FINAL PROJECT

REPORT

ZATDROID

Satellite Tracking and Augmented Reality App for ANDROID

1. ABSTRACT

The purpose of ZATDROID is to offer the user the possibility of tracking any artificial satellite. It locates it in Google Maps together with its predicted trajectory and also it allows the user to see it in real time in the sky with augmented reality camera view. The application is implemented in Java for ANDROID devices (tablets or smartphones).

Orbital mechanics calculations are developed following Newton equations and NORAD (*North American Aerospace Defence Command)* propagation models, firstly published in 1980 *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

ZATDROID downloads information from a data base of satellites provided by CELESTRAK.COM, does the orbital mechanics calculations (*NORAD models*), gets the device sensors magnitudes, connects to the web service `Google Maps Elevation` to get the altitude of the user location, processes data in terms of XML language, creates a GOOGLE MAPS views with the updated position in real time of the satellite picked and shows the position in the sky in augmented reality using OPENGL for the camera view.

The GUI (*Graphical User Interface*) manages to lead the users through an easy and friendly navigation to achieve their aims quickly, showing the results of the user picks with *BreadCrumbs*, supporting English and Spanish, showing icon-based menus and keeping the users informed of the longer processes by "*progress bars*".

ABSTRACT

El objectivo de ZATDROID es ofrecer la posibilidad al usuario de encontrar satélites artificiales. Posiciona en GOOGLE MAPS junto con su trayectoria y permite al usuario verlo en tiempo real en el cielo con la vista de la cámara en realidad aumentada.

Los cálculos de mecánica orbital se han desarrollado siguiendo las ecuaciones de Newton y los modelos de propagación de NORAD (*North American Aerospace Defence Command)*, publicados por primera vez en 1980 *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

ZATDROID descarga información de una base de datos de satélites ofrecida por CELESTRAK.COM, realiza los cálculos de mecánica orbital (modelos NORAD), obtiene las magnitudes de los sensores del dispositivo, se conecta al servicio web `Google Maps Elevation` para obtener la altitud de la localización del usuario, procesa datos en lenguaje XML, crea una vista de GOOGLE MAPS con la posición del satélite elegido actualizada en tiempo real y muestra la localización en el cielo en realidad aumentada usando OPENGL para la vista de cámara.

La GUI (*Interfaz gráfica de usuario*) gestiona la experiencia del usuario a través de una fácil y amigable interfaz de navegación para alcanzar los objetivos rápidamente, mostrando a los usuarios sus elecciones con un menú *BreadCrumbs*, ofreciendo todo en inglés y español, con menús basados en iconos y manteniendo a los usuarios informados de procesos largos con barras de proceso.

2. ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisors Fernando and Jesús for the continuous support of my project, for his patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me all the time.

Besides my advisors, I would like to thank my friend Rafael for all his expert advices, insightful comments and user experience knowledge.

Thanks also to all those anonymous people writing full and documented answers on the internet, without their daily help, I could not have achieved this Project.

My sincere thanks to "*travis*" from `mybringback.com`, author of the fantastic ANDROID video tutorials that introduced me into the Apps world, Maciej Grzegorczy, author of the SATFINDER app and T. S. Kelso webmaster of the fabulous `celestrak.com` for their generosity in spreading knowledge around the world.

Last but not the least, I would like to thank my family for supporting me spiritually throughout my life and being an example of effort and perseverance, especially Julia, who has suffered from lack of attention during the time dedicated to this project.

I wish this Project could help my future kid explore the world of physics and technology and wake up the sense of astonishment and surprise in Nature.

3. TABLE OF CONTENTS

4. LIST OF FIGURES

5. INTRODUCTION

During the years I have been studying Computer Science, it has been always on my mind the idea of joining both of my study worlds: Space Engineering and Computers.

At the moment I had to decide the purpose of my final work project, I realized that it was the time to finally develop my initial idea. After a lot of work thinking and asking friends that are working in computer science and space, I came up with this project, ZATDROID. It includes skills from both sides of my career, keeping me motived and offering me the possibility to learn topics that I have never worked.

Moreover, I found that it could be a chance for many people interested in tracking artificial satellites from their devices: tablets and smartphones. Also this App brings closer to the non-expert user the satellites we are using every day whose identity is unknown for us.

The code has been developed in JAVA for ANDROID, using Eclipse with ANDROID SDK and has been tested and designed for a *Sony Xperia Neo V* with ANDROID 2.3. It has been tested in other devices with ANDROID 4 and works perfectly.

5.1. IDENTIFICATION

Título: ZATDROID: Satellite Tracking and Augmented Reality App for ANDROID
Autor: Rodrigo Santos Álvarez
Director: Fernando Díaz Gómez / Jesús Álvarez Gómez
Departamento: Computer Science (ATC, CCIA, LSI)

5.2. CD STRUTURE

CD:

- Documentation
  - Final_Year_Project_Report.pdf
  - Technical_Manual.pdf
  - User_Manual.pdf
- Software
  - Source Code
  - ZATDROID.APK
- Solapa.pdf
- Titulo.pdf
- Resumen.pdf

5.3. OBJECTIVES

1) To develop an ANDROID App, easy and friendly
2) To bring closer to the user artificial satellites orbiting the Earth
3) To be able to run the orbital mechanics calculations that predicts the position of a satellite.
4) To show the real time position of the satellite over GOOGLE MAPS, as well as its trajectory
5) To track the present position over the observer´s sky, using Augmented Reality.
6) To provide the app in English and Spanish.
7) To support different devices (models, trademarks, sizes)
8) To gain skills in ANDROID Software development and XML language.

5.4. STATE OF THE ART

On the one hand, the previous work consisting on searching information about existing applications, programs or studies has been important when starting to set the objectives and methodology. After some emails and internet search, this is a list of the applications that put together some of the functionalities that ZATDROID wants to implement:

- GPREDICT [1]
- VIS SAT [2]
- SATFINDER [3]
- SATELLITE AR [5]

On the other hand, the previous training in skills needed for the Project were to be considered. Most of the investment work was about the orbital mechanics part, which seemed to be much more specific tan computer science in ANDROID application development. Information about satellites georeference, device georeference, orbit prediction and methodology to solve mechanics equations has been difficult to find and understand. The main source is the document from NORAD (*North American Aerospace Defence Command*) which develops Perturbation models in orbits propagation *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

5.5. GENERAL DESCRIPTION

5.5.1. FUNCTIONAL REQUIREMENTS

INPUTS:

The starting point once you have decided to create an application giving information concerning satellites, is retrieving the sat description and main parameters of the orbit from a reliable and updatable source. Although there are sites where all magnitudes needed can be found (latitude, longitude…) directly just by developing a search inside the page, the objective of this project is more ambitious and the calculations are going to be done within the code. After searching the web for some time, CELESTRAK.COM finally fitted the requirements.

CELESTRAK.COM was created and is maintained by Dr. T.S. Kelso, recognized worldwide as an expert in the area of satellite tracking and orbit determination. Regularly sought out for advice and counsel by NASA, European Space Agency (ESA), Russian Space Institute, US and AF Space Command, and many universities and commercial space organizations.

The satellites data, together with their orbital parameters are to be found in a txt file called "TLEs (*two line element sets*)" from NORAD (*North American Aerospace Defence Command*) whose description is shown in appendix 1. Basically, it gives the orbital parameters in two lines with a known format. These parameters are measured in one moment in time every day, so that ZATDROID has an initial point to calculate the propagation for the orbit.

DATA MANAGEMENT:

XML:

Firstly, the txt file, in which TLEs are provided, are parsed into XML, following the example of appendix 2. It is also created a XSD (*XML Scheme Definition*) scheme.

Once the XML is ready, a search can be launched using SAX (*Simple API for XML*) in ANDROID to find the satellite picked and select all the data included in the XML file. Then a java object (sat class) is created with the information of the satellite picked.

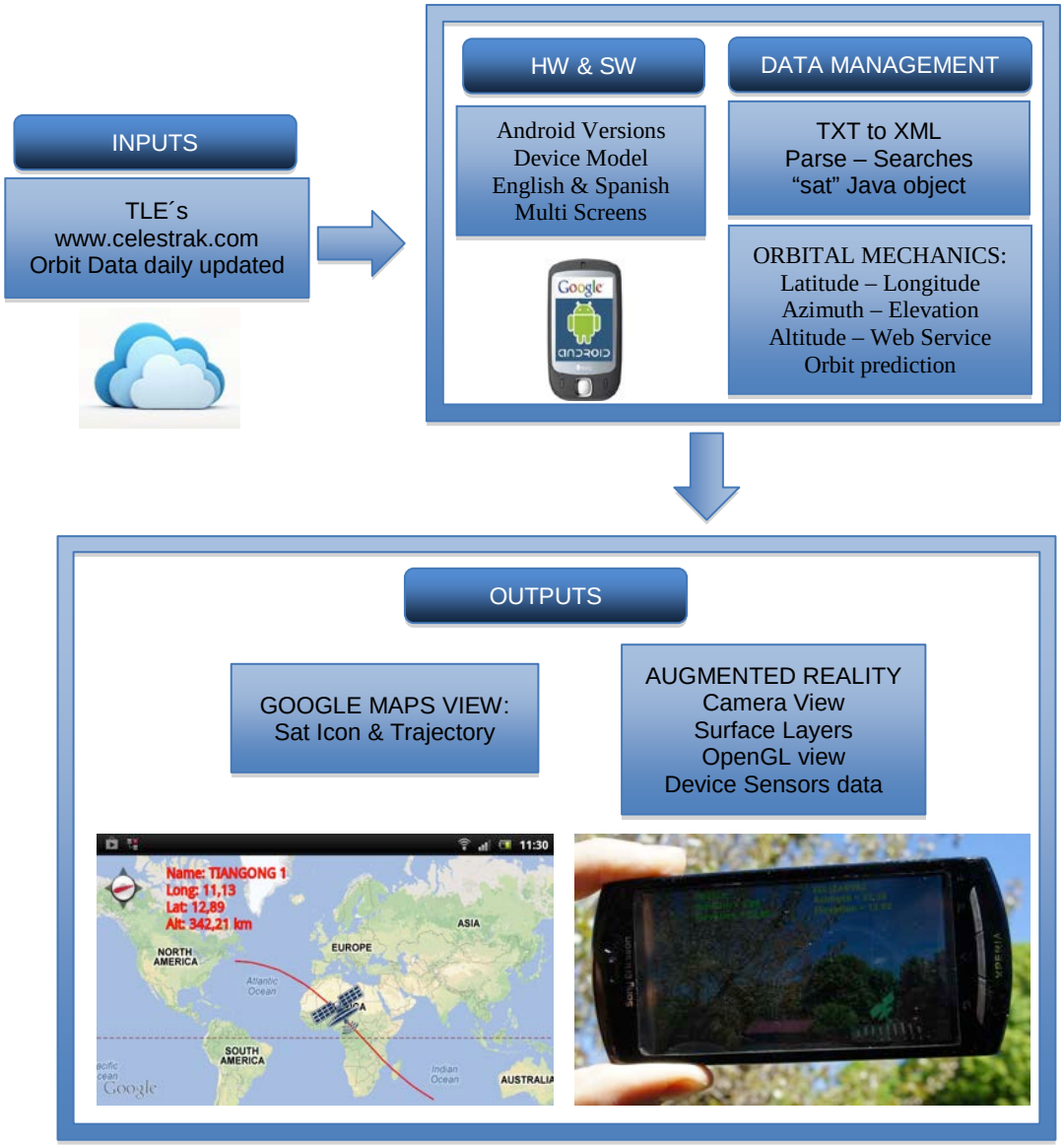


Figure 1. Description of the data flow and how the system manages to get the inputs, work with the data and export the two main outputs.

ORBITAL MECHANICS:

Longitude, latitude and altitude are magnitudes needed for the `Google Maps` positioning. Also azimuth and elevation of the satellite are essential for the augmented reality functionality.

The method to calculate these magnitudes was first introduced in 1980, *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4] where SGP (*Simplified general propagation model*) and SGP4 / SPD4 (*SGP version 4 for low and deep space orbits*) were implemented. These models develop the Newton´s Gravitational equations and some more accuracy modifications that takes into account perturbations. It was written in `FORTRAN`.

Some new developments have been achieved to improve the code using `C` language [6]. Nevertheless, the heart of the code remains the same. ZATDROID takes advantage of this implementation and develops all the calculations in Java for ANDROID. The orbital mechanics equations are explained in detail later on. ZatDroid implements SGP4/SDP4 NORAD prediction models, based on [4] and with some improvements of [6]

Furthermore, longitude, latitude and altitude of the users location is required for the augmented reality feature. GPS sensor provides these magnitudes. In case GPS is not available, network positioning is used for latitude and longitude and for the altitude, `Google Maps Elevation` web service is asked, though `USGS Elevation Query` is also another possibility.

OUTPUTS:

GOOGLE MAPS VIEW:

Once the application has all the parameters of the orbit and information of the satellite, it can be located inside a map view. The user is shown the latitude and longitude of the sat updated every second together with the name. In another layer, an icon is representing the satellite updated every second and finally the predicted trajectory for the next minutes.

AUGMENTED REALITY:

Taking advantage of a camera view, some layers are added to provide information concerning the position of the satellite in the sky where the user is. An icon must be drawn when the camera device is pointing at the azimuth and elevation of the satellite. Both azimuth and elevation of the satellite and the camera point of view need to known together with geographical coordinates (latitude and longitude) of the device.

Azimuth and elevation of the camera, that is to say, where is the camera pointing at, is retrieved from the accelerometer sensor. After some matrix transformations of the vectors from the coordinate reference systems of the device and the Earth, using OPENGL to set the camera view properly taking into account both directions of the satellite and the device, all the data is ready to draw the completed view.

While the user is rotating the device to find the satellite in the sky, information is being shown concerning the satellite and device parameters onto another layer. Also a line indicates the direction in which the user can find the satellite and finally the icon appear when the camera points at the right direction in the sky. The program gives information in case the satellite is not visible (under the horizon) and the time lasting until next overhead pass.

ZATDROID has the chance to implement an option that retrieves a list of satellites that are visible (over the horizon) at the moment where the user is located. It is not useful though, since it lasts some minutes to do all the calculations. For every satellite, SPG4/SPD4 model must be run, so the calculus time really increases.

HARDWARE AND SOFTWARE:

ZATDROID is required to:

- Run under ANDROID versions from 2.3 to 4.3 (*newest version up to dat*e).
- Support Multilanguage (*English and Spanish*)
- Support multiscreen sizes
- Support different smartphones models
- Tests have been carried out successfully with *Sony Xperia Neo V, Samsung Note, Samsung Galaxy S4, Samsung Tablet 11', Nexus 4, Samsung S3-Mini*

5.5.2. SCREENS SEQUENCE DIAGRAM

A more complete explanation of the diagram can be found in other technical documents. The GUI (*Graphical User Interface*) is based on icons with explanatory text, taking advantage of ANDROID multilingual support: English and Spanish, depending on the language of the device and making the user experience unique and easy.

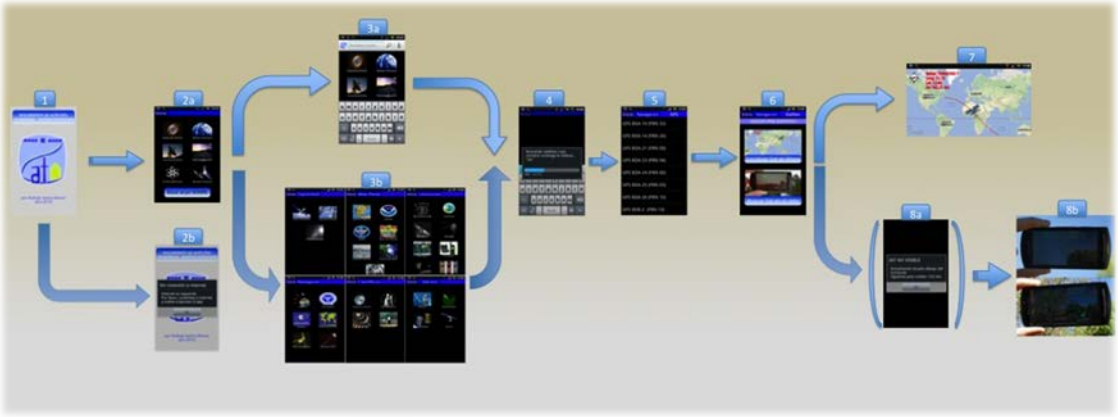


Figure 2. Explanation of the sequence of the screens through the application.

- **1-2b:** Firstly, the app checks if the device has internet connexion available. If not, a message pops up not to let the app execute. Internet is necessary.
- **2a**: Secondly, the user can choose between searching a satellite by its type or typing key words.
- **3a – 3b - 4**: Thirdly, after a progress bar to keep the user informed or some screens to decide the type, one specific satellite is finally picked. During this process, at the top of the screen the application implements *Bread Crumbs*. It allows the user to navigate through the types chosen.
- **6**: Once the satellite is picked, the screen shows the two functionalities of the application: `Google Maps` and Augmented reality with self-explanatory photos.
- **7**: For the `Google Maps` screen, it consists of the map view with two layers, indicating the latitude, longitude and name updated every second and the icon in the right placed also updated.
- **8a – 8b:** For the Augmented reality screen, in case the satellite is located under the horizon a message pops up with the time lasting until next overhead pass. Then the camera view is shown with some layers for the data of the satellite and data of the device updated every second, an arrow indicating the direction to follow in order to find the sat in the sky and the satellite icon when the device points at the right direction.

6. ORBITAL MECHANICS

This project makes sense since it involves a lot of knowledge in orbital mechanics. This section aims to be an introduction of the complex calculations used, as well as some general concepts concerning satellites. For a detailed description of the equations, processes and calculations, the technical manual is to be referenced.

6.1. COORDINATE SYSTEMS

Orbital mechanics need some coordinate systems to develop all the calculations. Only a picture and the name is given below.

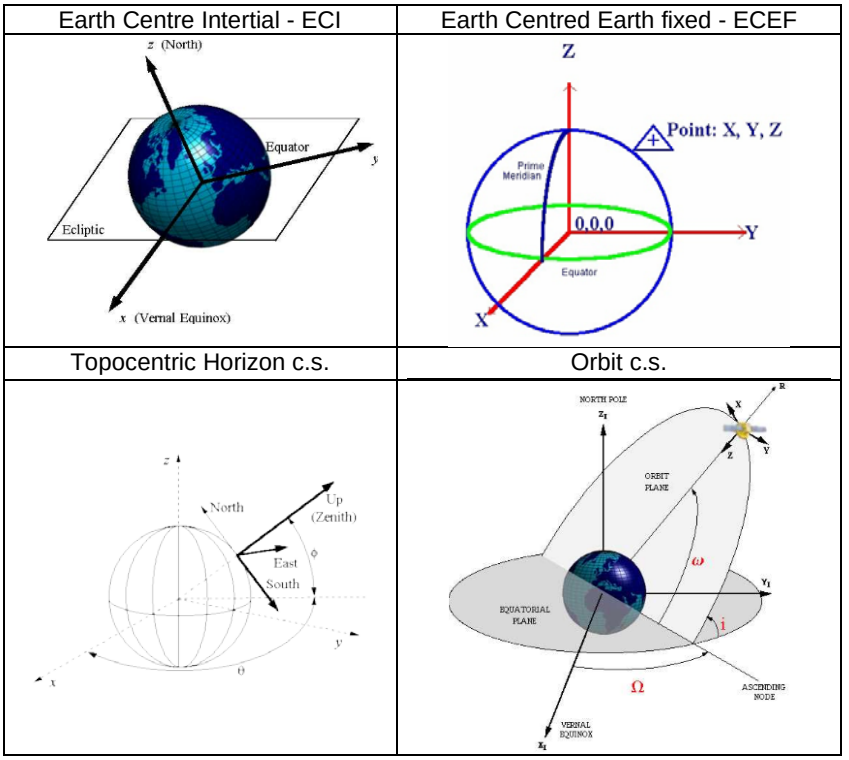


Figure 3. Orbital mechanics coordinate systems

An important coordinate system must be taken into account when working with devices such as smartphones or tables. Sensors (gravity, accelerometer and magnetic) give the magnitudes in terms of vectors, and every vector must be represented into a coordinate system, which is also used by OPENGL when working with 3D geometry or moving the camera in a camera view. The rotation matrix helps interpreting the vectors and their transformations.

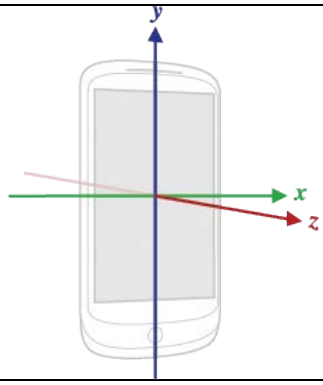
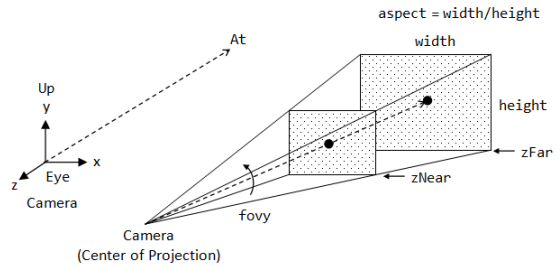
Sensors API c.s.	OPENGL Camera positioning
	

Figure 4. Device coordinate systems

6.2. NEWTON´S AND KEPLER´S LAWS

It was Kepler who at the beginning of the 17th century stated the 3 laws that describes the movement of the planets around the Sun. Kepler took advantage of the experimental work and measurements made by Tycho Brahe to realize these laws that can also be applied to the movement of any object in space around a more massive object:

- The orbit of every planet is an ellipse with the Sun at one of the two foci.
- A line joining a planet and the Sun sweeps out equal areas during equal intervals of time
- The square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit.

Later, Newton developed the fundamental laws of physics upon which the theory of orbital mechanics is based.

- **Newton's law of universal gravitation**, self explained by the well-known formula

$$\vec{F} = -\frac{GMm}{r^2}\frac{\vec{r}}{r} \qquad [7.1]$$

G = 6.67·10−11 Nm^2/kg^2 is the universal gravitational constant
The vector $\vec{r}$ goes from M to m and the force is on m.

- Newton's law of motion:

$$\vec{F} = m \cdot \ddot{\vec{r}}\frac{\vec{r}}{r} \qquad [7.2]$$

Assuming that:

- The mass of a satellite is $m << M$ and $G(M+m) \approx GM$
- The gravitational forces are the only forces acting on the two bodies and these two bodies are not under influence of any other gravitational force from any other body.

And making use of Newton´s laws and some mathematical techniques, the three Kepler laws are proven in the next paragraphs:

1st LAW:
Solving the "*two bodies problem*" in polar coordinates and realizing the movement in such a central force is plain (the plane is determined by the position vector $\vec{r}$ and the velocity $\dot{\vec{r}}$) we can achieve the final formula that defines the orbits:

$$\vec{r} = \frac{\vec{L}_0^2 / GM}{1 + \frac{A}{GM}\cos(\nu)} = \frac{p}{1 + e\cos(\nu)} \quad [7.3]$$

$$\cos(\nu) = x \cdot \vec{r} / r \quad [7.4]$$

Depending of the value of e (eccentricity) the orbits will be:

- Ellipses: $0 < e < 1$
- Parabolas: $e = 1$
- Hyperbolas: $e > 1$

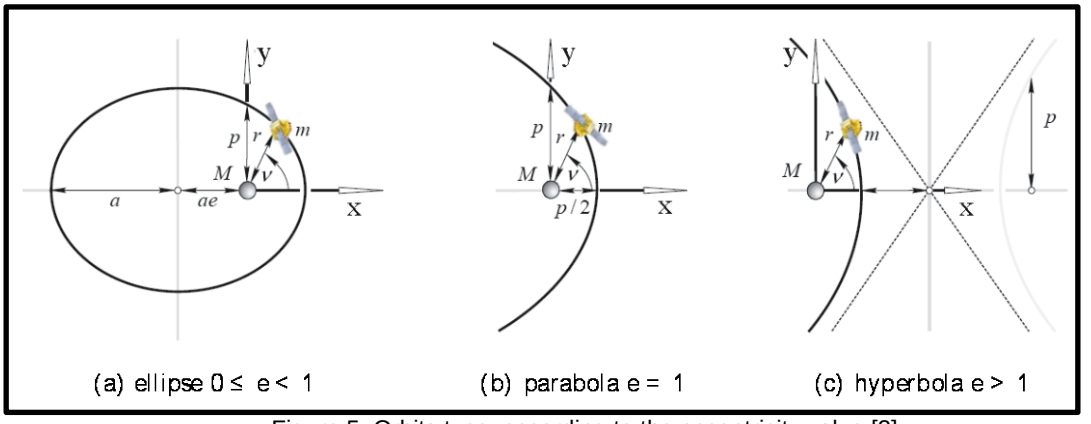


Figure 5. Orbits type, according to the eccentricity value [2]

2nd LAW:
Taking into consideration that the angular momentum of the motion is conserved since the gravitational force is a central force:

$$\vec{L} = \vec{r} \times \left(m \cdot \dot{\vec{r}}\right) = m\left(\vec{r} \times \dot{\vec{r}}\right) \quad [7.5]$$

$$\frac{\partial}{\partial t}\vec{L} = 0 \Rightarrow \vec{L} = \vec{L}_0 \quad [7.6]$$

$$\frac{dA}{dt} = \frac{1}{2}L_0 \quad [7.7]$$

It is proven that "A line joining a planet and the Sun sweeps out equal areas during equal intervals of time", meaning dA/dt the area swept out.

3rd LAW:

$$\frac{dA}{dt} = \frac{A}{T} = \frac{\pi a^2 \sqrt{1-e^2}}{T} \quad [7.8]$$

$$\frac{dA}{dt} = \frac{1}{2} L_0 = \frac{1}{2} \sqrt{GMa\left(1-e^2\right)} \quad [7.9]$$

$$T^2 = \frac{4\pi^2}{GM} a^3 \quad [7.10]$$

a: semimajor axis of the orbit.

Final result for T (period) equation comes from [7.8] and [7.9] calculating the same magnitude through two different ways.

6.3. KEPLERIAN ELEMENTS

With all the data retrieved from Newton´s laws it can be stated for ellipse orbits (the most common ones for satellites) that there are six constants that describes easier the orbit than using the equations. These so called keplerian elements are not constant if disturbance forces are considered and all of them become time-variant. (*see next section*)

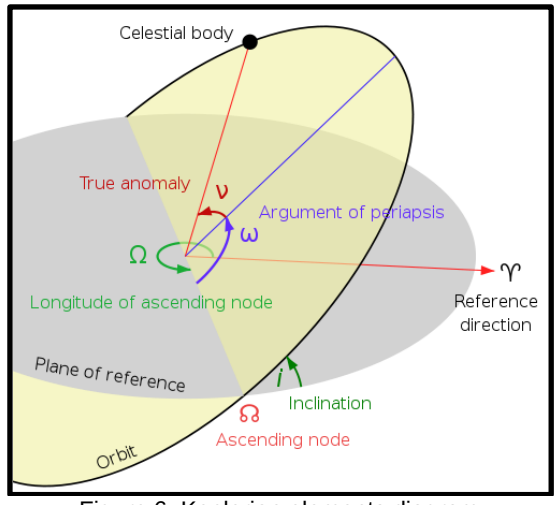


Figure 6. Keplerian elements diagram

- Shape and size of the ellipse.
  1) **Eccentricity (*e*):** already explained.
  2) **Semimajor axis: (*a*):** the sum of the periapsis and apoapsis distances divided by two.
- Orientation of the orbital plane in which the ellipse is embedded:
  3) **Inclination (*i*)** vertical tilt of the ellipse with respect to the reference plane, measured at the ascending node
  4) **Longitude of the ascending node (☊):** horizontally orients the ascending node of the ellipse with respect to the reference frame's vernal point

- Finally:
  5) **Argument of periapsis (ω):** defines the orientation of the ellipse in the orbital plane, as an angle measured from the ascending node to the periapsis.
  6) **Mean anomaly at epoch (M_0):** defines the position of the orbiting body along the ellipse at a specific time (the "*epoch*").

6.4. NORAD ORBIT PROPAGATION MODELS [2]

Disturbance forces can be minute in comparison to the main gravitational force. Nevertheless, if we are to calculate a more accurate prediction, these forces must be taken into account.

The North American Aerospace Defence Command (NORAD) developed some Perturbation models in *SpaceTrack* 3 [4], depending on its accuracy:

- **SGP** (*Simplified General Perturbation model*):
  - Geopotential disturbances: the Earth is not spherical. It has a bulge at the equator and is flattened at the poles.
  - Perturbances due to Sun and the Moon: more tan only one object in space causes perturbations.
- **SGP4** (*Simplified General Perturbation model version 4*) adding:
  - Atmospheric drag: removes energy from the satellite.
  - Solar radiation: less important in lower orbits.
- **SDP4** (*Simplified General Deep Space Perturbation model version 4*)
  - It is based on SGP4 but for orbits with a period longer than 225 minutes, where additional deep-space perturbations have to be considered.

ZatDroid implements SGP4/SDP4 perturbation models described in [4] and with some improvements of [6]. Code from C, FORTRAN or PASCAL has been migrated to JAVA for ANDROID

NORAD maintains a set of elements needed to do the calculations for the propagation of the orbits. These elements are refined every day so that the model keeps its accuracy and can be found as txt files called TLEs (*two line element sets).* CELESTRAK.COM is the source from ZATDROID downloads these elements every time the user executes the app.

6.5. SATELLITES ORBITS

Finally, a brief description of the orbits is explained, as it is written in [8]

SYNCHRONOUS:

EARTH SYNC.

Also known as Geostationary. It allows the satellite to look always at the same point of the Earth. Therefore, it is highly demanded and there is a belt with a lot of satellites. Its altitude is roughly 35.876Km, flying at 3Km/s over the equator, inclination 0.

In ZATDROID these satellites can be recognized easily as their latitude and longitude are constant and no trajectory line is drawn in the `Google Maps` view.

Figure 7. Geostationary belt

SUN SYNC:
These orbits allows a satellite to pass a section of the Earth at the same time of the day. The surface illumination angle will be nearly the same every time. This consistent lighting is a useful characteristic for satellites that image the Earth's surface in visible or infrared wavelengths (e.g. weather and spy satellites) and for other remote sensing satellites (e.g. those carrying ocean and atmospheric remote sensing instruments that require sunlight)

POLAR:
The satellite passes over the planet's poles on each revolution. The inclination of these orbits therefore is $i \approx 90º$. These orbits are the only ones that allows to pass the poles to search any information.

7. BREAKTHROUGH DESIGN FEATURES

7.1. MULTILANGUAGE SUPPORT

Following ANDROID developers advices, all strings used in ZATDROID have been encoded inside the XML file `strings.xml` in English, which is the main language. This file is located in *res/values* There is also a file `string-es.xml` in *res/values-es* in which all strings are duplicated Spanish.

Whenever the user runs ZATDROID, the language of the strings are shown in the language of the device automatically, as long as it is English or Spanish. English is by default.

Figure 8. Multilanguage screen

7.2. MULTIPLE SCREENS SUPPORT: ICONS, TEXT

Bearing in mind the wide range of devices it is mandatory to support different screen sizes. Although in the ANDROID help it is recommended the use of "*dp*" (*density pixels*) when defining pictures or text size in XML layouts, it has not been the solution this time.

Many tests have been run with different devices (*Sony Xperia Neo V*, *Samsung Note*, *Samsung Galaxy S4*, *Samsung Tablet 11'*, *Nexus 4*, *Samsung S3Mini*) and using "*dp*" did not show the icons and text the same way in all devices. So another definition of the layouts was necessary.

Finally, it was found out that using only xml layouts was not the right procedure. Instead, layouts definition inside Java code was selected. This option lets the programmer customize deeper the layout configuration. `addView` adds views (layouts) to other layouts.

- On the one hand, icons needs to set height and width according to the size of the device. This is achieved by retrieving the device width and height with this piece of code:
```
final float height=getResources().getDisplayMetrics().heightPixels
final float width=getResources().getDisplayMetrics().widthPixels
```
Every icon height and width can be configured as a certain percentage of total:
```
int h = (int) (0.05 * height); // 5%
int w = (int) (0.20 * width);  // 20%
```
- On the other hand, text size (`setTextSize`) and gap between letters (`setTextScaleX`) is something more complex. The text must adjust to the size of its box perfectly.

Gap between letters:

A new function has been created `scaleButtonText`.

- A `Paint` object is created with 100% height and setTextScaleX = 1
- Ask the `paint` for the bounding rectangle if it were to draw this text.
- Determine the width
- Calculate the new scale in x direction to fit the text to the button width

Text Size:

It is set to a percentage of the total height of the box.

7.3. ASYNCTASK AND PROGRESS BAR: RUNNING CODE IN BACKGROUND.

There are two events within the application when a connection through internet is established to download files. This happens when connecting to CELESTRAK.COM and downloading TLEs in a txt file. Such a process may last some seconds and give errors. It is also the moment when both XML and XSD file is created from txt.

`Asynctask` is defined in the ANDROID developers site:
"This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.
An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called `Params`*,* `Progress` *and* `Result`*, and 4 steps, called* `onPreExecute`*,* `doInBackground`*,* `onProgressUpdate` *and* `onPostExecute`"

So this every time ZATDROID downloads a file from the internet, `asynctask` is launched and a progress bar is shown to the user to keep informed of the task that are being performed.

7.4. SHARING AND STORING INFORMATION

ZATDROID requires to share information: files, java objects, parameters. There are many ways to share information in ANDROID.

SQLITE DATABASES:

In case a complex data management is needed with all the advantages of a relational data base. ZATDROID makes no use of this option because there is no need to manage a lot of information with tables and relationships. Also, XML files has been picked as the way to manage data because of its simplicity and the objective to learn concerning XML during this project.

PASSING OBJECTS: [9]

- SERIALIZABLE Class:
  The problem with this approach is that reflection is used and it is a slow process. This mechanism also tends to create a lot of temporary objects and cause quite a bit of garbage collection.

- PARCELABLE Class
  This code will run significantly faster. One of the reasons for this is that we are being explicit about the serialization process instead of using reflection to infer it. It also stands to reason that the code has been heavily optimized for this purpose.

To pass the PARCELABLE object between activities, `Intent.putExtra` is used.

ZATDROID uses the "sat" class that implements PARCELABLE and passed between activities providing the information of the satellite picked.

PASSING PARAMETERS:

- SHAREDPREFERENCES [10]

"The `sharedPreferences` *class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use* `sharedPreferences` *to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed)."*

ZATDROID uses `sharedPreferences` to share Strings that save names of satellites picked for the user.

- Final Static
  Constants are storaged in classes this way.

ZATDROID has a class named *k* for all constants used in mathematical calculations apart from other constants in other classes.

SHARING FILES:

- External - Internal Storage: [11]
  When building an app that uses the internal storage, the Android OS creates a unique folder, which will only be accessible from the app, so no other app, or even the user, can see what's in the folder.
  The external storage is more like a public storage, so for now, it's the SD card, but could become any other type of storage (remote hard drive, or anything else).
  The internal storage should only be used for application data, (preferences files and settings, sound or image media for the app to work). The external storage is often bigger. Besides, storing data on the internal storage may prevent the user to install other applications.

ZATDROID uses internal Storage to save the XML and XSD files because they are small and are used only for the application.

7.5. BREADCRUMBS NAVIGATION BUTTONS

In every screen at the top the user can find some buttons as indicated in figure 8. These so-called `BreadCrumbs` functions are:

- To be helpful as they provide information of the satellite type picked during the process of several screens. All types are shown so that the user, at first sight, remembers the types and name of the satellite picked.
- To allow the user to navigate through the previous screens, changing the type picked, just by clicking on one of the buttons of the breadcrumbs.

The layouts affected are created programmatically, not with XML. This is another example where layouts defined with code allow to add some more functionalities than with XML.

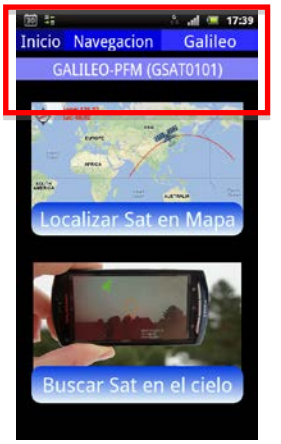


Figure 9. BreadCrumbs

7.6. GOOGLE SEARCH ACTIVITY

The first screen gives the opportunity to select searching a satellite by it type or launching a search by some key words. See Figure 9.

This last functionality uses GOOGLE Search Activity from Android. The button calls `onSearchRequested` method and the GOOGLE interface is launched. It opens a dialog with a keyboard where you can write key words to find a satellite. It also supports voice search.

CELESTRAK provides txt files with TLEs classified by type. To be able to perform the search in all satellite files provided by CELESTRAK, ZATDROID needs to download all files, merge them together in one file and then search. This process lasts some seconds, it is encoded inside an `Asynctask` and a progress bar is shown to the user.

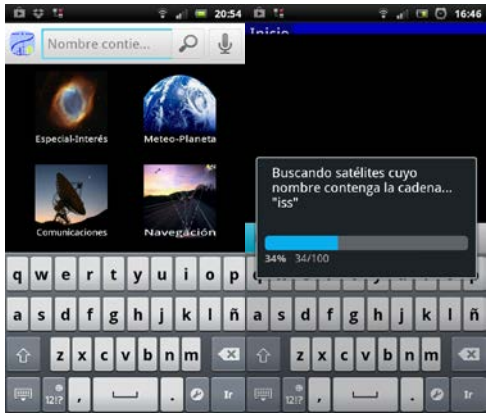


Figure 10. GOOGLE Search & progress bar

7.7. DEVICE LOCATION PROVIDER

Latitude, longitude and altitude of the device must be known whenever a prediction for the location of the satellite in the sky is intended to do. Besides knowing the satellite georeference, also device coordinates are essential.

With the help of [12] ZATDROID implements a customized `LocationListener` to retrieve **latitude** and **longitude** coordinates:

- Checks that Network or GPS is enabled. If not, it shows a message to the user requiring at least one connection.
- Firstly retrieves location from Network provider. It is quicker and usually devices enables network but not always GPS.
- Secondly, if GPS is enabled, as it is more precise, uses it to get a more accurate location.

Finally, **altitude** is also required.

- If GPS is enabled, it is easy. GPS gives it together with latitude and longitude.
- Otherwise, a web service is called using a `httpRequest`. There are two possibilities based on 3D maps of the Earth: [13]
  - USGS Elevation Query Web Service: [14]
  - Google Maps Elevation API Web Service: [15]

ZATDROID implements `Google Maps Elevation API Web Service`, although there is no big difference between them, advantages or disadvantages.

7.8. SENSORS MANAGEMENT

Method `mSensorManager` provides access to all sensors installed in the device. Three sensors are to be used by ZatDroid:

- **Accelerometer**: measures the acceleration applied to the device that gives the orientation.
- **Magnetic Field**: magnetic vector is provided device coordinate system.
- **Gravity**: Gravity vector is given in device coordinate system.

Vectors need a reference coordinate system and as the device is moving and rotating, the transformation matrix is also necessary. Two methods inside `mSensorManager` (`getRotationMatrix` and `getOrientation`) give information about the orientation and coordinate system of the device and the transformation matrix.

7.9. SMOOTH MOVEMENTS FILTER [16]

The augmented reality functionality requires to retrieve device orientation in real time. This information is used to locate the satellite icon on the screen when the user is rotating the device searching the right orientation (azimuth and elevation). As a result, the icon is moving through the screen.

This movement depends on the accuracy, updating time and speed of rotating the device. One method has been implemented to smooth the movement of the icon on the screen: `filterSmoothMovements`. It takes two factors into account:

- `SmoothFactorCompass`: so that the small jumps do not disturb
- `SmoothThresholdCompass`: minimum distance so that the icon jumps

7.10. MIGRATION OF SGP4 CODE FROM FORTRAN / C TO JAVA FOR ANDROID

Document [4] stated the definition of the methods to calculate the propagation of the orbits taking into account perturbations due to disturbance forces. This was in 1980 in FORTRAN language. A new revision was developed in 2006 in C [6] Some other reviews and improvements have been implemented but the heart still remains.

ZATDROID implements the code from 1980 [4] with some modifications (not all) from 2006 [6] migrated to JAVA for ANDROID.

7.11. AUGMENTED REALITY VIEW: LAYERS OVER CAMERA VIEW.

Augmented Reality views means to be able to merge the camera view with some other artificial layers with digital information and let both worlds interchange information and interact with each other.

This has been achieved thanks to the `GLSurfaceView` and `SurfaceView` classes and then overlapping them with `addViews`. All these layouts have been created programmatically, as already explained before to increase customization.

ZATDROID implements 6 layers at the same time: (Figure 10)

1) The camera view, as usual.
2) A rectangle in the centre that changes colour when device orientation is directly pointing at the satellite in the sky
3) An arrow indicating the direction of the satellite to rotate the device
4) An icon symbolizing the satellite in the right azimuth and elevation.
5) `TextView` with the name and orientation (azimuth and elevation) of the satellite
6) `TextView` with elevation and azimuth of the device updated in real time.

Figure 11. Augmented reality view

7.12. OPENGL USAGE

When implemented AR View, it was complicated to visualize the icon of the sat only when the device was pointing at the direction of the satellite. OPENGL has been used to rotate the virtual camera that shows the icon at the same time as the device camera is rotated by the user.

- `GLU.gluLookAt` and `GL10.glFrustumf` control the focus, position and movements of the virtual camera.
- As explained before, the moving coordinate systems made the vectors transformations really complicated. Complex matrixes were used.
- The geometry of the arrow and rectangle was easy, but the icon was inside a rectangle and was always moving, so updating the geometry of the icon really increased the difficulty.

7.13. GOOGLE MAPS VIEW

As one of the two main functionalities, locating the satellite icon in the GOOGLE MAPS in real time with latitude and longitude is exciting. These maps have the same appearance as the maps you see in your computer, but here you can customize to show whatever you want to (Figure 10)

- `MapActivity` is the object used in ZATDROID. Newer versions of Android has deprecated `MapView` and replaced it with `GoogleMap` object. The heart is similar but other complements are improved. For this app, `MapView` fits all requirements.
- `com.google.android.maps.Overlay` is the layer where all the information is added and shown onto the map.

- A canvas let the program add the satellite icon as a bitmap together with the `geoPoint` represented by the latitude and longitude of the satellite.
- Also in this canvas a `drawPath` object allows to draw a trajectory line. This trajectory represents the orbit in its last 20 minutes and its predicted 20 coming minutes, all calculated iterating the propagating models.
- And the last layer is the `textView` with name, latitude and longitude of the satellite updated in real time.

The satellite icon is updated almost instantly. The refreshing delay for the trajectory is one second though, that is, the trajectory stays old when zooming or moving the map. It was tested to update the trajectory several times per second so that the visual appearance was smoother, but the high amount of calculations, made it not possible for the program.

If the user ever sees a satellite at high altitudes (30.000 Km) and the trajectory is not drawn, it is not a bug. The satellite follows a geostationary orbit and therefore it is fixed. Actually, it is rotating with the Earth, so for an inhabitant it is always in the same position, same latitude and longitude.

Figure 12. GOOGLE MAPS View

7.14. XML/XSD MANAGEMENT

The TLEs downloaded from CELESTRAK is provided in txt format as described in appendix 1. One of the objectives of this project was to learn XML language. Therefore, the data management in ZATDROID was in XML. So TLEs are transformed into XML. The XML format is shown in appendix 2.

The way to create XML from the txt is done manually, by parsing txt NORAD format and then writing XML tags and content. The files are storaged in the internal memory, as explained before in this section.

7.15. SAX MANAGEMENT

SAX (*Simple API for XML*) and DOM (*Document Object Model*) are the two methods to deal with XML data in ANDROID. A brief description is now given: [17]

SAX:

- Parses node by node
- Doesn't store the XML in memory
- We can´t insert or delete a node
- SAX is an event based parser
- SAX is a Simple API for XML
- Doesn't preserve comments
- SAX generally runs a little faster than DOM

DOM:

- Stores the entire XML document into memory before processing
- Occupies more memory
- We can insert or delete nodes
- Traverse in any direction.
- DOM is a tree model parser
- Document Object Model (DOM) API
- Preserves comments

ZATDROID is:

- Dealing with big documents
- Not inserting nodes, just reading
- Intending not to use a lot of memory
- Not reading the whole document, just the node required.
- Not needing to store the whole document and creating the DOM tree.

So SAX was decided to be the method for the searches into XML. ZATDROID looks into the XML for the satellite picked by the user and read all the information of this certain satellite to create afterwards a "sat" JAVA object.

8. PLANNING AND BUDGET

8.1. WORK ESTIMATION

FUNCTION POINTS is considered a useful tool to make an estimation of the lines of code. But this project has some special characteristics that may not fit with a typical estimation, so that the results would not be coherent.

- Only one person is in charge of the whole project
- The training stage must be extremely large, as the programmer is untrained.
- The orbital mechanics stage is out of the reach of the FUNCTION POINTS analysis
- There is no routine in the time dedicated every week to the project because the programmer has a full time job. The project is being developed in his free time.
- In any moment the project could be delayed due to any reason.

So the planning is created from the stages and taking into account experts advices, self experience in other projects from other disciplines.

8.2. WORK FLOW AND TASKS

The work flow defines the stages of the project. Figure 12.

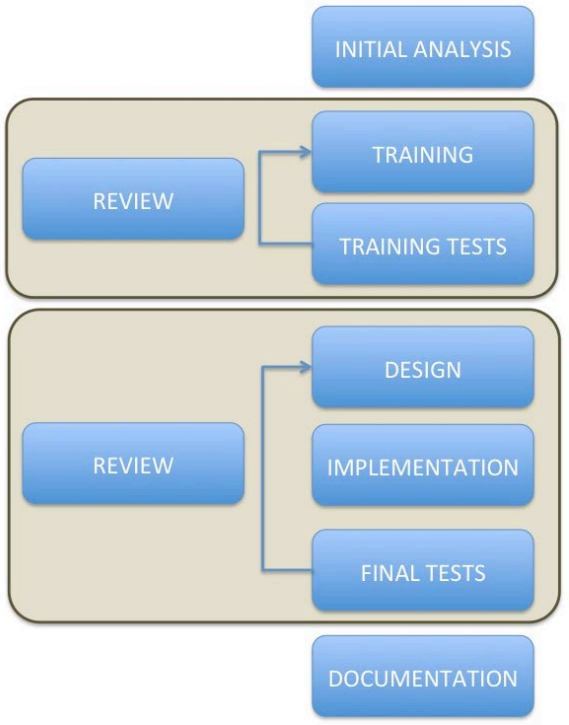


Figure 13. Work Flow Stages

Taking the stages definition as a starting point, a detailed tasks breakdown structure can be made.

1. **Initial analysis**
   - 1.1. Objectives definition
   - 1.2. Resources analysis
     - 1.2.1. Human Resources
     - 1.2.2. Technical resources
   - 1.3. Schedule
     - 1.3.1. Initial tasks definition
     - 1.3.2. Initial requirements definition
     - 1.3.3. Schedule estimated
2. **Training**
   - 2.1. Android programming. SDK
     - 2.1.1. Eclipse SDK
     - 2.1.2. Activities flow
     - 2.1.3. Layouts
       - 2.1.3.1. Types
       - 2.1.3.2. Components
       - 2.1.3.3. Programmatically
       - 2.1.3.4. Threads
     - 2.1.4. Parameters
     - 2.1.5. Classes
     - 2.1.6. Sharing and passing information
   - 2.2. GOOGLE MAPS View
     - 2.2.1. MapView
     - 2.2.2. Layers and Markers
     - 2.2.3. Geometry
   - 2.3. Augmented reality views
     - 2.3.1. Camera View
     - 2.3.2. Layers
     - 2.3.3. Geometry
   - 2.4. OpenGL
     - 2.4.1. Coordinate systems
     - 2.4.2. Transformation Matrix
     - 2.4.3. Geometry
   - 2.5. XML management
     - 2.5.1. Understanding the language
     - 2.5.2. Creating files
     - 2.5.3. XSD scheme
     - 2.5.4. Search: SAX, DOM
   - 2.6. Orbital Mechanics calculations. TLEs. SGP4/SDP4.
     - 2.6.1. Satellites orbits
     - 2.6.2. Equations
     - 2.6.3. Methods understanding
     - 2.6.4. Programming code
   - 2.7. State of the art: applications
3. **Training Tests** (carried out at the same time as the training)
   - 3.1. Activities tests
   - 3.2. Sensor tests
   - 3.3. OPENGL tests

3.4. Google Maps tests
3.5. XML Management
3.6. Augmented Reality tests: camera view and layers
3.7. Orbital Mechanics Calculations tests.

4. Design

4.1. Requirements.
4.1.1. Analysis of the estimated requirements
4.1.2. Set the final requirements
4.2. Classes diagram
4.2.1. Activities classes
4.2.2. Calculations Classes
4.3. Sequence Diagram
4.4. GUI Design

5. Implementation

5.1. Android core
5.1.1. Activities Structure
5.1.2. Sensors
5.1.3. Sharing and passing data
5.2. GUI
5.2.1. Screen definition
5.2.2. Icons and Text
5.2.3. Multilanguage support
5.3. XML files management
5.3.1. XML Parsing
5.3.2. XSD Creation
5.3.3. Search: SAX Handling
5.4. Google Maps Activity
5.4.1. MapView
5.4.2. Markers
5.4.3. Geometry
5.4.4. Layers
5.5. Augmented Reality
5.5.1. Camera View
5.5.2. Layers
5.5.3. OpenGL Geometry
5.5.4. Device orientation sync with sat icon
5.6. Orbital Mechanics Code. SGP4/SDP4
5.6.1. TLE download
5.6.2. Time calculations
5.6.3. SGP4/SDP4 Code

6. Final Tests

6.1. Activities structure
6.2. GUI
6.2.1. User experience
6.2.2. Activities
6.3. AR functionality
6.3.1. Layers
6.3.2. Icon movement with device orientation
6.4. GOOGLE MAPS functionality
6.4.1. Icon marker

6.4.2. Trajectory line
6.5. Orbits propagation check
6.6. Devices compatibility
6.7. Android versions compatibility

7. Documentation
7.1. Final Year Project Report
7.2. Technical Manual
7.3. User Manual

8.3. SCHEDULE ESTIMATED

Not all the tasks are written in the open project file, as they are too specific to give a general view.

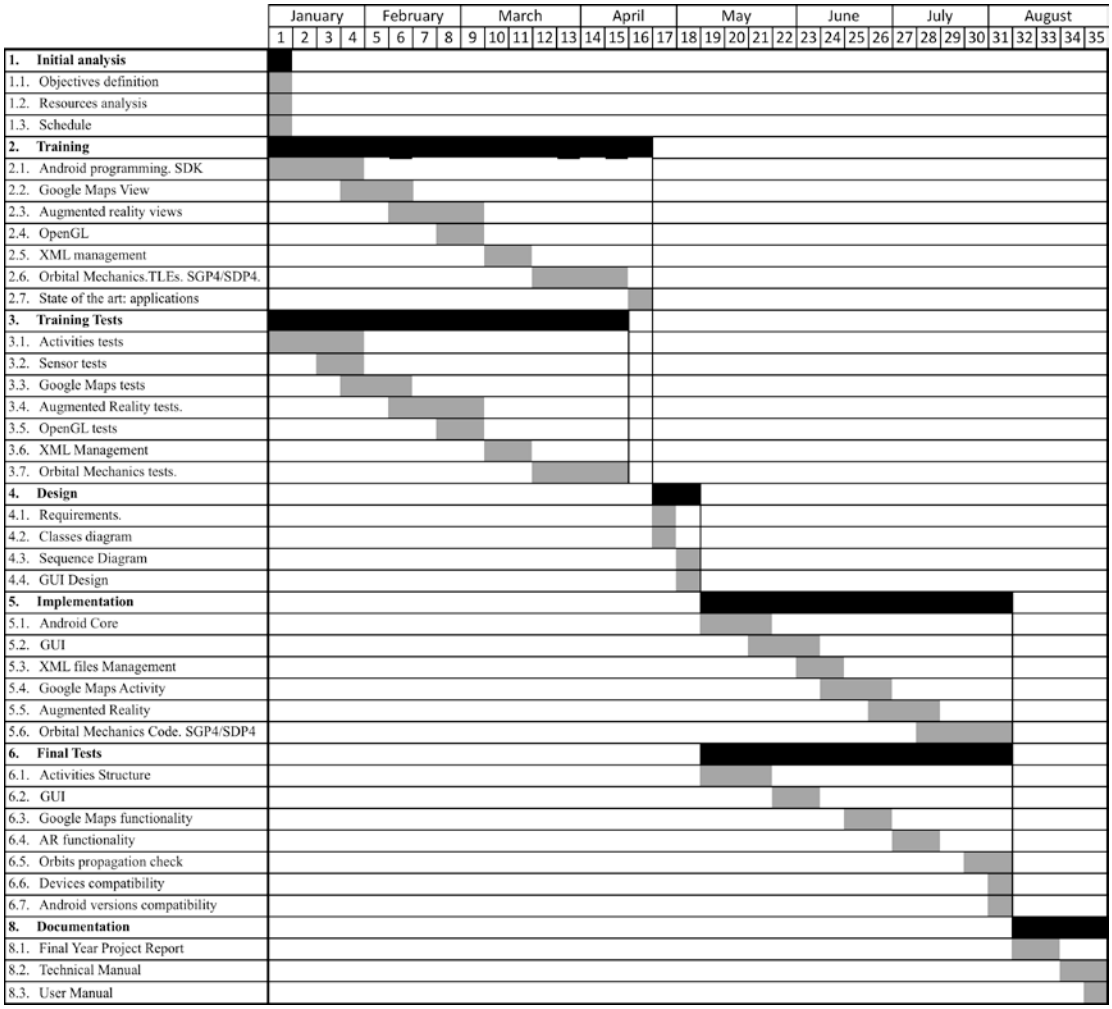


Figure 14. Estimated Schedule

The total estimated time is 35 weeks, with one person working full time in the project, 5 days a week, 1400 hours.

8.4. SCHEDULE ACHIEVED

The beginning of 2012 was the start point of this project. After some months working, I was offered a new job, so from July 2012 until December 2012 the project stayed in stand-by. Then, in January 2013 it all begun again and in May 2013 I finally finished it. June was again dedicated to my job, and July and August has been the time to finish. So I had 13 months of actual work in three periods:

January - June 2012 January	January - May 2013	July-August 2013
6 Months	5 Months	2 Months
13 Months		

Figure 15. Actual Work periods

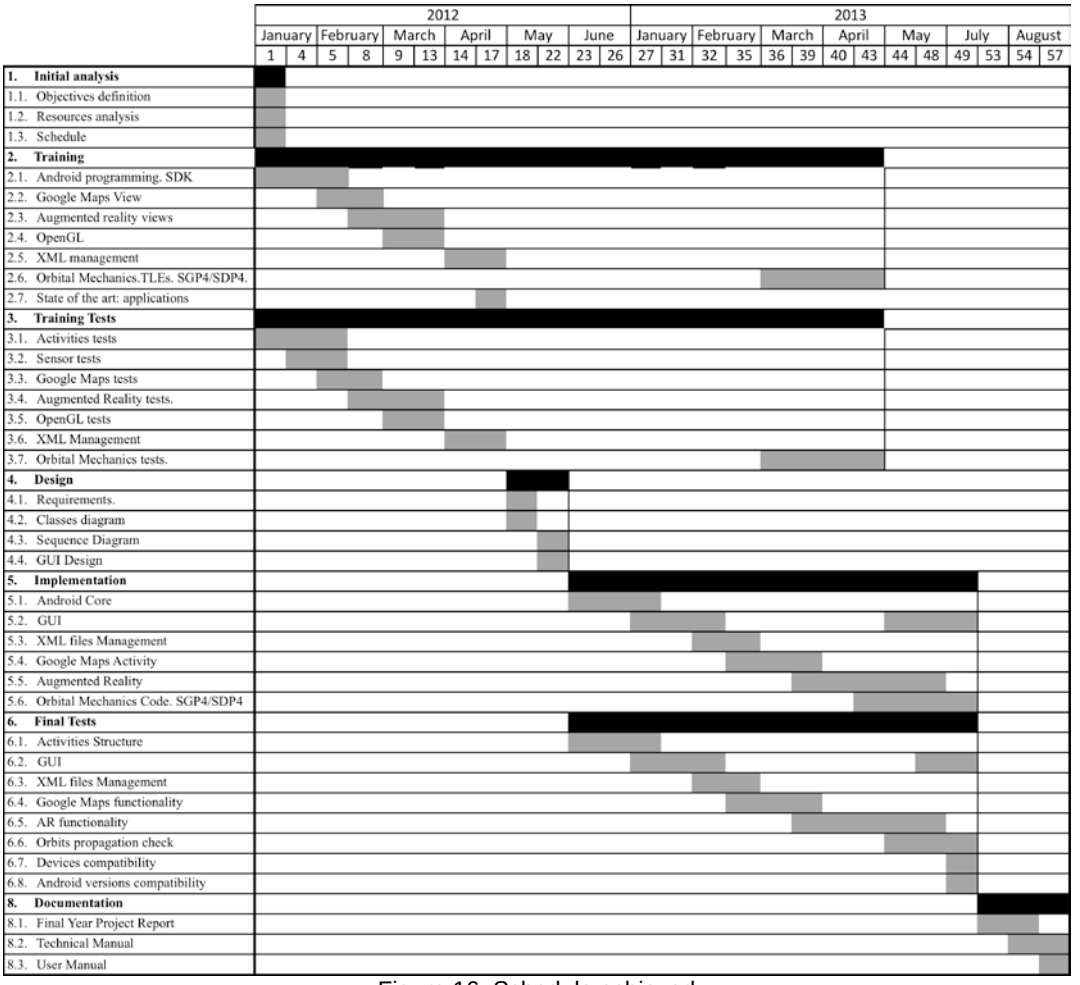


Figure 16. Schedule achieved

Moreover, only 4 hours per day (at most) were available to go forward with the project, although weekends were also dedicated to the project. So, from the 8 months estimated, one person full time, the project has been developed in 13 months. It may be thought that 16 (double) months were needed, but if holidays and weekends are to be added to the schedule, 13 months are enough. It may be agreed that 1400 hours is a good estimation for the project.

The main deviation noticed about technical issues has been that, after the first implementation of the GUI with `ListViews`, tests performed with users revealed that the GUI was not friendly and gave no information to the user about choices made in the satellite type. So a new process of improving the GUI was carried out:

- Adding `BreadCrumbs` to the screens
- Changing `ListViews` for icon based layouts.

After June 2012, the implementation was started and most of the training finished. So the implementation in 2013 went forward quickly. The key point that was entirely performed in 2013 was the Orbital mechanics training and implementation. Actually, training and implementation was mixed and this feature was the last to be finished.

8.5. COSTS ESTIMATION

Once again, there are some tools to estimate costs for a project. COCOMO may be one of the most used in the subjects of the degree. But for this case, no tool has been used to calculate an estimation, because the project is simple looking from the costs point of view.

- Only one person is working on it.
- Most of the time has been invested in training. Are these hours costs?
- The material resources are easily measured.

A final budget is detailed in the next section, which would be completely similar to the estimated one, except for little things.

8.6. DETAILED BUDGET

Resource	% Use	Total Resource Cost	Total cost for the project
Computer	50%	1000€	500€
MS Office 2010 Prof	25%	480€	120€
Eclipse SW	100%	0	0
Start UML	100%	0	0
Printer & toner	10%	200€	20€
Paper office material	100%	20€	20€
ADSL (13 months)	25%	400€	100€
Sony Ericson Neo V	75%	150€	112€
5 project copies printed and bookbound	100%	300€	300€
		TOTAL:	1.172€

Human resources	€ per Hours	Hours	Total cost for the project
Computer Science Engineer	12€	1.400€	16.800€
		TOTAL:	16.800€

Figure 17. Detailed Budget

So the final costs are **17.972€.** This is a huge quantity for the project.

- Half of the working hours have been training. These may be not taken into account as the programmer is supposed to have the skills before the contract. So the prize would reduce 8.000 €, and the app final cost would be **9000€.**
- Bearing in mind the idea of **monetizing** the App in the market, if we sell it in Google Play:
  - Costs are 25€ once to register.
  - The App. price: 70% for the seller, 30% for Google.

MONETIZING THE APP		
Objective: 9.000€		
App Price	€ for the author	Downloads
€ 0,20	€ 0,14	64.286
€ 0,30	€ 0,21	42.857
€ 0,40	€ 0,28	32.143
€ 0,50	€ 0,35	25.714
€ 0,60	€ 0,42	21.429
€ 0,70	€ 0,49	18.367
€ 0,80	€ 0,56	16.071
€ 0,90	€ 0,63	14.286
€ 1,00	€ 0,70	12.857
€ 1,10	€ 0,77	11.688
€ 1,20	€ 0,84	10.714
€ 1,30	€ 0,91	9.890
€ 1,40	€ 0,98	9.184
€ 1,50	€ 1,05	8.571

Figure 18. Monetizing the App. Price Study.

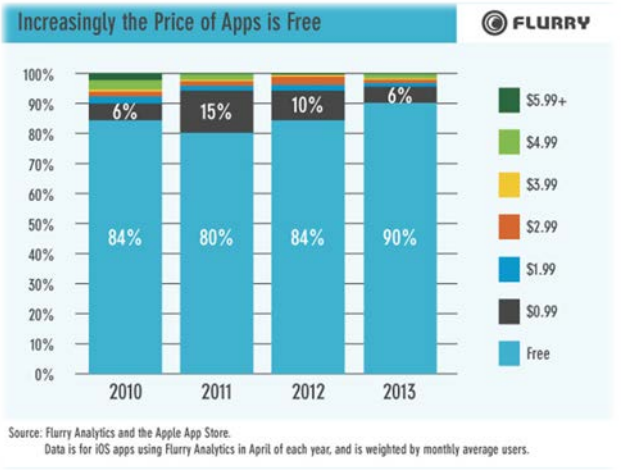


Figure 19. App Prices on the market

Figure 18 explains how the rate of users downloading free apps is increasing.

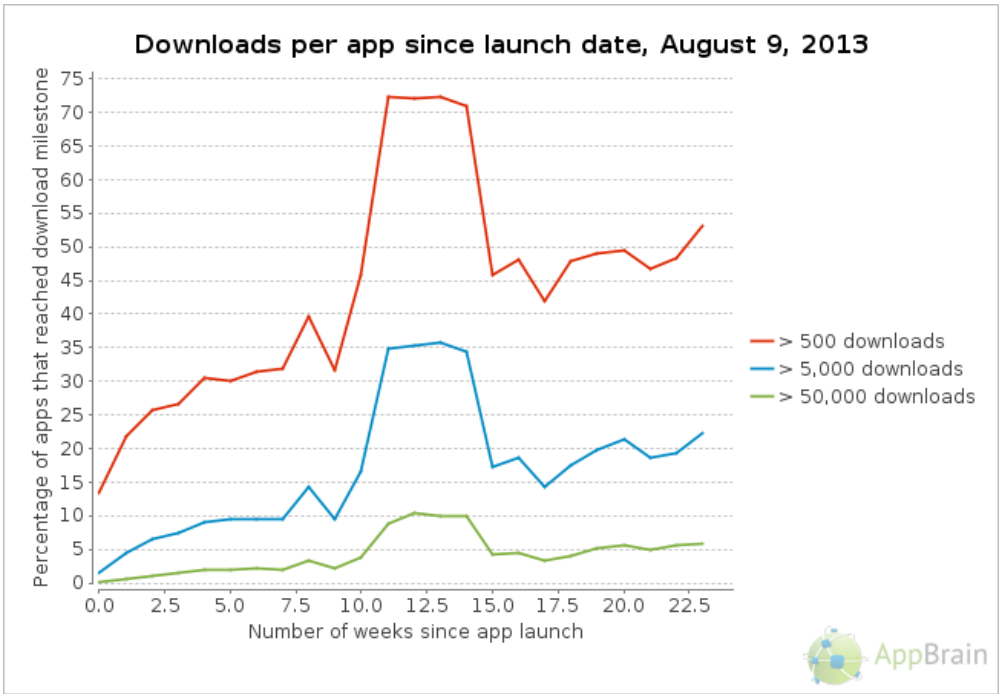


Figure 20. Downloads per App

Figure 19 shows how difficult is to gain more than 5.000 downloads. Only 20% (average) of the apps reach more than 5.000 downloads after 20 weeks in the market.

With the comparison of the price and downloads needed to reach 9.000€, it is difficult for ZATDROID to be profitable.

9. FUTURE WORK

Once the initial objectives are achieved and after developing this project, there are some issues that could be improved or added to the requirements of ZATDROID. Here are some of them:

9.1. FULL DATABASE HOSTED IN SERVER

It would be nice to the user experience that there is no need to choose a satellite to performed a GOOGLE MAPS View or an Augmented Reality View. Instead, a complete database with all satellites from CELESTRAK could be developed. Every satellite would have the real time parameters updated so that the device could show all of them (or just some with filters) when initializing the Google Maps View or the Augmented reality view.

This new feature would require a large computational. Therefore, it would mean a change in the core of the project. The idea should be to implement the calculation modules in a server and let the device connect to the server to provide a wide range of options in the visualizations functionalities.

9.2. MIGRATION TO ANDROID 4.3 NEW FEATURES

ZATDROID has been developed under ANDROID 2.3. and tested in a *Sony Xperia Neo V*. From the start of the project (January 2012) ANDROID has launched new versions with new features: OPENGL 3.0, Optimized Location and Sensor Capabilities, transparent overlays, fragments, Action Bar…

9.3. MIGRATION TO IOS

It would be interesting to be able to migrate the whole app to IOS, as it will be an opportunity to expand the market. Nevertheless, this is complicate as it involves lot of modifications.

10. LIST OF REFERENCES

[1] Csete, A. (2009). GPREDICT: Free, Real-Time Satellite Tracking and Orbit Prediction Softtware. Retrieved from http://gpredict.oz9aec.net/

[2] Daum, P. (2005). VIS SAT: A Satellite Footprint Visualization Tool (Master Thesis). Lancaster University.

[3] Grzegorczyk , M. (2013). SATFINDER. Retrieved from http://esys.com.pl/satfinder

[4] *Hoots, Felix R., & Roehrich, R. L. (1980). Spacetrack Report #3: Models for Propagation of the NORAD Element Sets. Colorado Springs, CO: U.S. Air Force Aerospace Defense Command.*

[5] SATELLITE AR. (2011). ANALYTICAL GRAPHICS, INC. RETRIEVED FROM HTTP://SPACEDATA.AGI.COM/MOBILEAPPS/ABOUT.HTM

[6] Vallado, D. A., Crawford, P., Hujsak, R., & Kelso, T. S. (2006). *Revisiting spacetrack report #3*. In Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference, 2006 (Vol. 3, pp. 1984–2071). (*AIAA-2006-6753)*

[7] Videotutorials YOUTUBE, stackoverflow.com

[8] WIKIPEDIA.COM

[9] http://www.developerphil.com/parcelable-vs-serializable/

[10] developer.android.com/

[11] http://stackoverflow.com/questions/5092591/what-are-the-differences-among-internal-storage-external-storage-sd-card-and-r

[12] http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/

[13] http://stackoverflow.com/questions/1995998/android-get-altitude-by-longitude-and-latitude

[14] http://gisdata.usgs.gov/xmlwebservices2/elevation_service.asmx

[15] https://developers.google.com/maps/documentation/elevation

[16] http://stackoverflow.com/questions/4699417/android-compass-orientation-on-unreliable-low-pass-filter

[17] http://stackoverflow.com/questions/12140851/sax-vs-dom-in-android

11. LIST OF ABBREVIATIONS

SDK: Software Development Kit
AR: Augmented Reality
NORAD: North American Aerospace Defence Command
TLEs: Two Line Elements Sets
XML: eXtensible Markup Language
XSD: XML Scheme Definition
SAX: Simple API for XML
SGP: Simplified General Propagation Model
SGP4: Simplified General Propagation Model Version 4
SDP4: Simplified General Deep Space Perturbation model version 4
GUI: Graphical User Interface

12. APPENDICES

APPENDIX 1: TLEs Structure

```
AAAAAAAAAAAAAAAAAAAAAAA
1 NNNNNU NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNN +NNNNN-N +NNNNN-N N NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNN NNN.NNNN NNN.NNNN NN.NNNNNNNNNNNNN
```

Line 1	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
08	Classification (U=Unclassified)
10-11	International Designator (Last two digits of launch year)
12-14	International Designator (Launch number of the year)
15-17	International Designator (Piece of the launch)
19-20	Epoch Year (Last two digits of year)
21-32	Epoch (Day of the year and fractional portion of the day)
34-43	First Time Derivative of the Mean Motion
45-52	Second Time Derivative of Mean Motion (decimal point assumed)
54-61	BSTAR drag term (decimal point assumed)
63	Ephemeris type
65-68	Element number
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)

Line 2	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
09-16	Inclination [Degrees]
18-25	Right Ascension of the Ascending Node [Degrees]
27-33	Eccentricity (decimal point assumed)
35-42	Argument of Perigee [Degrees]
44-51	Mean Anomaly [Degrees]
53-63	Mean Motion [Revs per day]
64-68	Revolution number at epoch [Revs]
69	Checksum (Modulo 10)

```
GPS BIIA-15 (PRN 27)
1 22108U 92058A 05274.69046540 .00000036 00000-0 10000-3 0 4762
2 22108 54.7300 89.1232 0185387 243.8332 114.2955 2.00554288 95652
```

APPENDIX 2: XML Scheme

```
<sat>
      <info>
            <name>NOAA 1 [-]</name>
            <number>04793U</number>
      </info>
      <keplerianElements>
            <inclination>102.0640</inclination>
            <rightAsensionAscendingNode>216.2473</rightAsensionAscendingNode>
            <eccentricity>0.0032328</eccentricity>
            <argumentPerigee>114.5254</argumentPerigee>
            <meanAnomaly>245.9193</meanAnomaly>
            <meanMotion>12.5395227</meanMotion>
            <epochYear>12</epochYear>
            <epoch>189.55987732</epoch>
      </keplerianElements>
      <otherParameters>
            <meanMotionDerivate>-.00000031</meanMotionDerivate>
            <bstarDragTerm>10000E-3</bstarDragTerm>
            <ephemeridesType>0</ephemeridesType>
      </otherParameters>
</sat>
```