## 

## Sprint 3

## Sprint 4

## 

## Sprint 5

## Sprint 6

# Calendario

## Replanificación N°1 - Al inicio del Sprint N°3, 20/Octubre/2014.

Se re planifica de acuerdo con la performance mostrada por el equipo hasta este punto del proyecto. La re planificación da lugar a los siguientes cambios:

Los sprints cambian su duración de las cuatro semanas estimadas inicialmente a seis semanas. Esto se debe a que se verifica la necesidad de incrementar las horas de desarrollo y testing, y a que el tiempo semanal disponible para aplicar al proyecto resulta menor al estimado inicialmente.

Se incrementa la cantidad de horas de desarrollo en un 25% promedio.

Se incrementa la cantidad de horas utilizadas en testing en un 100% promedio.

Se detalla que parte del desarrollo en el sprint 3 contiene perfeccionamiento de la lectura del marker.

Como consecuencia de lo anterior, la HH totales pasan de 866 a 1057.

Debido a complicaciones de los integrantes del equipo, el inicio del sprint 3 se posterga. Se da inicio al mismo en fecha 15/Diciembre/2014.

## Replanificación N°2 - Al inicio del Sprint N°4, 26/Enero/2015.

Nuevamente, por complicaciones personales de los integrantes del equipo se posterga el inicio del sprint 4 -cuyo inicio debería tener lugar el 26/Enero/2015- hasta el 3/Agosto/2015. No se toma ninguna acción adicional.

## Replanificación N°3 - Al inicio del Sprint N°5, 26/Enero/2015.

Nuevamente, el trabajo sufre una postergación, lo que da lugar a esta nueva replanificación, la cual establece el inicio del sprint N°5 en fecha 26/Septiembre/2016, en lugar del 14/Septiembre/2015. Se contemplan dos semanas de receso entre el 19/Diciembre y 1/Enero. No se toma ninguna acción adicional.

## **Sprint N°1** (137 HH) Inicio: 04/08/2014 Finalización: 07/09/2014

Documentación: Documento de set up del ambiente de desarrollo. [8 HH][AA]

Investigación: Integración de OpenCV en Aplicaciones Android. [32 HH][EG]

Investigación: Diseño y definición del marker con justificación de su elección. [18HH][AA]

Diseño: Diseño de aplicación básica. [18 HH][EG]

Documentación: Documento de diseño aplicación básica con UML. [6 HH][AA]

Desarrollo: Armado del ambiente de desarrollo. Aplicación cliente (v0.1) básica funcional y estable para el uso de la cámara con integración con OpenCV. [50 HH][AA+EG]

Testing: Aplicación cliente en su versión actual. [4 HH][AA]

Estado del Sprint N°1: Cumplido [T.Estimado: 4 semanas, T.Real: 5 semanas]

## **Sprint N°2** (144HH) Inicio: 08/09/2014 Finalización: 19/10/2014

Documentación: actualización de documentos de diseño. [18HH][EG]

Desarrollo: Aplicación cliente (v0.2) con reconocimiento y tracking del marker, superando la dificultad de mantener la efectividad de dichas acciones aún cuando se produzcan rotaciones y/o ligeros cambios en ángulo de la cámara respecto al marker. [100HH][AA+EG]

Investigación: Diseño y definición de un código de identificación adecuado, con detección y corrección de errores. [18HH][AA]

Testing: Aplicación cliente en su versión actual. [8HH][AA]

Estado del Sprint N°2: Cumplido [T.Estimado: 4 semanas, T.Real: 6 semanas]

## **Sprint N°3** (156HH) Inicio: 15/12/2014 Finalización: 25/01/2015

Documentación: actualización de documentos de diseño. [4HH][EG]

Desarrollo: Aplicación servidor (v0.1). [45HH][EG]

Desarrollo: Aplicación cliente (v0.3) con lectura del código de identificación de cliente y comunicación básica con aplicación servidor. Refinamiento de la lectura del marker (Código QR o equivalente) [75HH][AA]

Investigación: OpenGL y motores 3D. [20HH][EG]

Documentación: actualización de documentos de set-up de ambiente (por el motor 3D). [4HH][EG]

Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA+EG]

Estado del Sprint N°3: Cumplido [T.Estimado: 6 semanas, T.Real: 6 semanas]

## **Sprint N°4** (152HH) Inicio: 03/08/2015 Finalización: 13/09/2015

Documentación: Actualización de documentos de diseño. [4HH][AA]

Investigación: Algoritmos e implementaciones para hand & finger tracking usando OpenCV. [40HH][AA]

Desarrollo: Aplicación servidor (v0.2) Comunicación completa. El cliente lee el código y lo transmite al servidor, éste identifica al cliente y selecciona la información a representar en la aplicación cliente, y por último envía datos de overlay y animación a la aplicación cliente. [50HH][EG]

Desarrollo: Aplicación cliente (v0.4) Comunicación completa con servidor y animación básica con motor 3D elegido. Nota: la animación está fuera del alcance de la aplicación, solo se pretende mostrar que la selecciona y descarga correctamente del servidor, se la ubica adecuadamente en la escena y que la interacción con el usuario -en futuros sprints- funciona adecuadamente. [50HH][AA+EG]

Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA+EG]

Estado del Sprint N°4: Cumplido [T.Estimado: 6 semanas, T.Real: 6 semanas]

## **Sprint N°5** (163HH) Inicio: 26/09/2016 Finalización: 06/11/2016

Desarrollo: Aplicación servidor v1.0. Versión final, solo un refinamiento de la versión del sprint anterior con bugs corregidos. [24HH][AA]

Desarrollo: Aplicación cliente (v.0.5). Versión con hand and finger tracking inicial. [125HH][AA+EG]

Documentación: Actualización de documentos. [6HH][EG]

Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA+EG]

Estado del Sprint N°5: Cumplido [T.Estimado: 6 semanas, T.Real: 6 semanas]

## **Sprint N°6** (151HH) Inicio: 07/11/2016 Finalización: 18/12/2016

Desarrollo: Aplicación cliente (v.0.6). Versión con hand and finger tracking, aún básico pero integrado ya en las funciones de interacción con la animación. [125HH][AA+EG]

Documentación: Actualización de documentos. [4HH][AA]

Documentación: Creación del manual de usuario. [14HH][EG]

Testing: Aplicaciones cliente y servidor en su versión actual. [8HH][AA]

Estado del Sprint N°6: Cumplido [T.Estimado: 6 semanas, T.Real: 6 semanas]

## **Sprint N°7** (154HH) Inicio: 02/01/2017 Finalización: 12/02/2017

Desarrollo: Aplicación cliente (v.1.0). Versión con hand and finger tracking, totalmente integrado. [100HH][AA+EG]

Testing: Aplicaciones cliente y servidor en su versión actual (final). [36HH][AA+EG]

Documentación: Se crean videos de ejemplo de uso. [12HH][AA]

Demo: Se hace disponible un demo en Google Play. [6HH][EG]

Estado del Sprint N°7: Pendiente [T.Estimado: 6 semanas, T.Real: x semanas]

Cantidad Total de Horas Hombre: 1057