TABLE DES MATIERES

LISTE DES ABREVIATIONS

API : Application Progamming Interface (interface de programmation applicative)

Go : Gigaoctet

GNSS : Global Navigation Satellite System

GPS : Global Positioning System

GSM : Global System for Mobile

JVM : Java Virtual Machine

Mb: Megabits

MP3 : MPEG-1/2 Audio Layer 3

OpenGL : Open Graphis Library

RAM : Random Access Memory

SMS : Short Message Service

SQLite : Structured Query Language Lite

UMTS : Universal Mobile Telecommunications System

XML : Extensible Markup Language

LISTE DES ILLUSTRATIONS

Organigrammes

Tableaux

Figures

Introduction

La Topographie est une science complexe relatif à l'établissement des plans et/ou de cartes . La topométrie qui est son allier regroupe les opérations permettant la réalisation de ces dernières et leurs représentations. Etant à la base du génie civil et donc en amont de tous travaux de constructions ou d'aménagements, elle se doit d'être optimisée et modernisée continuellement, tant par ses méthodes que par ses traitements.

Nous vivons actuellement dans une ère où la modernité et l'innovation sont omniprésentes quel que soit le domaine d' étude. La topographie n'est pas en reste et reçoit aussi son lot d'innovations(scanner trois dimensions, positionnement par satellite, appareil de mesure robotisé...) ceci ayant toujours pour but d'optimiser les travaux de terrain et de bureau.

Cependant, dans notre pays, bon nombre de géomètres n'ont pas encore accès à ces technologies modernes et sont réduits la plupart du temps à l'utilisation de matériels obsolètes qui entachent par leurs vétustés la qualité du travail obtenu. Les carnets de terrain électroniques sont peu répandus et les données sont encore la plupart du temps sur papier, rallongeant le temps de travail au bureau.

Pour contribuer, ainsi, à la modernisation de la topographie dans notre pays, nous avons eu l'idée de d'élaborer une application mobile pour téléphones intelligents et tablettes fonctionnant sous le système Android d'où le titre de ce mémoire : "**Développement d'une application sur mobile, aide aux travaux topographiques de terrains**". Le but de ce logiciel est, comme ce titre l'indique, d'assister le topographe dans sa collecte de données sur le terrain. Cette application fera office de carnet de levé, d'outil de positionnement approché et d'outil de visualisation. Cet ouvrage traitera alors, successivement du langage de programmation et de la plateforme de développement, des algorithmes de calculs topographiques présents dans l'application, et enfin de la conception et la présentation de l'application proprement dite.

Partie I

Généralités sur le langage Java et le système Android

CHAPITRE 1

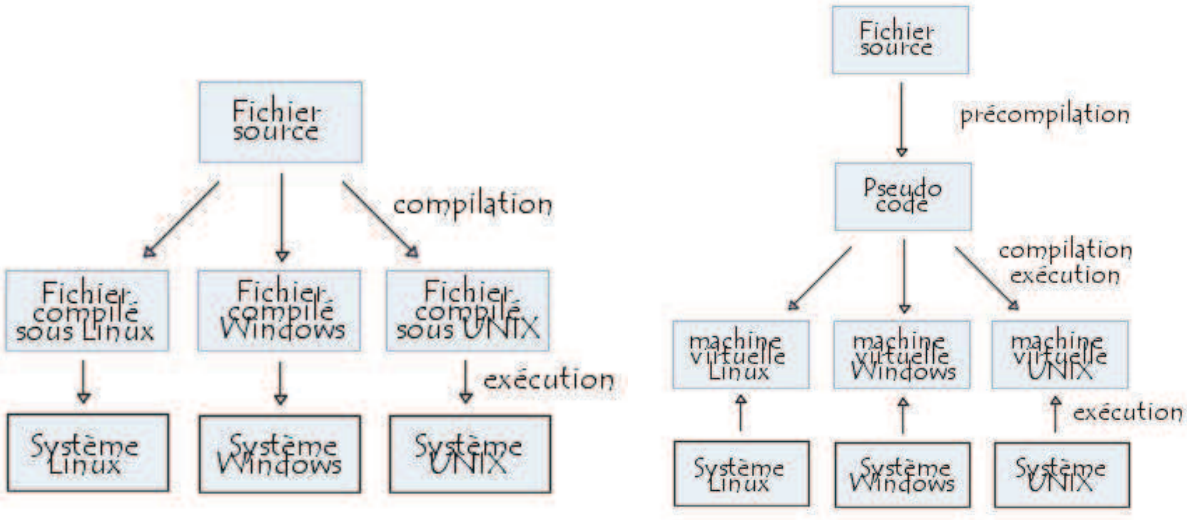
LE LANGAGE JAVA

1. Origine et historique du langage

Le java, appelé Oak à l'origine, est un langage de programmation récent né en 1990. Au départ, il a été conçu pour rendre les appareils électroménagers indépendants et interactifs entre eux. L'idée était de pouvoir commander ces appareils électroménagers (machines à laver, ...) à distance avec une télécommande. Mais malheureusement ce fut un échec. Alors pourquoi le java n'a pas disparu ? Tout simplement parce que dans les années 90, il y eut aussi l'expansion d'internet, et le java, étant un langage orienté internet, a été repris pour rendre les pages interactives entre elles. C'est là que java connu un grand succès car il peut être utilisé dans beaucoup de domaines différents. De plus c'est un langage qui peut s'utiliser avec n'importe quel machine et système d'exploitation : il est portable. En 1993 : il y a eu une crise avec java car une entreprise nommée "Sun" fut fondée et elle a repris le langage java, car Microsoft refusait de rendre son langage publique. Sun possédait donc le java, ce qui le fut devenir un langage privé et donc non accessible, ce fut une grande perte pour les ingénieurs. Mais heureusement, en 2006, Sun a accepter de rendre le langage java publique.

2. Environnement d'exécution

Le langage Java est un langage orienté objet qui doit être compilé. Cependant, le compilateur Java ne produit pas directement un fichier exécutable, mais du code intermédiaire sous la forme d'un ou plusieurs fichiers dont l'extension est *.class* . Ce code intermédiaire est appelé *bytecode*. Pour exécuter le programme, il faut utiliser la machine virtuelle Java qui va interpréter le code intermédiaire en vue de l'exécution du programme.



Organigramme 1 : Compilation et exécution d'un fichier .class [1]

L'utilisation d'une machine virtuelle a l'énorme avantage de garantir une vraie portabilité. Il existe des machines virtuelles Java pour de très nombreux environnements : Windows, MacOS, Linux et autres.

Ces machines virtuelles sont capables d'exécuter exactement le même code intermédiaire (les mêmes fichiers Java en *bytecode*) avec une totale compatibilité. C'est là une situation unique et assez remarquable qui a fait le succès de ce langage. La machine virtuelle Java n'est pas uniquement développée sur des ordinateurs classiques, une multitude d'appareils disposent d'une machine virtuelle Java : téléphones portables, assistants personnels (PDA).

3. Structure d'un programme en Java

Un programme Java est une suite d'instructions exécutées de manière séquentielle. D'une manière générale, les instructions sont séparées par des points-virgules *";"*. Ces instructions s'exécutent de gauche à droite et de haut en bas. Elles peuvent être organisées dans des blocs.

Tout programme Java contient au moins une classe, nommée par convention avec une majuscule contrairement aux méthodes. De plus, il nécessite un point d'entrée. Pour un programme simple, le point d'entrée est la méthode "*main*" qui doit être publique, statique et située dans une classe qui elle même doit être publique (d'où les mots-clés public et static) :

```
public class Main{
      public static void main(String arg[]){
            //intruction du programme
       }
}
```

Un bloc d'instructions est une suite d'instructions commençant par une accolade ouvrante { et se terminant par une accolade fermante } . Il est considéré comme une seule instruction par les instructions *for*, *while*, *if*, *else* et *case*. Ce bloc délimite également la portée des variables. Toute variable déclarée dans ce bloc n'est accessible qu'à partir de ce bloc, et n'existe que durant l'exécution du bloc.

4. Variables et opérateurs

4.1. Les variables

Une variable est ce qui va nous permettre de stocker des informations de toute sorte (chiffres, résultats de calcul, des tableaux, des renseignements fournis par l'utilisateur...). En Java, nous avons deux type de variables : des variables de type simple ou "primitif" et des variables de type complexe ou encore des objets.

Nom	Taille en octets lors des calculs	Valeur par défaut
boolean	Un seul bit suffit, mais on réserve souvent un octet pour les stocker.	false
byte	1	0x00
short	2	0x0000
int	4	0
long	8	0l
char	2	'x0000'
float	4	0f
double	8	0.0
Object	Dépendant de la machine virtuelle	`null`

Tableau 1 : Type de variable [2]

Pour instancier une variable, la syntaxe est la suivante : ***`NomDuType maVariable;`***

4.2. Les opérateurs

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ... On distingue plusieurs types d'opérateurs :

- les opérateurs de calcul (**+** , **-** , ***** , **/** , **=** , **%**)
- les opérateurs d'assignation (**+=** , **-=** , ***=** , **/=** , **%=**)
- les opérateurs d'incrémentation (**++** , **--**)
- les opérateurs de comparaison (**==** , **<** , **<=** , **>** , **>=** , **!=**)
- les opérateurs logiques (**||** , **&&**)

5. Les conditions et les boucles [1]

5.1. Les conditions

5.1.1. L'instruction *if*

L'instruction *if* permet d'exécuter une série d'instructions lorsqu'une condition est réalisée. La syntaxe de cette expression est la suivante :

```
if(condition réalisée){
	Liste d'instruction respectant la condition
} else {
	Liste d'instruction quand la condition n'est plus respectée
}
```

5.1.2. L'instruction *switch*

L'instruction *switch* permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Sa syntaxe est la suivante :

```
switch(variable) {
	case valeur1:
		/*instructions*/
		break;
	case valeur2:
		/*instructions*/
		break;
default : /*instructions par defaut si aucune correspondance*/
}
```

5.2. Les boucles

5.2.1. La boucle *for*

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructionsDans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur, la condition sur la variable pour laquelle la boucle s'arrête et enfin une instruction qui incrémente (ou décrémente) le compteur. La syntaxe de cette expression est la suivante :

```
for(compteur;condition;modification du compteur){
	instructions
}
```

5.2.2. La boucle *while* et *do while*

La boucle *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions. La syntaxe est la suivante :

```
while(condition réalisée) {
	liste d'instruction
}
```

La boucle *do while* est une variante de la boucle *while*, où la condition d'arrêt est testée après que les instructions aient été exécutées. La syntaxe est la suivante :

```
do{
	instructions
}while(condition réalisée)
```

CHAPITRE 2

LE SYSTEME ANDROID

1. Historiques [3][4]

En 2003, Andy Rubin, accompagné d'autres grands noms, a décidé de lancer Android Inc., une entreprise destinée à permettre aux "*appareils mobiles d'être plus concentré sur la localisation et les préférences qui lui sont propres*". Rubin et Android Inc. ont créé une nouvelle forme de système d'exploitation mobile : une plateforme simple et fonctionnelle, basée sur le kernel Linux, équipée d'une série d'outils conçus pour faciliter l'accès aux développeurs. Tout cela dans l'objectif d'avoir un système libre d'utilisation pour quiconque le souhaiterait, de développer un système d'exploitation mobile plus intelligent, qui ne se contenterait pas uniquement de permettre d'envoyer des SMS et transmettre des appels, mais qui devait permettre à l'utilisateur d'interagir avec son environnement (notamment avec son emplacement géographique). Ses principaux concurrents à l'époque étaient Symbian et Windows Mobile.

Cet aspect était assez convaincant pour Larry Page, fondateur de Google, qui s'est rapidement intéressé à ce projet malgré l'orientation de Google à l'époque qui se concentrait principalement sur le développement d'un outil de recherche. En 2005, Google fait l'acquisition de Android Inc., et la division Mobile de Google vu ainsi le jour.

C'est en 2007 que les choses s'envenimèrent. À cette époque, les constructeurs concevaient tous un système d'exploitation spécifique pour leurs téléphones, et il n'y avait aucune base commune entre les systèmes d'exploitation mobiles de deux constructeurs différents. Ce système entravait la possibilité de développer facilement des applications qui s'adapteraient à tous les téléphones, surtout entre constructeurs, puisque la base était complètement différente. Un développeur était plutôt spécialisé dans un système particulier et il devait se contenter de langages de bas niveaux comme le C ou le C++. De plus, les constructeurs faisaient en sorte de livrer des bibliothèques de développement très réduites de manière à dissimuler leurs secrets de fabrication. En janvier 2007, Apple dévoilait l'iPhone, un téléphone tout simplement révolutionnaire pour l'époque, capable d'aller sur internet, de lire des vidéos, etc. L'annonce est un désastre pour les autres constructeurs, qui doivent s'aligner sur cette nouvelle concurrence. Le problème étant que pour atteindre le niveau d'iOS (iPhone OS, le système d'exploitation pour iPhone), il aurait fallu des années de recherche et développement à chaque constructeur...

Google revient sur la scène avec une stratégie particulièrement bien ficelée, où 10 millions de dollars seront offerts aux développeurs créant les 10 meilleures applications Android, à partir d'une première version SDK rendue publique. À ce stade, les intentions de Google sont devenues beaucoup plus claires : il ne s'agit pas seulement de créer un nouvel iPhone, mais de créer un système flexible et adaptable différent de celui d'Apple. Ce logiciel constituerait un écosystème aussi indépendant que possible des constructeurs, et ouvert sur le monde des développeurs. On retrouvait là tout l'esprit insufflé par Andy Rubin.

C'est pourquoi est créée en novembre de l'année 2007 l'Open Handset Alliance, et qui comptait à sa création 35 entreprises évoluant dans l'univers du mobile, dont Google. Cette alliance a pour but de développer un système *open source* (c'est-à-dire dont le code source est accessible à tous) pour l'exploitation sur mobile et ainsi concurrencer les systèmes propriétaires, en particulier iOS. Cette alliance a pour logiciel vedette Android, mais il ne s'agit pas de sa seule activité.

2. La philosophie et les avantages d'Android [4]

2.1. Open source

Le contrat de licence pour Android respecte les principes de l'*open source*, c'est-à-dire que vous pouvez à tout moment télécharger les sources et les modifier selon vos goûts. Notons au passage qu'Android utilise des bibliothèques *open source* puissantes, comme par exemple SQLite pour les bases de données et OpenGL pour la gestion d'images 2D et 3D (pour faire des jeux).

2.2. Gratuit (ou presque)

Android est gratuit, autant pour nous que pour les constructeurs. En revanche, pour poster nos applications sur le Play Store, il nous en coûtera la modique somme de 25$. Ces 25$ permettent de publier autant d'applications que nous le souhaitons.

2.3. Facile à développer

Toutes les API mises à disposition facilitent et accélèrent grandement le travail. Ces APIs sont très complètes et très faciles d'accès. De manière un peu caricaturale, on peut dire que vous pouvez envoyer un SMS en seulement deux lignes de code (concrètement, il y a un peu d'enrobage autour de ce code, mais pas tellement).

Une API, ou « interface de programmation » en français, est un ensemble de règles à suivre pour pouvoir dialoguer avec d'autres applications.

2.4. Facile à vendre

Le *Play Store* (anciennement *Android Market*) est une plateforme immense et très visitée ; c'est donc une mine d'opportunités pour quiconque veut diffuser une application dessus.

2.5. Flexible

Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes. Les smartphones, les tablettes, la présence ou l'absence de clavier ou de *trackball*, différents processeurs... Non seulement c'est une immense chance d'avoir autant d'opportunités, mais en plus Android est construit de manière à faciliter le développement et la distribution en fonction des composants en présence dans le terminal (si votre application nécessite d'utiliser le Bluetooth, seuls les terminaux équipés de Bluetooth pourront la voir sur le Play Store).

2.6. Complémentaire

L'architecture d'Android est inspirée par les applications composites, et encourage par ailleurs leur développement. Ces applications se trouvent essentiellement sur internet et leur principe est que vous pouvez combiner plusieurs composants totalement différents pour obtenir un

résultat surpuissant. Par exemple, si on combine l'appareil photo avec le GPS, on peut poster les coordonnées GPS des photos prises.

3. Différents support[3]

3.1. Smartphones

Un smartphone (téléphone évolué) est un téléphone qui possède des fonctions similaires à celles des assistants personnels. Certains peuvent lire des vidéos, des MP3 et se voir ajouter des programmes spécifiques.

Le premier mobile commercialisé sous Android est le HTC G1/Dream produit par la firme Taïwanaise HTC, lancé aux États-Unis sur le réseau T-Mobile le 22 octobre 2008.

Le 5 janvier 2010 Google annonce le Nexus One, téléphone conçu par la firme de Mountain View et sous-traité à HTC. Doté de caractéristiques alors assez impressionnantes (écran AMOLED de 3,7 pouces, processeur de 1 GHz, 512 Mo de mémoire RAM et Android 2.1.

En décembre 2010, Samsung fabrique le Nexus S pensé par Google sous Android 2.3, et le mois de sa sortie celui-ci est envoyé à une altitude de 32 000 mètres. Google annonce passer les 300 000 activations de smartphones par jour.

Le 18 octobre 2011, Samsung et Google dévoilent le Galaxy Nexus, premier smartphone sous Android 4 « Ice Cream Sandwich ». Celui-ci intègre le déverrouillage par reconnaissance faciale, l'utilisation de boutons virtuels et un système de reconnaissance vocale avancé.

En 2012, Google sort, en partenariat avec LG le Nexus 4, un smartphone haut de gamme qui est le tout premier à bénéficier de la version 4.2 d'Android. Le 31 octobre 2013, Google sort le LG Nexus 5 équipé d'Android 4.4 (KitKat) et supportant la 4G. En 2014, Google sort, en partenariat avec Motorola le Nexus 6 qui est le premier à avoir la version Android 5 Lollipop.

3.2. Tablettes

Une tablette tactile, tablette électronique, ardoise électronique, tablette numérique, ou tout simplement tablette, est un ordinateur portable ultraplat qui se présente sous la forme d'un écran tactile sans clavier et qui offre à peu près les mêmes fonctionnalités qu'un ordinateur personnel. Elle permet d'accéder à des contenus multimédias tels que la télévision, la navigation sur le web, la consultation et l'envoi de courrier électronique, l'agenda, le calendrier et la bureautique simple. Il est possible d'installer des applications supplémentaires depuis une boutique d'applications en ligne. En quelque sorte, la tablette tactile est un intermédiaire entre l'ordinateur portable et le smartphone.

En septembre 2010, Samsung présente à l'IFA de Berlin le Samsung Galaxy Tab, tournant sous Android 2.2 (FroYo) et sortie fin 2010. Archos avec sa 7^{e} génération de tablettes internet introduit Android (lancée en septembre 2009). Dans la même lignée, les tablettes Archos de la génération 8 (Gen 8) intègrent Android 2.2 (FroYo).

En juillet 2012, Google lance un partenariat avec Asus pour fabriquer la Nexus 7 , tablette multimédia low-cost et première tablette de Google. Trois mois plus tard, la Nexus 10, tablette 10 pouces fabriquée par Samsung, est présentée en même temps que le Nexus 4, fabriqué par LG.

Elle propose une définition d'écran record pour une tablette (2560 x 1600 pixels) ainsi qu'un processeur double-cœur Exynos fabriqué par Samsung, cadencé à 1,7 GHz.

En juillet 2013, Google a présenté, en même temps que *Android* 4.3 « Jelly Bean », la version 2013 de la Nexus 7, conçue une nouvelle fois par Asus. En octobre 2014, Google a présenté en même temps que la version Android 5 Lollipop le Nexus 6 et la tablette Nexus 9 conçue par HTC.

4. Architecture et plateforme

4.1. Architecture du système

Dans le Guide du développeur, Android est défini comme étant une *pile de logiciels*, c'est-à-dire un ensemble de logiciels destinés à fournir une solution clé en main pour les appareils mobiles smartphones et tablettes tactiles. Cette pile comporte un système d'exploitation (comprenant un noyau Linux), les applications clés telles que le navigateur web, le téléphone et le carnet d'adresses ainsi que des logiciels intermédiaires entre le système d'exploitation et les applications. L'ensemble est organisé en cinq couches distinctes:

- le noyau Linux avec les pilotes ;
- des bibliothèques logicielles telles que WebKit, OpenGL, SQLite ou FreeType ;
- un environnement d'exécution et des bibliothèques permettant d'exécuter des programmes prévus pour la plate-forme Java ;
- un framework - kit de développement d'applications ;
- un lot d'applications standard parmi lesquelles il y a un environnement de bureau, un carnet d'adresses, un navigateur web et un téléphone.

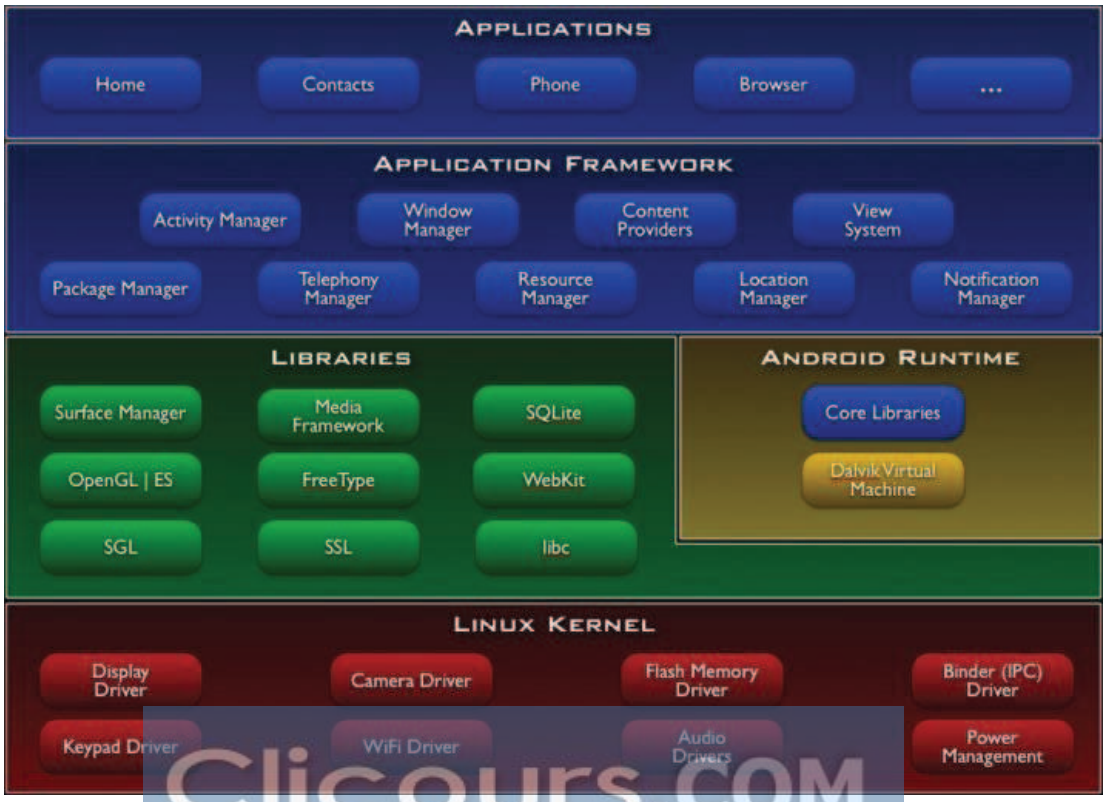


Figure 1 : Organisation des couches logiciel d'Android [5][6]

Les services offerts par Android facilitent notamment l'exploitation des réseaux de télécommunications GSM, Bluetooth, Wi-Fi et UMTS, la manipulation de médias, notamment de la vidéo H.264, de l'audio MP3 et des images JPEG ainsi que d'autres formats, l'exploitation des senseurs tels que les capteurs de mouvements, la caméra, la boussole et le récepteur GPS, l'utilisation de l'écran tactile, le stockage en base de données, le rendu d'images en 2D ou 3D en utilisant le processeur graphique, l'affichage de page web, l'exécution multitâche des applications et l'envoi de messages SMS.

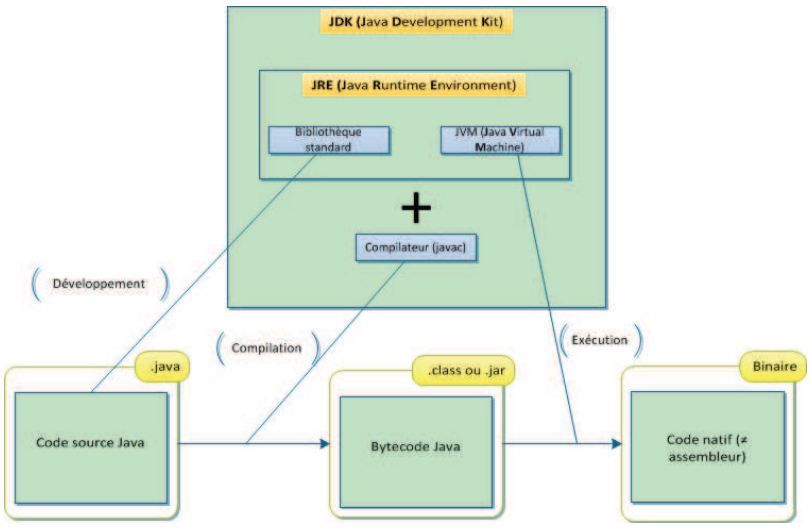
Android est distribué en open source sous licence Apache. La licence autorise les constructeurs qui intègrent Android dans leurs appareils à y apporter des modifications

Le noyau Linux est utilisé pour les fondations d'Android, les services classiques des systèmes d'exploitation : utilisation des périphériques, accès aux réseaux de télécommunication, manipulation de la mémoire et des processus et contrôle d'accès. Il s'agit d'une branche du noyau Linux, modifiée en vue de son utilisation sur des appareils mobiles. Le X Window System, les outils de GNU, ainsi que certains fichiers de configuration qui se trouvent d'ordinaire dans les distributions Linux ne sont pas inclus dans Android. L'équipe de développement d'Android a apporté de nombreuses améliorations au noyau Linux, et la décision a été prise par la communauté de développement de Linux d'incorporer ces améliorations dans le noyau Linux 3.3.

4.2. Android et la plateforme Java

Jusqu'à sa version 4.4, Android comporte une machine virtuelle nommée Dalvik, qui permet d'exécuter des programmes prévus pour la plate-forme Java. C'est une machine virtuelle conçue dès le départ pour les appareils mobiles et leurs ressources réduites - peu de puissance de calcul et peu de mémoire. En effet les appareils mobiles contemporains de 2011 ont la puissance de calcul d'un ordinateur personnel vieux de dix ans. La majorité, voire la totalité des applications est exécutée par la machine virtuelle Dalvik.

Le bytecode de Dalvik est différent de celui de la machine virtuelle Java de Oracle (JVM), et le processus de construction d'une application est différent : le code source de l'application, en langage Java est tout d'abord compilé avec un compilateur standard qui produit du bytecode pour JVM (bytecode standard de la plateforme Java) puis ce dernier est traduit en bytecode pour Dalvik par un programme inclus dans Android, du bytecode qui pourra alors être exécuté. [3]



Organigramme 2 : Processus d'exécution d'un code source Java sous Android

L'ensemble de la bibliothèque standard d'Android ressemble à J2SE (*Java Standard Edition*) de la plateforme Java. La principale différence est que les bibliothèques d'interface graphique AWT et Swing sont remplacées par des bibliothèques d'Android.

Le développement d'applications pour Android s'effectue avec un ordinateur personnel sous Mac OS, Windows ou Linux en utilisant le JDK de la plate-forme Java et des outils pour Android. Des outils qui permettent de manipuler le téléphone ou la tablette, de la simuler par une machine virtuelle, de créer des fichiers APK (les fichiers de paquet d'Android), de déboguer les applications et d'y ajouter une signature numérique. Ces outils sont mis à disposition sous la forme d'un plugin pour l'environnement de développement Eclipse et plus récemment avec Android Studio fourni par Intellij Platform.

La bibliothèque d'Android permet la création d'interfaces graphiques selon un procédé similaire aux frameworks de quatrième génération que sont XUL, JavaFX ou Silverlight: l'interface graphique peut être construite par déclaration et peut être utilisée avec plusieurs *skins* - chartes graphiques. La programmation consiste à déclarer la composition de l'interface dans des fichiers *XML* ; la description peut comporter des *ressources* (des textes et des pictogrammes). Ces déclarations sont ensuite transformées en objets tels que des fenêtres et des boutons, qui peuvent être manipulés par de la programmation Java. Les écrans ou les fenêtres (*activités* dans le jargon d'Android), sont remplis de plusieurs *vues* ; chaque vue étant une pièce d'interface graphique (bouton, liste, case à cocher…). Android 3.0, destiné aux tablettes, introduit la notion de *fragments*: des panneaux contenant plusieurs éléments visuels. Une tablette ayant - contrairement à un téléphone - généralement suffisamment de place à l'écran pour plusieurs panneaux.

4.3. Android Runtime (ART)

A partir de la version 5.0 sortie en 2014, l'environnement d'exécution ART (Android RunTime) remplace la machine virtuelle Dalvik. Cet environnement d'exécution plus performant a été développé par Google pour pallier le potentiel limité de Dalvik, créé en 2007. Avec ART, contrairement à la machine virtuelle java, les fichiers .apk ne sont plus lancés directement, mais décompressés et lancés avec de nouvelles bibliothèques et API ; les applications prennent ainsi plus de place (+20 %), mais les gains en performance et en autonomie des batteries sont conséquents (+20 à 30 %).[3]

5. Evolution des versions du système

Code name	Version number	Initial release date	API level
	1.0	September 23, 2008	1
	1.1	February 9, 2009	2
Cupcake	1.5	April 27, 2009	3
Donut	1.6	September 15, 2009	4
Eclair	2.0–2.1	October 26, 2009	5–7
Froyo	2.2–2.2.3	May 20, 2010	8
Gingerbread	2.3–2.3.7	December 6, 2010	9–10
Honeycomb[a]	3.0–3.2.6	February 22, 2011	11–13
Ice Cream Sandwich	4.0–4.0.4	October 18, 2011	14–15
Jelly Bean	4.1–4.3.1	July 9, 2012	16–18
KitKat	4.4–4.4.4, 4.4W–4.4W.2	October 31, 2013	19–20
Lollipop	5.0–5.1.1	November 12, 2014	21–22
Marshmallow	6.0–6.0.1	October 5, 2015	23
N	*Developer Preview 1*		

Tableau 2 : Historique des versions du système Android [3]

Partie II

Les procédés de calcul topométriques

CHAPITRE 3

GENERALITES SUR LES TRAVAUX TOPOGRAPHIQUES DE TERRAIN

1. Préambule

La Topographie, dans son sens le plus général, est une science très vaste qui a pour objet tout ce qui concerne l'établissement des plans et cartes ainsi que leur utilisation. Une carte représente une surface très étendue généralement à une échelle très petite. Un plan représente une surface plus restreinte et à une échelle très grande, l'échelle étant le rapport de similitude de la figure du plan à la figure de terrain.

La topométrie est l'ensemble des opérations effectuées, principalement sur le terrain, pour la détermination métrique des éléments d'une carte ou d'un plan. La topométrie est aussi la technique qui permet de maîtriser la mesure et la représentation en 2 ou 3 dimensions d'éléments localisés du monde réel.

Le lever de détails est l'ensemble des opérations intervenant dans un lever topographique et consistant à déterminer à partir des points du canevas d'ensemble , polygonal ou de détails, la position des différents objets d'origine naturelle ou artificielle existant sur le terrain. Le levé, nom donné au document résultant d'un lever, est destiné, éventuellement après traitement numérique, à l'établissement de plans graphiques ou numériques : c'est la phase de report.

Un canevas est un ensemble discret de points judicieusement répartis sur la surface à lever, dont les positions relatives sont déterminées avec une précision au moins égale à celle que l'opérateur attend du levé. Ces points servent d'appui au lever des détails, implantations, etc. Le canevas s'exprime par les coordonnées de ces points dans un même système. Afin de décrire le terrain, on dispose de tout un panel de techniques et méthodes. Chaque lever de détails doit s'accompagner d'un croquis de levé aussi précis, soigné et descriptif que possible. Ce croquis est d'une aide précieuse, voire indispensable, lors de l'établissement du plan définitif.

2. La collecte de données sur terrain

2.1. Méthode conventionnelle avec station totale

La station totale est l'instrument idéal pour le lever précis d'un grand nombre de points. L'appareil permet de mesurer et d'enregistrer distances et angles en une seule manipulation. Ces données peuvent être enregistrées sur un support informatique en vue d'un traitement par ordinateur.

2.1.1. Mise en station

Il faut tout d'abord mettre en station l'appareil, c'est-à-dire qu'il faut le positionner de manière que l'axe vertical de l'appareil soit perpendiculaire au plan horizontal de la station. On oriente ensuite l'appareil, qui peut se faire de trois manières :

- par un repère local pour un lever non rattaché : l'opérateur vise un point de référence et cale l'angle horizontal de l'appareil à 0 sur ce point (ou bien on effectue la lecture angulaire sur ce point)
- par un repère général pour un lever rattaché : l'opérateur vise un point connu et affiche sur ce point le gisement de la direction station-point de manière à travailler directement dans le repère général ;

- au moyen d'une station libre, très pratique puisqu'il est possible de stationner un point quelconque choisi à l'endroit le plus propice pour observer un maximum de points par station.

La mise en station initiale doit souvent être épaulée par deux ou trois stations de recoupement au départ desquelles l'instrument de mesure va viser les mêmes points pour confirmer leurs coordonnées par triangulation.

Les mises en station sur un point connu doivent intégrer aussi les déformations curvilinéaires (la terre est un géoïde et sa représentation en plan est en deux dimensions). Des distorsions apparaissent déjà lorsque le point de station est dans un rayon de 300 m des points du semis. Il va alors falloir déplacer l'appareil de mesure pour pouvoir couvrir tout le terrain de l'étude.

2.1.2. Déroulement du levé

Il est ensuite possible de relever tous les points caractéristiques du terrain après la mise en station (il convient de connaître parfaitement la précision de l'appareil suivant la documentation fournie par les constructeurs, notamment celle concernant la mesure des angles horizontaux et verticaux). Lors du déroulement du lever, le porte-miroir y dirige les opérations. Le porte-miroir choisit les points à lever et l'ordre dans lequel il les stationne : cela est fonction de la codification des points et doit être pensé sur le terrain en vue d'un gain de temps lors de la phase de report. Pour des raisons de visibilité, il peut être ponctuellement nécessaire de modifier la hauteur de voyant. Une pratique courante est d'utiliser toujours la même hauteur de voyant qui devient la hauteur par défaut (par exemple 1,60 m) et d'utiliser, en cas de problèmes de visibilité, des hauteurs standard (1,2 m et 2 m) : cela peut permettre de lever certains doutes ou de remédier à des oublis...

Il peut faire un croquis au fur et à mesure du lever. Dans un souci de gain de temps, il est préférable qu'une troisième personne effectue ce croquis. À défaut, le porte-miroir peut préparer un croquis du terrain pendant les temps de déplacement de station et de mise en station ; l'opérateur reportera alors sur ce croquis les numéros des points levés. L'opérateur installé derrière la station totale vise à chaque point le centre du miroir et déclenche la mesure.

En entrant des codes pour les points de détail levés, certains tracés peuvent être automatisés au moment de la phase d'habillage du levé sur ordinateur. Par exemple, un même code (numéro) sera associé à tous les points levés en crête de talus et un autre code à ceux levés en pied de talus. Le logiciel traitant les données doit être programmé pour reconnaître ces codes et dessiner lui-même les talus.

Quatre méthodes principales sont utilisées en planimétrie. Il est possible de déterminer l'emplacement d'un point sur un plan horizontal:

- à partir d'un seul point connu, par cheminement, méthode consistant à mesurer des distances horizontales et des azimuts le long d'une ligne brisée ;
- à partir d'un seul point connu, par rayonnement, méthode consistant à mesurer des distances horizontales et des angles horizontaux;
- à partir de deux points connus, par triangulation et/ou intersection, méthodes consistant à mesurer des distances horizontales et des azimuts, ou encore des angles

Les démarches de calculs de chacune de ces méthodes seront décrites dans le chapitre suivant.

2.2. Méthode GNSS

2.2.1. Description succincte du système GNSS

Un GNSS est un système de géolocalisation fonctionnant au niveau mondial et reposant sur l'exploitation de signaux radio émis par des satellites dédiés. Ce genre de système est mis en place par le département de la Défense des États-Unis à des fins militaires au tout début, notamment avec le système GPS. Il est très rapidement apparu que des signaux transmis par les satellites pouvaient être librement reçus et exploités, et qu'ainsi un récepteur pouvait connaître sa position sur la surface de la Terre, avec une précision sans précédent, dès l'instant qu'il était équipé des circuits électroniques et du logiciel nécessaires au traitement des informations reçues. Une personne munie de ce récepteur peut ainsi se localiser et s'orienter sur terre, sur mer, dans l'air ou dans l'espace au voisinage proche de la Terre.

Le GNSS présente de nombreux avantages par rapport aux méthodes topographiques conventionnelles :

- L'intervisibilité entre points n'est pas requise.
- Il peut être utilisé à toute heure du jour ou de la nuit.
- Il fournit des résultats de précision.
- Il permet de traiter un volume de travail plus important en moins de temps et avec moins de personnel.

Cependant, pour utiliser le GPS, il faut que l'antenne GPS puisse capter les signaux émis par au moins 4 satellites. Des obstacles tels que des bâtiments élevés ou des arbres peuvent entraver la réception de ces signaux. Le GPS est donc inopérant en intérieur. Il est également difficile d'utiliser le GPS en centre ville ou dans des zones boisées. Du fait de ces limitations, il peut être plus rentable, pour certains types d'applications, de mettre en œuvre une station totale optique voire de combiner un instrument de ce type au GPS.

2.2.2. Technique de mesure GNSS

Il existe plusieurs techniques de mesure pouvant être utilisées par la plupart des capteurs GPS pour la topographie. Il incombe au géomètre de choisir la technique adéquate pour son application.

2.2.2.1. Positionnement absolu

Le type de positionnement dont il a été question jusqu'à présent était effectué à l'aide d'un seul récepteur. Ce type de positionnement se nomme positionnement absolu, puisque seules les observations recueillies par un récepteur contribuent à la détermination de sa position. La précision théorique du positionnement absolu est maintenant d'environ 3 m pour le GPS par exemple, depuis que l'armée américaine n'introduit plus volontairement d'erreurs dans les éphémérides ni de variation dans la fréquence nominale des horloges des satellites. Cet autre dispositif de sécurité se nommait la disponibilité sélective(SA: *Selective Availability*). Il avait pour but de restreindre l'accès au plein potentiel du GPS.

2.2.2.2. Positionnement relatif

On observe les mêmes satellites en même temps sur les deux stations et le calcul se fait sur les différences des mesures. Dans ce cas, on détermine les composantes du vecteur compris entre une station connue dans le système WGS84 et une station inconnue Le vecteur entre les deux points est appelé ligne de base dans la littérature GPS.

La simple différence consiste à former à un instant donné la différence de mesures entre un satellite et deux récepteurs. Il faut disposer de deux récepteurs et faire les mesures aux mêmes époques. L'estimation ne portera plus sur les coordonnées d'un point mais sur le vecteur entre deux points, c'est du positionnement relatif, ce qui implique de connaître un point en WGS84.

Généralement, c'est ce type de positionnement qui est le plus exploité dans la topographie.

3. Détermination de l'incertitude de mesure [7]

Une mesure est entachée d'une certaine erreur, d'une incertitude. Elle provient de divers facteurs : la méthode utilisée, l'instrument employé, l'expérience de l'opérateur, la grandeur mesurée… Différentes notions sont utilisées pour qualifier la qualité de la mesure, et divers moyens existent pour répartir les résidus d'une série de mesure.

3.1. Erreurs et fautes

La faute : manquement à une norme, aux règles d'une science, d'une technique (Petit Larousse). On parle de faute généralement à propos de l'opérateur, et peut être due à un manque de soin, le non respect des règles de base, le manque d'expérience…

L'erreur systématique : se répète et se cumule à chaque mesure. Elle est le plus souvent due aux imprécisions de l'instrument (qualité des composants, défauts de réglages…) et aux contraintes de sa mise œuvre. L'influence de ces erreurs peut souvent être évaluée par calcul, et prise en compte dans la détermination finale.

L'erreur accidentelle : de valeur et de signe aléatoires, elle peut avoir diverses origines : défaut de calage de l'appareil à la mise en station, erreur de pointé, de lecture, des paramètres extérieurs non maîtrisables (température, hygrométrie…), erreur de réfraction accidentelle…

Sur une série de mesures (cheminement altimétrique, polygonal), l'influence des erreurs systématiques doit être minimisée par la méthode employée. Par contre, il reste les erreurs accidentelles qui sont généralement considérées comme les seules participant aux fermetures.

3.2. Méthodes de compensation

Tout protocole de mesure génère des erreurs. Il est capital d'identifier, quantifier et réduire les erreurs systématiques, mais les erreurs accidentelles doivent être réparties sur l'ensemble. Plusieurs méthodes sont possibles, mais partent toutes globalement de l'hypothèse de l'équiprobabilité de chaque source d'erreur accidentelle lors de chaque mesure. Par exemple, sur un cheminement altimétrique, la probabilité de faire une erreur de lecture sur mire est identique qu'il s'agisse de la première ou de la nième dénivelée.

3.2.1. Compensation proportionnelle

C'est le mode de compensation le plus simple. Il exploite l'hypothèse d'équiprobabilité au mot : l'erreur globale constatée sur la série de mesures est la résultante des erreurs sur chaque mesure de la série. Par conséquent, la fermeture est répartie sur chaque mesure individuelle. Pour une fermeture f obtenue sur n mesures, la correction à appliquer aux observations est alors donnée par :

$$\boldsymbol{c = -\frac{f}{n}}$$

Elle peut s'avérer tout à fait suffisante pour la répartition de la fermeture d'un nivellement géométrique à portées strictement équidistantes et équivalentes.

3.2.2. Compensation pondérée

La compensation pondérée est une amélioration de la compensation proportionnelle. Elle prend en compte, par la pondération des observations, une certaine appréciation de la qualité des mesures. Tout le problème est alors de déterminer le facteur significatif agissant sur cette qualité. De même que précédemment, la correction à appliquer à la j-ième observation sur n, de facteur de pondération p, est donnée par :

$$\boldsymbol{c^j = -f\frac{p^j}{\sum_{i=1}^{n} p_i^j}}$$

Dans le cas d'un tour d'horizon, on pourra prendre la distance au point comme facteur de pondération. En effet, en triangulation, le pointé sur des cibles lointaines est souvent bien plus précis que sur des cibles proches.

3.2.3. Compensation par les moindres carrés

Les méthodes précédentes s'appliquent dans les cas simples, où les mesures redondantes ne sont que peu ou pas présentes. Dès lors que l'on s'intéresse à un réseau de mesures, engendrant des déterminations multiples d'une même grandeur, il est impératif de pouvoir tirer parti de l'ensemble des observations sans créer de discordances entre elles.

Le principe des moindres carrés a pour objectif de minimiser les carrés des écarts entre les observations et la valeur vraie de la grandeur observée. Elle se base exclusivement sur la redondance de mesures. Un calcul abouti par moindres carrés donne accès à la valeur la plus probable de la grandeur mesurée, avec un indicateur de qualité primordial : l'erreur moyenne quadratique (souvent notée emq ; en anglais, rmse, *root mean square error*).

La complexité de la méthode ne nous permet pas de la présenter dans le détail. Nous nous limiterons par conséquent à une expression simplifiée, matricielle. La première étape est de définir des valeurs approchées des inconnues, pour pouvoir écrire la matrice V des écarts avec chaque mesure. Ensuite, l'équation suivante donne les appoints à apporter aux valeurs approchées pour obtenir les valeurs les plus probables, compte tenu des observations réalisées.

$$\boldsymbol{A = {}^tV.P.V}$$

Ainsi, l'erreur moyenne quadratique du calcul (mq_0), également dite réduite à l'unité de poids, est donné par la relation :

$$mq_0 = \pm\sqrt{\frac{\sum_{i=1}^{n}(p_i v_i^2)}{n-q}}$$

avec p le poids de l'observation, v l'écart entre valeurs approchée et observée, n le nombre total d'observations, q le nombre d'observations strictement nécessaires au calcul de l'inconnue.

3.3. Sources d'erreurs du GPS

3.3.1. Les délais de propagation dans l'ionosphère et l'atmosphère

Le signal du satellite peut être ralenti durant sa traversée de l'ionosphère, l'effet étant similaire à la réfraction de la lumière à travers un bloc de verre. Ces délais de propagation dans l'atmosphère peuvent entraîner une erreur dans le calcul de la distance puisque la vitesse du signal en est affectée (la vitesse de la lumière n'est constante que dans le vide).

3.3.2. Erreurs dues aux horloges du satellite et du capteur

Bien que les horloges embarquées à bord des satellites soient extrêmement précises (à environ 3 nanosecondes près), elles peuvent être sujettes à de légères dérives et entraîner des erreurs affectant la précision de la position. Le ministère de la défense des Etats-Unis surveille les horloges des satellites par l'intermédiaire du secteur de contrôle de manière à corriger toute dérive constatée.

3.3.3. Erreurs dues à des trajets multiples

Les trajets multiples surviennent lorsque le capteur se trouve à proximité d'une surface réfléchissante de grande dimension telle qu'une étendue d'eau ou un bâtiment. Le signal du satellite ne se propage pas en ligne directe vers l'antenne, il atteint d'abord l'objet proche du capteur et est ensuite réfléchi vers l'antenne faussant ainsi la mesure. L'utilisation d'antennes spéciales intégrant un plan de masse (un disque métallique d'un diamètre d'environ 50 centimètres) permet de réduire les trajets multiples en empêchant les signaux réfléchis à faible hauteur d'atteindre l'antenne.

3.3.4. Coefficient d'affaiblissement de la précision du résultat

Le coefficient d'affaiblissement de la précision du résultat (Dilution of Precision, DOP) permet d'apprécier la qualité de la géométrie des satellites et se réfère à la répartition dans l'espace des satellites utilisés pour déterminer une position. Le DOP peut amplifier les effets dus aux erreurs dans les distances vers les satellites.

4. Les nouvelles technologies au service de l'amélioration des travaux de terrain

Nous avons vu dans les paragraphes précédents que les appareils de collectes de données deviennent de plus en plus complexes, et mettent à profit les innovations technologiques actuelles. Le report sur papier tend à disparaitre puisque les données collectées sont immédiatement stocker dans les appareils de mesures pour ensuite être traitées au bureau ultérieurement.

Cependant, il se peut que le géomètre ait besoin immédiatement de visualiser les résultats de ses observations pour pouvoir adopter sa méthode de travail ou alors de contrôler celle-ci. Les stations totales modernes sont accompagnés en options d'un carnet électronique qui permet de visualiser les observations faites.

Ce type de matériel n'est pas encore très répandu dans notre pays, puisque la plupart des géomètres ont recours à la location lors des travaux fonciers ou alors utilisent le matériel mis à leur disposition s'ils travaillent en entreprise. Les données sont souvent alors stocker dans des carnets de plusieurs page retranscrit à la main, que l'on va ressaisir dans un logiciel de traçage comme AutoCAD une fois au bureau. Les risques d'erreurs de transcription ou alors l'illisibilité du à l'altération du papier soumis au milieu extérieur sont à craindre. Le carnet de terrain dédié n'est pas encore très répandu chez le géomètre malgache vu leur (environ $ 4000 pour le Trimble TSC3 Controller. S6 sur E-Bay).

Figure 2 : Carnet de terrain électronique Trimble sur E-Bay

Pour aider à résoudre ce problème, par soucis de modernité et en vue d'améliorer les conditions de travail de chaque géomètre de notre pays, nous avons décider d'élaborer une application mobile pour smartphones et tablettes fonctionnant sous le système Android, décrit dans les chapitres précédent, afin d'assister le topographe débutant autant qu'expert dans ces collectes de données.

L'application en question permet de collecter les données observées sur le terrain et les stockent dans la mémoire du téléphone. Ces données peuvent être extraites sous formes de fichier numériques depuis l'application pour être immédiatement traité dans un logiciel de traçage et passer ainsi l'étape de la saisie, ou alors être visualiser directement depuis l'application.

CHAPITRE 4

ALGORITHMES DE CALCUL DES METHODES DE DENSIFICATION

Ce chapitre traite principalement des algorithmes de calculs utilisés dans notre application pour déterminer les coordonnées de points.

1. Gisement et rayonnement [8][11]

On définit le gisement comme l'angle, dans le plan horizontal, entre un vecteur, défini par deux points connus en coordonnées, et la direction du nord cartographique. Il est compté dans le sens horaire positivement de 0 à 400 grades. Pour un direction AB, on le note G_{AB} (ou aussi V_{AB}).

Pour $A(X_A,Y_A)$ et $B(X_B,Y_B)$ où X et Y expriment les coordonnées d'un point dans un système de projection défini, la relation suivante permet de calculer la valeur de G_{AB}

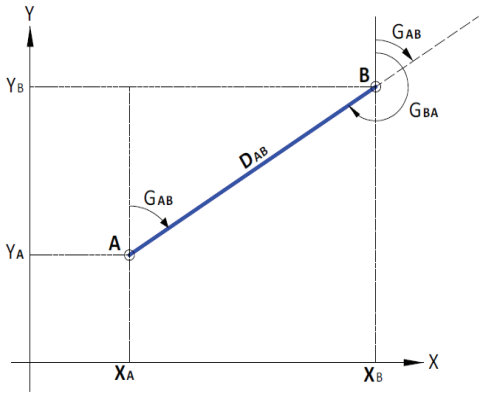


Figure 3 : Gisement d'une direction AB

$$\tan G_{AB} = \frac{X_B - X_A}{Y_B - Y_A} \qquad (1)$$

Lors de la mise en station d'un théodolite sur un point S connu en coordonnées, la position du zéro du limbe est au départ quelconque. Les lectures d'angles horizontaux sont comptées positivement dans le sens des aiguilles d'une montre par rapport à une direction origine correspondant à la lecture « zéro ».

Le gisement d'une direction peut se déduire du gisement de l'origine des lectures d'angles horizontaux mesurées lors du tour d'horizon. Celui ci appelé G0 d'orientation peut se calculer à partir de l'observation de points connus en coordonnées.

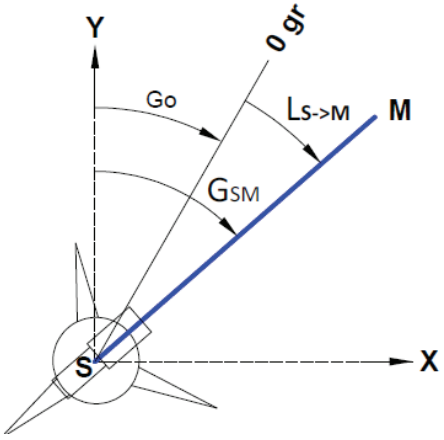


Figure 4 : G0 de la station S

$$G_{SP} = G0 + L_{Sp} \quad (2)$$
$$G0 = G_{SP} - L_{Sp} \quad (3)$$

Pour améliorer la précision de l'orientation de la station, plusieurs lectures sur des points connus en coordonnées sont déterminées. Pour obtenir une orientation correcte, il faut au minimum deux visées (trois ou quatre sont préférables) réparties sur les quatre quadrants autour du point de station S. On obtient alors le G0 moyen de la station.

En topographie, il est très fréquent de connaître un point S et de chercher les coordonnées d'un point P visible depuis S. On dit que P est rayonné depuis S si l'on peut mesurer la distance horizontale D_{SP} et le gisement G_{SP} .Quel que soit le quadrant, on peut alors calculer les coordonnées du point P par les formules suivantes :

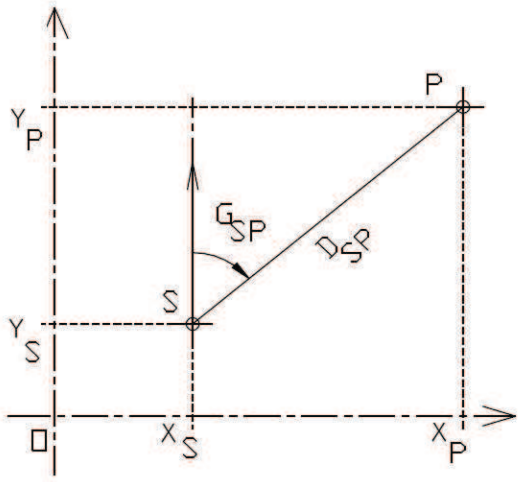


Figure 5 : Rayonnement de S à P

$$X_P = X_S + D_{SP} . sinG_{SP} \quad (4)$$
$$Y_P = Y_S + D_{SP} . cosG_{SP} \quad (5)$$

A défaut de mesurer directement G_{SP}, on mesure un angle α avec une direction dont le gisement est connu ou bien on calcule un *G0 moyen* de station.

2. Cheminements planimétriques [9]

2.1. Type de cheminement

Le cheminement est un procédé de topographie très courant dans lequel on parcourt des droites afin d'en faire le levé planimétrique. Il est particulièrement bien adapté aux terrains plats ou boisés.

Il existe deux types de cheminement. Si le cheminement suivi forme une figure fermée, tel que le périmètre délimitant une propriété, il s'agit d'un cheminement fermé. Si le cheminement forme une ligne comportant un point de départ et un point d'arrivée, tel que l'axe d'un canal d'alimentation d'eau, il s'agit d'un cheminement encadré.

Un cheminement ni fermé ni encadré est une antenne.

2.2. Procédés de calculs

Pour effectuer un levé par cheminement, il faut réaliser des *mesures* afin de connaître:

- la distance entre les stations de cheminement;
- l'orientation de chaque section de cheminement.

2.2.1. Mesure des angles de gauches

C'est l'angle que l'on trouve à sa gauche dans le sens de calcul, ce sens de calcul étant celui dans lequel on parcourt les sommets lors du calcul : il peut être différent du sens de parcours sur le terrain bien qu'il soit préférable de conserver le même.

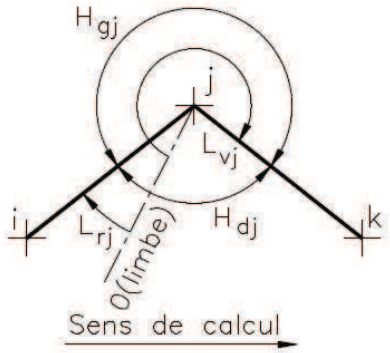


Figure 6 : Angle de gauche

En station au sommet *j*, on note :

- Lr_j la lecture arrière au sommet *j* sur le sommet précédent ;
- Lv_j la lecture avant au sommet *j* sur le sommet suivant ;
- Hg_j l'angle topographique de gauche (ou angle à gauche) dans le sens de calcul

On déduit donc la relation : $\boldsymbol{Hg_j = Lv_j - Lr_j}$ $\quad (6)$

2.2.2. Transmission de gisement

L'orientation du premier coté du cheminement est calculée à partir de visées d'orientation sur d'autres points connus, la transmission de cette orientation s'effectue à l'aide de l'angle gauche observé à chaque sommet. Deux cas peuvent se présenter :

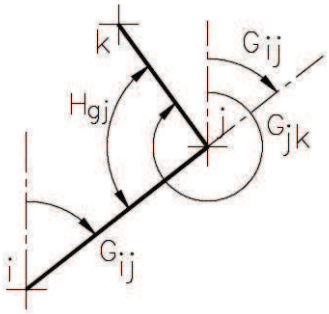


Figure 7 : Transmission de gisement

Au sommet *j* et à partir de l'angle de gauche, on peut écrire : $\boldsymbol{G_{jk} = G_{ij} + H_{gj} + 200}$ **(7)**

Si l'on considère la figure suivante, la formule devient : $\boldsymbol{G_{jk} = G_{ij} + H_{gj} - 200}$ **(8)**

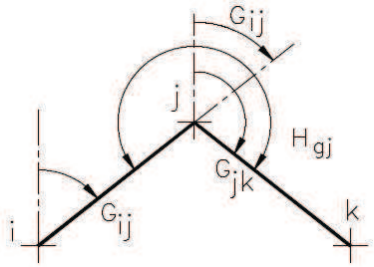


Figure 8 : Transmission de gisement

La formule générale est donc : $\boldsymbol{G_{jk} = G_{ij} + H_{gj} \pm 200}$ **(9)**

2.2.3. Fermeture et compensation angulaire

2.2.3.1. Fermeture d'un cheminement encadré

Considérons le cheminement suivant :

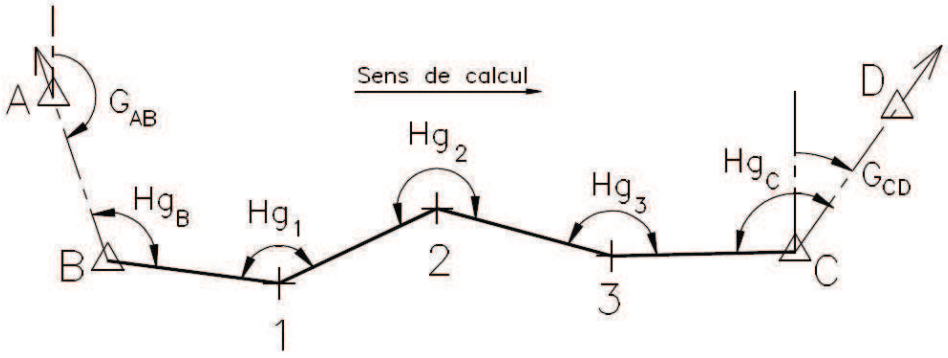


Figure 9 : Cheminement encadré

Les points A, B, C et D sont connu donc on peut calculer G_{AB} que l'on notera G_d, et G_{CD} noté G_f. On calcule de proche en proche tous les gisements de tous les côtés pour arriver au gisement d'arrivée G_{CD} connu qui sert de contrôle des erreurs de lecture d'angles.

On déduit alors :

$$\boldsymbol{G_{B1} = G_{AB} + H_{gB} - 200}$$
$$\boldsymbol{G_{1-2} = G_{B1} + H_{g1} - 200}$$
$$\boldsymbol{\ldots \quad \ldots \quad \ldots \quad \ldots}$$
$$\boldsymbol{G'_f = G_{3C} + H_{gC} - 200} \quad \textbf{(10)}$$

G'_f est le gisement d'arrivée observé (G'_{CD})

En additionnant membres à membres nous avons :

$$G_f^{'} = G_d + \sum(H_{gj}) - 200.(n+1) \quad (10)$$

Avec :
n le nombre de côtés du cheminement polygonal
$\sum(H_{gj})$ la somme de tous les angles de gauche

La fermeture angulaire dans le cas d'un cheminement encadré est la différence entre le gisement d'arrivée observé noté G'_f et le gisement d'arrivée calculé noté G_f
D'où on a la formule de la fermeture :

$$fa = G'_f - G_f \quad (11)$$

2.2.3.2. Fermeture d'un cheminement fermé

Soit le cheminement fermé suivant :

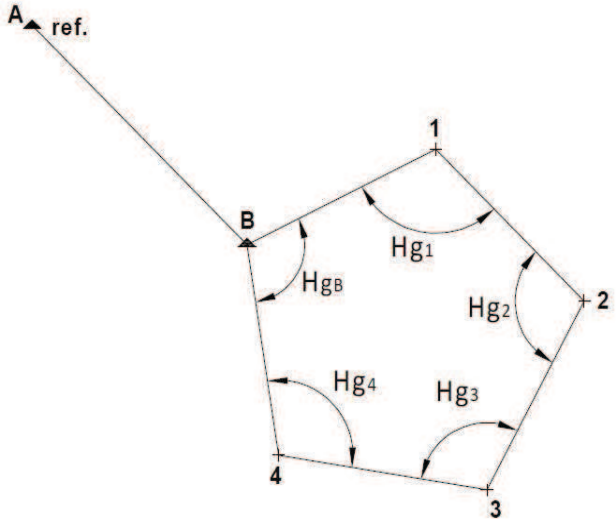


Figure 10 : Cheminement fermé

La somme théorique des angles internes d'un polygone à *n* côtés est donnée par la relation :

$$\sum Hg_j = (n-2).200 \quad (12)$$

Cette somme doit correspondre à la somme des angles gauches observés. La différences entre ces deux sommes donnent la fermeture en cheminement fermé, d'où : $fa = \sum Hg_{obs} - \sum Hg_j \quad (13)$

2.2.3.3. Tolérance sur la fermeture angulaire

L'écart type angulaire par station dépend de l'appareil de mesure utilisé; en la notant σa, on obtient les formules de tolérances , avec *n* côtés :

- Pour un cheminement fermé : $Ta = 2.7.\sigma a.\sqrt{n} \quad (14)$
- Pour un cheminement encadré : $Ta = 2.7.\sigma a.\sqrt{n+1} \quad (15)$

2.2.3.4. Compensation angulaire

La compensation consiste à répartir l'écart de fermeture angulaire sur tous les angles observés. La compenser angulairement d'un cheminement ne se fait que si l'écart de fermeture angulaire est inférieur à la tolérance réglementaire. Si ce n'est pas le cas, la manipulation doit être reprise en entier car il s'agit d'une faute.

La compensation angulaire est la quantité à répartir sur les différentes mesures ; c'est donc l'opposé de la fermeture angulaire :

$$\boldsymbol{Ca = -fa} \quad \textbf{(17)}$$

Dans le cas de notre application, on a opté pour la compensation pondérée qui sera efficace dans la majorité des cas. On répartit l'écart de fermeture *fa* en considérant que l'on commet plus l'erreur en angle sur une visée courte que sur une visée longue. Comme à chaque station intervient la distance de la visée arrière et celle de la visée avant (voir fig. 2.13.), on fait intervenir des poids p_j tels que, au sommet *j* :

$$\boldsymbol{p_j = \frac{1}{D_{j-1}} + \frac{1}{D_j}} \quad \textbf{(18)}$$

Où *Dj* et *Dj-1* sont exprimées en kilomètre.

La compensation angulaire *Cj* sur chaque lecture est alors :

$$\boldsymbol{C_j = \frac{Ca}{\sum p_j} . p_j} \quad \textbf{(19)}$$

2.2.4. Calcul de coordonnées et fermeture planimétrique

2.2.4.1. Calcul de fermeture

Une fois les distances et gisements entre chaque sommet du cheminement connu, on calcule leur coordonnées à partir du point de départ par rayonnements successifs.

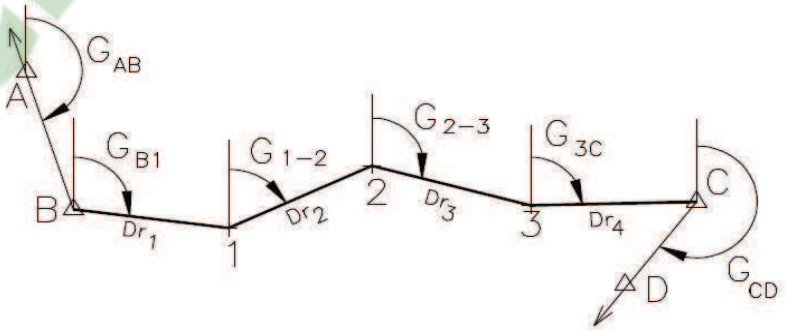


Figure 9: Cheminement encadré

On a alors pour X :

$$\boldsymbol{X_1 = X_B + D_{r1} . sinG_{B1}}$$

$$X_2 = X_1 + D_{r2} \,.\, sinG_{1-2}$$
$$\ldots \quad \ldots \quad \ldots \quad \ldots$$
$$X'_C = X_3 + D_{r4} \,.\, sinG_{3C} \qquad (20)$$

Il en va de même pour Y et en sommant membre à membre, on approche les coordonnées du point d'arrivée *C'* :

$$X'_C = X_B + \sum(D_{rj} \,.\, sinG_{ij}) \qquad (21)$$
$$Y'_C = Y_B + \sum(D_{rj} \,.\, cosG_{ij}) \qquad (22)$$

Cependant, on connait les coordonnées du point d'arrivée *C*. Il est donc possible de déduire l'erreur de fermeture planimétrique due aux erreurs de lectures et des erreurs de mesures de distances. On obtient les formules suivantes :

$$f_x = X'_C - X_C = X_B + \sum(\Delta X) - X_C \qquad (23)$$
$$f_y = Y'_C - Y_C = Y_B + \sum(\Delta Y) - Y_C \qquad (24)$$

A partir de ces fermetures, la fermeture planimétrique f_p qui représente la distance entre le point C' calculé et le point C réel peut être connu par la formule :

$$f_P = \sqrt{f_x^2 + f_y^2} \qquad (25)$$

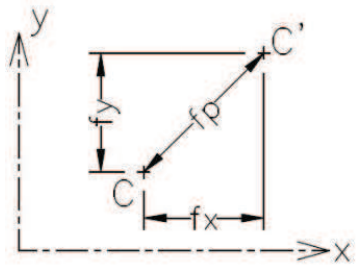


Figure 11 :Fermeture planimétrique

2.2.4.2. Tolérance, compensation et ajustement planimétrique

La tolérance de fermeture en longueur est : $T_L = 2.7\sigma_L\sqrt{n}$ .Les compensations C_x et C_y sur les ΔX et ΔY sont l'opposé respectivement des fermetures f_x et f_y .

L'ajustement planimétrique est le calcul qui consiste à répartir les fermetures planimétriques sur les mesures du cheminement. On choisit ici encore une fois la compensation pondérée où D_j représente la longueur de chaque côté :

$$C_{Xj} = \frac{C_X}{\sum_{i=1}^{i=n} D_i} . D_j \qquad (26) \qquad \text{et} \qquad C_{Yj} = \frac{C_Y}{\sum_{i=1}^{i=n} D_i} . D_j \qquad (27)$$

3. Recoupement [9]

3.1. Principe du recoupement

3.1.1. Intersection

Un point intersecté M est un point non stationné que l'on vise depuis des points d'appui connus en coordonnées de manière à déterminer les gisements des visées que l'on détermine les *G0* des points d'appui. Le point M se situe sur chaque demi-droite matérialisant chaque visée : ces demi-droites sont les lieux géométrique de M ; il se situe donc à leur intersection. Deux lieux sont donc nécessaires et suffisants pour déterminer le point M . Pour le contrôle, une visée supplémentaire est nécessaire et pour que le point M soit déterminé avec sécurité, il est conseillé d'effectuer une quatrième visée.

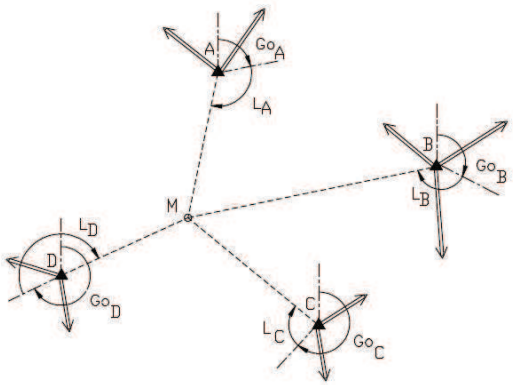


Figure 12 : Point par intersection

Ici on a : $\boldsymbol{G_{AM\,obs} = G0_A + L_A}$

$\boldsymbol{G_{BM\,obs} = G0_B + L_B}$

$\boldsymbol{G_{CM\,obs} = G0_C + L_C}$

$\boldsymbol{G_{DM\,obs} = G0_D + L_D}$

3.1.2. Relèvement

Un point relevé est un point stationné depuis lequel l'opérateur effectue un tour d'horizon sur des points anciens connus. Le relèvement est simple à réaliser sur le terrain puisqu'il ne nécessite qu'une seule station. La précision des visées angulaires étant meilleure pour des visées lointaines, c'est la méthode idéale pour de longues visées sans possibilité de mesure de distance. L'opérateur voit l'arc AB sous un angle α ; le point M se situe donc sur un arc de cercle passant par A, M et B : il est appelé arc capable AMB ; c'est un lieu géométrique du point M. Deux arcs capables sont donc nécessaires et suffisants pour déterminer par leur intersection le point M. En topographie quatre lieux sont nécessaires pour le contrôle et la sécurité ce qui donne donc quatre arcs capables.

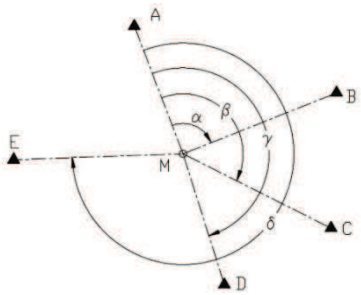


Figure 13:Point Relevé

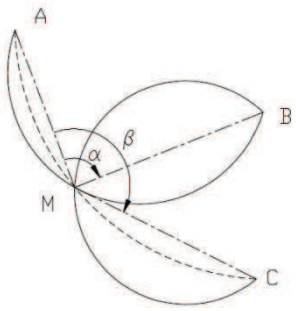


Figure 14 : Arc capable

On a alors dans notre cas : $\boldsymbol{AMB = \alpha = L_B - L_A}$

$\boldsymbol{AMC = \beta = L_C - L_A}$

$\boldsymbol{AMD = \gamma = L_D - L_A}$

$\boldsymbol{AME = \delta = L_E - L_A}$

3.1.3. Recoupement

Le recoupement est le procédé qui utilise simultanément l'intersection et le relèvement pour la détermination d'un point. Pour obtenir les quatre lieux nécessaires, il faut au minimum soit:

- une visée d'intersection et quatre de relèvement soit 1 + 3 = 4 lieux indépendants ;
- deux visées d'intersection et trois de relèvement soit 2 + 2 = 4 lieux indépendants ;
- trois visées d'intersection et deux de relèvement soit 3 + 1 = 4 lieux indépendants.

Le recoupement est pratique quand les points d'appui sont peu nombreux et stationnables.

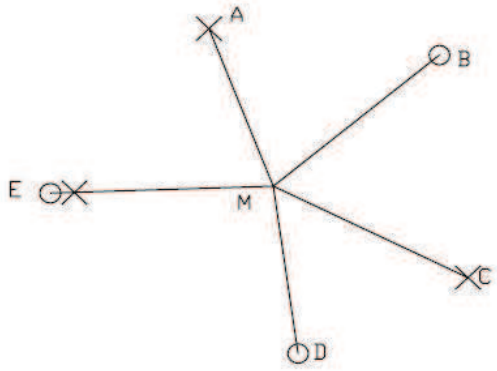


Figure 15 :Point recoupé

Ce croquis représente une recoupement avec des visées de relèvement sur E, B et D et des visée d'intersection sur A,E et C.

3.2. Procédés de calcul

3.2.1. Calcul approché du point

Selon le nombre de visée disponible, on calcule un point approché Mo à partir des visées de relèvement ou celles d'intersections. Pour cela, on utilise les formules de Delambre. Dans le cas de visées d'intersection on a la figure suivante :

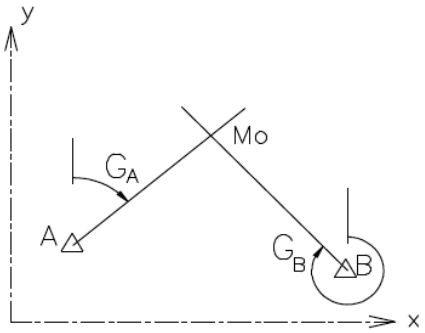


Figure 16 : Approximation du point par intersection

D'où l'on a :

$$\boldsymbol{Y_{Mo} = Y_A + \frac{(X_A - X_B) - (Y_A - Y_B).tanG_B}{tanG_B - tanG_A}} \qquad \boldsymbol{(28)}$$

$$\boldsymbol{X_{Mo} = X_A + (Y_{Mo} - Y_A)tanG_A} \qquad \boldsymbol{(29)}$$

Dans le cas d'un calcul avec des visées de relèvement, on a le croquis suivant :

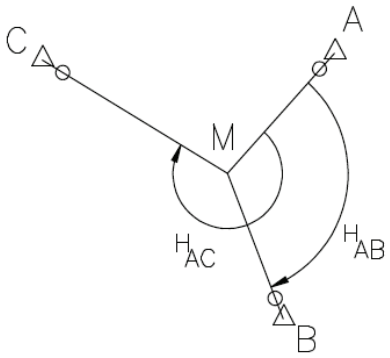


Figure 17: Approximation du point par relèvement

$$\boldsymbol{tanG_{AM} = \frac{(X_B - X_A).cotanH_{AB} - (X_C - X_A).cotanH_{AC} + (Y_C - Y_B)}{(Y_B - Y_A).cotanH_{AB} - (Y_C - Y_A).cotanH_{AC} - (X_C - X_B)}} \qquad \boldsymbol{(30)}$$

$$\boldsymbol{tanG_{BM} = \tan(G_{AM} + H_{AB}) = \frac{tanG_{AM} + tanH_{AB}}{1 - tanG_{AM}.tanH_{AB}}} \qquad \boldsymbol{(31)}$$

On reporte ensuite ces résultats dans les formules de Delambre utilisées pour l'intersection.

3.2.2. Calcul du point définitif par moindre carré

Les moindres carrés est la solution choisi pour compenser les coordonnées approchées. La détermination du point définitif se fera à partir du point approché. Le but est ainsi de trouver les corrections dx et dy à apporter aux coordonnées du point approché pour définir la position du point définitif.

La relation d'observation est la forme générale d'une observation. Sachant que chaque observation effectuée sur le terrain donne une expression reliant les valeurs approchées aux valeurs mesurées. Cette expression a comme forme générale :

valeur approché + variation = valeur observée + v

où v est appelé résidu.

En partant de la formule générale, la relation relative à une visée de la station S vers le point visé V sera :

variation de gisement (due à dx et dy) + Gapproché - Gobservé = v

ou autrement dit :

dG + Gapp - Gobs = v.

On doit donc transformer les observations angulaires qui sont uniquement des lectures azimutales (celles réduites du tour d'horizon) en gisements. Pour cas du relèvement, le G0 ne pourra être connu que quand les coordonnées du point relevé seront définitives. On peut donc écrire:

G0définitif = G0provisoire + dG0 (variation encore inconnue)

Soit L la lecture issue du tour d'horizon. En reprenant la formule générale :

dG + Gapp - Gobs = v

On peut remplacer :

dG + Gapp - (L + G0définitif) = v
dG + Gapp - (L + G0provisoire + dG0) = v
dG + Gapp - (Gobs + dG0) = v
dG + Gapp - Gobs - dG0) = v

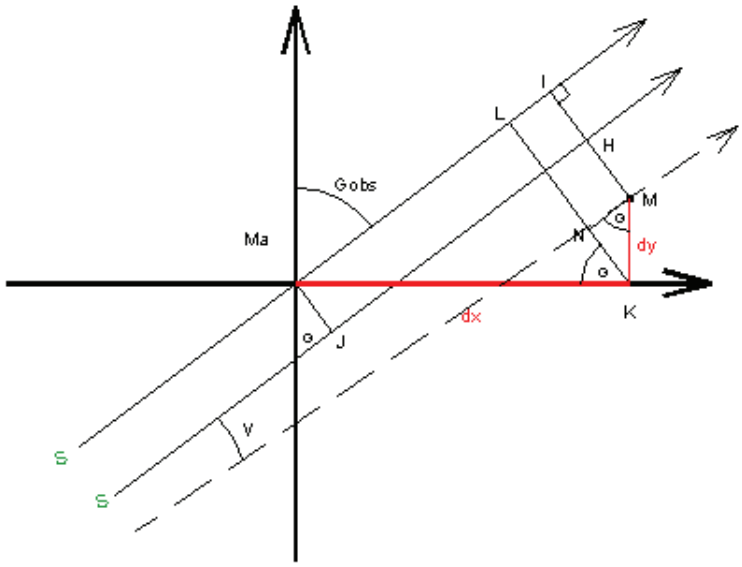


Figure 18 : Approximation de dG

Il nous reste à exprimer **dG.** Soit une droite SH représentant un lieu géométrique issue d'une station S à la distance D sous le gisement G. On se propose de calculer l'erreur angulaire commise v (le résidu) lorsqu'on prend M comme point définitif au lieu de choisir un point de la droite. Soit Ma, le point approché.

$$v = \frac{MH}{D} \text{ v en radians) d'ou } \frac{MH}{Dxsin\,1"} \text{ en grades}$$

Or MH = MI + IH or IH = MaJ = s x (Gcal – Gobs) = s x DG

avec **s = D x sin 1"**, la sensibilité, $dv = \frac{MH}{D} = \frac{sxDG+MI}{S} = DG + \frac{MI}{s}$

mais *MI = KL - NK = dx.cosG - dy.sinG*

$$\boldsymbol{dv = +\frac{dx.cosG}{s} - \frac{dy.sinG}{s} + G_{cal} - G_{obs}} \quad (32)$$

$$\boldsymbol{dv = +\frac{dx.cosG}{Dxsin1"} - \frac{dy.sinG}{Dxsin1"} + G_{cal} - G_{obs}} \quad (33)$$

En reprenant la remarque faite ci-dessus, concernant le G0 (**dG + Gapp - Gobs - dG0) = v**), l'expression devient:

$$\boldsymbol{+\frac{dx.cosG}{Dxsin1"} - \frac{dy.sinG}{Dxsin1"} + G_{cal} - G_{obs} - dG0 = v} \quad (34)$$

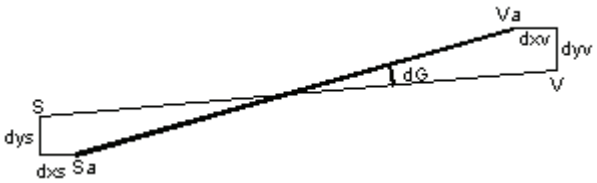


Figure 19 : Approximation de dx et dy

Maintenant dG que nous cherchions est exprimé en fonction des variations dx et dy nécessaires au déplacement entre le point visé approché Va et le point visé qui sera pris comme définitif V. Ici dG ne dépend donc que du point visé. Il est des cas (relèvement) où la station subit aussi des variations dxs et dys. dG en dépend donc aussi. On pourrait démontrer, à l'identique du point visé, cette influence. Ainsi dG devient:

$$dG = -\frac{dxs.cosG}{Dxsin1"} + \frac{dys.sinG}{Dxsin1"} + \frac{dxv.cosG}{Dxsin1"} - \frac{dyv.sinG}{Dxsin1"} \qquad (35)$$

d'où

$$-\frac{dxs.cosG}{Dxsin1"} + \frac{dys.sinG}{Dxsin1"} + \frac{dxv.cosG}{Dxsin1"} - \frac{dyv.sinG}{Dxsin1"} + G_{cal} - G_{obs} - dG0 = v \qquad (36)$$

Il s'agit de notre équation d'observation. Cette équation varie selon qu'il s'agit de visée de relèvement ou alors de visée d'intersection. Dans le cas de relèvement,

$$-\frac{dxs.cosG_{cal}}{D} + \frac{dys.sinG_{cal}}{D} + G_{cal} - G_{obs} - dG0 = v \qquad (37)$$

Et pour l'intersection,

$$+\frac{dxv.cosG_{cal}}{D} - \frac{dyv.sinG_{cal}}{D} + G_{cal} - G_{obs} = v \qquad (38)$$

Où D : distance entre la station et le point visé
G_{cal} : gisement calculé à partir du point approché
G_{obs} : gisement observé sur chaque visée

La résolution du système d'équation qui sera formée par chaque visée permettra d'obtenir les variations dX et dY et de calculer ainsi les coordonnées définitives du point approché.

$$X_M = X_{M0} + dx \qquad (39)$$
$$Y_M = Y_{M0} + dy \qquad (40)$$

4. Transformation de coordonnées par G.P.S. [10]

4.1. Rappel sur la projection Laborde

La projection Laborde, du commandant du même nom, est la projection cartographique principalement utilisée à Madagascar, créé en 1926 et en usage officiel depuis. La projection Laborde utilise une triple projection conforme de manière à obtenir une projection conforme cylindrique oblique à la fin. La première étape de la projection consiste à projeter l'ellipsoïde sur une sphère, appelée « aposphère ». L'étape suivante est de projeter la sphère sur un cylindre selon la projection Mercator Gauss-Schreiber Transverse. La dernière étape distorde le plan obtenu par rotation de manière à ce que l'isomètre central soit le grand axe de Madagascar.

Une projection cylindrique conforme a été choisie de manière à ce que l'isomètre central soit selon le grand axe de Madagascar. L'orientation de l'axe du cylindre est de 21 grades. Comme toute projection conforme, la projection Laborde conserve localement les angles.

Voici les paramètres de cette projection :

- Ellipsoïde : International 1924:
  - *Demi grand axe : 6 378 388.000*
  - *Première excentricité : 0.08199188998*
  - *Aplatissement 1/f : 297.00*
- Unité d'angle : grade
- Longitude de l'origine de projection : M_0 *= 49 grades Est.*
- Latitude de l'origine de projection : L_0 *= 21 grades Sud.*
- Méridien d'origine : Paris *(il faut ajouter 2.5969213 grad pour passer par rapport à Greenwich)*
- Coordonnées cartographiques de l'origine :
  - X_0 *= 400000 m*
  - Y_0 *= 800000 m*
- Azimut de la ligne isomètre : *21 grades.*
- Facteur d'échelle au centre de projection : K_0 *= 0.9995*

4.2. Calcul des coordonnées rectangulaires à partir des coordonnées géographiques

On commence d'abord par calculer les constantes de projection :

- Grande normale

$$N_0 = \frac{a\sqrt{1+e'^2}}{\sqrt{1+e'^2 cos^2 L_0}} = 6380638.718\ m \qquad (41)$$

- Rayon de l'ellipse méridienne

$$\rho_0 = \frac{a\sqrt{1+e'^2}}{\sqrt[\frac{3}{2}]{1+e'^2 cos^2 L_0}} = 6342217{,}332\ m \qquad (42)$$

- Rayon de courbure au point central

$$R_0 = K_0\sqrt{N_0\rho_0} = 6\ 358\ 218.318\ \text{m} \qquad (43)$$

- Latitude sur la sphère au point central

$$tan\varphi_0 = \sqrt{\frac{\rho_0}{N_0}}.tanL_0 \qquad (44)$$ d'où φ_0= 20.94115410 Sud

- Coefficient de longitude

$$\alpha = \frac{sinL_0}{sin\varphi_0} = \sqrt{1+e'^2 cos^4 L_0} = 1.002\ 707\ 5409 \qquad (45)$$

On effectue ensuite le passage des coordonnées géographiques en coordonnées rectangulaires.

L : latitude sur l'ellipsoïde
φ : Latitude sur la sphère
M : Longitude sur l'ellipsoïde
λ : Longitude sur la sphère
L'indice 0 indique que le point est une origine.

Calcul des coordonnées sur la sphère

$$\boldsymbol{\lambda = \alpha(M - M_0)} \qquad \mathbf{(46)}$$

$$\boldsymbol{\varphi = \arcsin\{th[\alpha(argth(sinL) - eargth(e.sinL)]\}} \qquad \mathbf{(47)}$$

Calcul des coordonnées de Cassini Solder

$$\boldsymbol{v = argth(cos\varphi.sin\lambda)} \qquad \mathbf{(48)}$$

$$\boldsymbol{u = \arctan\left(\frac{tan\varphi}{cos\lambda}\right)} \qquad \mathbf{(49)}$$

Calcul des coordonnées symétriques

$$\boldsymbol{x = R.(\varphi_0 - \varphi - u)} \qquad \mathbf{(50)}$$

$$\boldsymbol{y = R.v} \qquad \mathbf{(51)}$$

Coordonnées cartésiennes

$$\boldsymbol{X = X_0 + x + Ax^3 - 3Bx^2y - 3Axy^2 + By^3} \qquad \mathbf{(52)}$$

$$\boldsymbol{Y = Y_0 + y + Bx^3 + 3Ax^2y - 3Bxy^2 + Ay^3} \qquad \mathbf{(53)}$$

Où A et B sont respectivement :

$$\boldsymbol{A = \frac{1}{12R^2}(1 - ncos2\theta)} = 0.43256013 * 10^{-15} \qquad \mathbf{(54)}$$

$$\boldsymbol{B = \frac{1}{12R^2}nsin2\theta} = 0.12634047 * 10^{-14} \qquad \mathbf{(55)}$$

Partie III

Conception et présentation de l'application

CHAPITRE 5

LES BASES DE LA PROGRAMMATION SUR ANDROID

1. Qu'est ce qu'une application Android

Une application Android est un logiciel téléchargeable et exécutable sur les téléphones intelligents ou tablettes tournant sous le système du même nom. Les applications sont accessibles dans les boutiques d'applications (Play Store, Amazon AppStore, Mobogenie) et présentent plusieurs avantages. Elles peuvent être gratuites ou payantes.

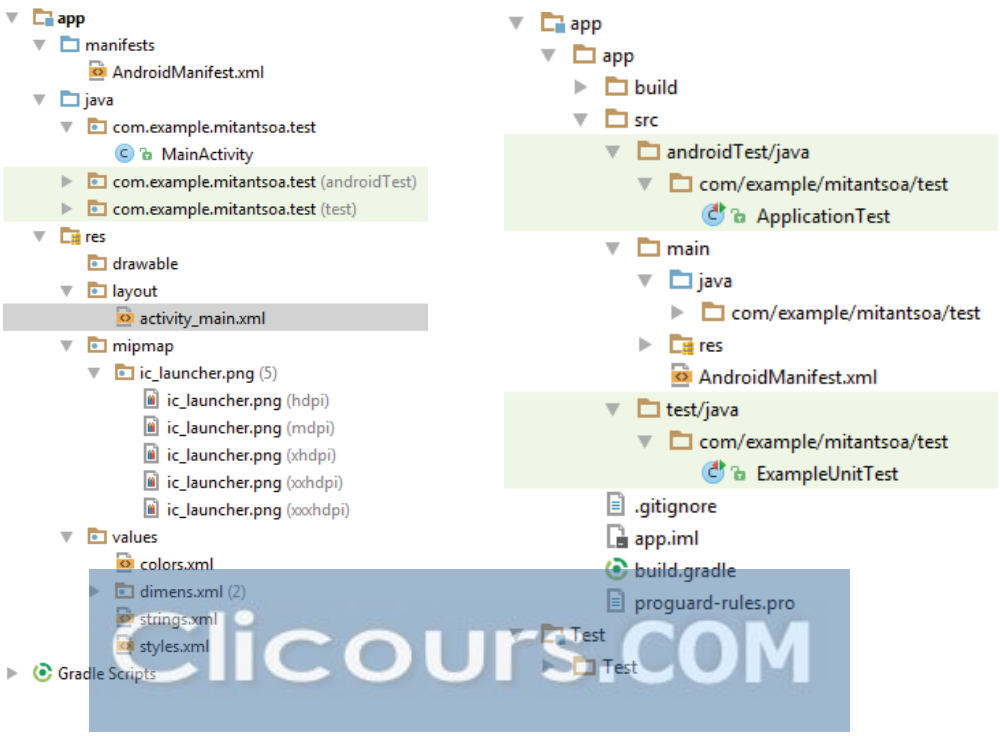
Les applications Android sont développées sur des ordinateurs. Le langage de programmation utilisé est le Java. Ce langage peut être utilisé sur tous les ordinateurs quel que soit son système d'exploitation.

2. Structure d'un projet Android

Un projet possède la structure normalisée. Les fichiers utilisés sont organisés comme suit :

- le manifeste *(AndroidManifest.xml)* qui décrit l'application à construire et ses composants ;
- le dossier *res (pour ressources)* qui contient les ressources du projet (les layouts, les chaînes de caractères, les images, les fichiers internes, les fichiers XML, les couleurs et thèmes) dont les sous-dossiers sont :
    - *Drawable* : pour les images, classé en trois dossiers, un par qualité (haute, moyenne, faible),
    - *Layout* : pour définr les différents fichiers de layouts des activités qui déclarent et positionnent les composants graphiques,
    - *Values* (pour les chaînes de caractères, les couleurs, les thèmes, les préférences…),
    - *Raw* (pour les fichiers brutes),
    - *XML* (pour les fichiers XML) ;
- le dossier *src* pour les sources (classe Java) ;

et enfin le dossier test pour les tests.



Le fichier manifeste est un élément clef du projet, il définit :

- votre projet et sera interprété par le système et le Play Store;
- les dépendances systèmes de votre projet (par exemple : la version du système, les instruments nécessaires, la géolocalisation) ;
- la version de votre projet (un nom de version name et un numéro de version (un Int)) ;
- les différents composants de votre application (Activity, Service, ContentProvider).

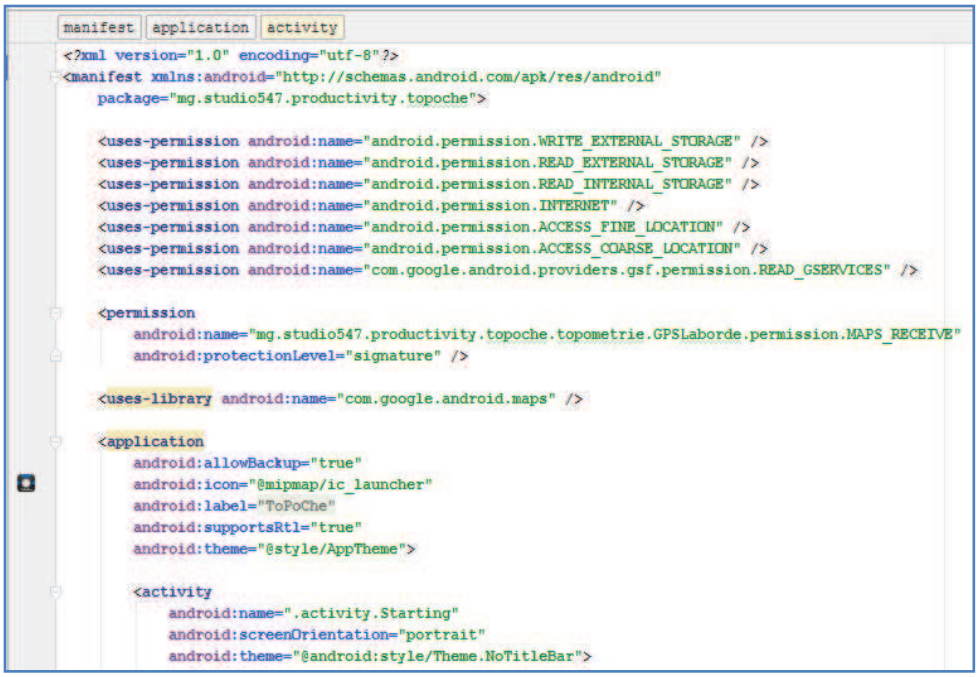


Figure 20: Extrait du fichier AndoridManifest.xml

3. Activités et interfaces graphiques

3.1. Notion d'activité

Une activité est la composante principale pour une application Android. Elle représente l'implémentation et les interactions de vos interfaces. En d'autres termes, une activité est un support sur lequel nous allons greffer une interface graphique. Cependant, ce n'est pas le rôle de l'activité que de créer et de disposer les éléments graphiques, elle n'est que l'échafaudage sur lequel vont s'insérer les objets graphiques et elle va établir les liens entre l'interface graphique et le logique derrière les éléments qui la constitue.

De plus, une activité contient des informations sur l'état actuel de l'application : ces informations s'appellent le *context*. Ce *context* constitue un lien avec le système Android ainsi que les autres activités de l'application, comme le montre la figure suivante.

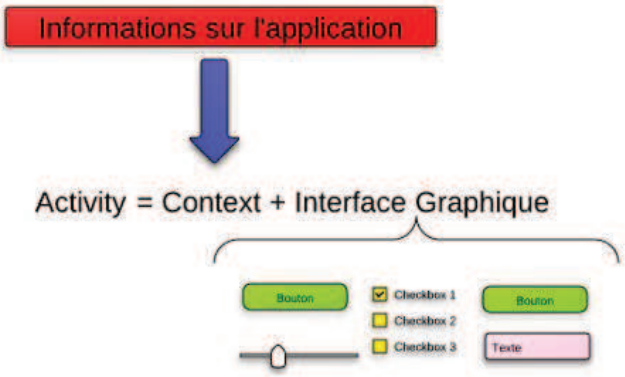


Figure 21: Composante d'une activité[4]

Le système, pour des raisons de priorisation d'activités (coup de téléphone), peut "tuer" une activité quand il a besoin de ressources. Une activité n'a pas de contrôle direct sur son propre état, il s'agit plutôt d'un cycle rythmé par les interactions avec le système et d'autres applications. Une activité possède quatre états que sont :

- « Active » : l'activité est lancée par l'utilisateur, elle s'exécute au premier plan ;
- « En Pause » : l'activité est lancée par l'utilisateur, elle s'exécute et est visible, mais elle n'est plus au premier plan. Une notification ou une autre activité lui a volé le focus et une partie du premier plan ;
- « Stoppée » : l'activité à été lancée par l'utilisateur, mais n'est plus au premier plan et est invisible. L'activité ne peut interagir avec l'utilisateur qu'avec une notification ;
- « Morte » : l'activité n'est pas lancée.

Le schéma suivant indique le cycle de vie d'une activité et les méthodes appelées lors des changements d'état :

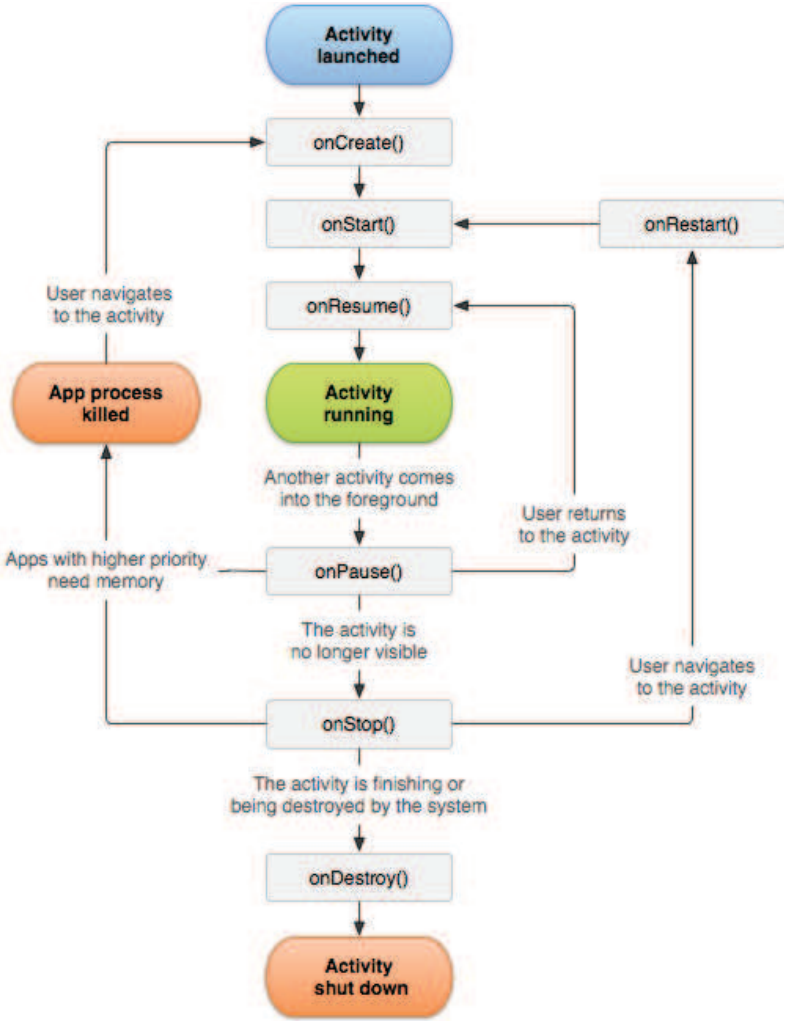


Figure 22: Cycle de vie d'une activité[5]

Il existe une activité pour tous les écrans d'une application. L'activité pourra occuper la totalité de l'écran ou partiellement, ou être dans une dialogue. Elle pourra être lancée depuis une application mais aussi par n'importe quelle autre application. En fait, une activité ne se limite pas à un écran, il s'agit plus d'une portion de code capable de fonctionner de manière totalement indépendante. Chacune des activités devra être déclarée dans le fichier *AndroidManifest.xml* pour pouvoir être utilisée. Reprenons la figure précédente du manifeste :

```
<activity
    android:name=".activity.Starting"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.NoTitleBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity
    android:name=".activity.Main"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.NoTitleBar">
</activity>
```

Figure 23: Extrait du fichier AndroidManifest.xml

Pour créer une classe comme activité, il faut étendre la classe *Activity*. Notre code ressemblera donc au minimum à cela :

```
package com.example.mitantsoa.test;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Figure 24 : Extrait d'une classe hérité de Activity

Ligne par ligne, cette portion de code nous montre le package de l'application *com.example.mitantsoa.test,* les imports de classes qui se trouvent dans le package *android*, la classe *MainActivity* qui hérite de la classe *Activity* du package cité précédemment et les méthodes relatives au cycle de vie d'une activité ainsi que la méthode *setContentView(View view)* qui permet d'indiquer quelle interface graphique correspond à l'activité. (ici un layout)

3.2. Constitution de l'interface graphique

3.2.1. Vues

La classe de base pour réaliser une vue sous Android est la classe *View*. Cette classe est la base des interfaces utilisateur (disposition d'écran, interaction). Une ou plusieurs vues peuvent être regroupées dans ce que l'on appelle *ViewGroup* .

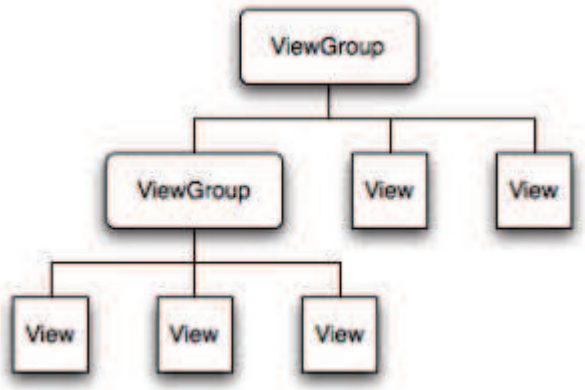


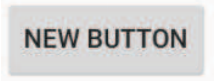
Figure 25: Arborescence de View[5]

Les vues sont ainsi la source des interfaces graphiques. Les vues les plus couramment utilisées sont les suivantes :

- *TextView* : affiche une chaine de caractère

- *EditText* : affiche un champs où on saisit un texte

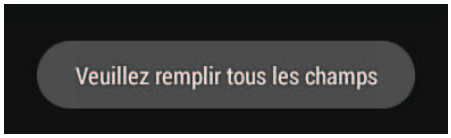
- *Button* : définit un bouton cliquable



- *CheckBox* : donne un ensemble de case à cocher (sélection multiple)

- *RadioButton* : ensemble de cases dont seule une peut être coché

- *Toast* : petit message qui s'affiche sur l'écran temporairement



Ceci n'est pas une liste exhaustive de toutes les vues puisqu'il existe une infinité de possibilité de combinaisons et de variations. Parcourir le site dédié à la documentation Android permettra d'approfondir le sujet.

3.2.2. Les Layouts

Pour constituer une interface graphique, il faut mettre en page les différentes vues à l'aide des layouts. On peut déclarer les Layouts par code Java ou par XML. La déclaration en XML convient pour les interfaces statiques et la déclaration Java convient aux interfaces dynamiques. Nous allons nous pencher plus sur la constitution des interfaces avec le XML. Ce langage balisé permet de définir les propriétés de chaque vue et leur agencement par rapport au layout. Vue et layout peuvent partager ainsi des attributs commun, dont voici quelques exemples :

- **android:layout_width** : définit la largeur de l'élément.
- **android:layout_height** : définit la hauteur de l'élément.
- **android:text** : Texte à afficher dans l'élément.

Pour plus de clarté, voici une portion de code XML pour une activité :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
android:orientation="vertical" >

<ImageView
    android:id="@+id/android_picture"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/normal_padding"
    android:src="@drawable/android"
    android:contentDescription="@string/image_content_description" />

<Button
    android:id="@+id/create_account"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="@dimen/small_padding"
    android:text="@string/create_account" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:paddingTop="@dimen/normal_padding"
    android:text="@string/email"
    android:textSize="@dimen/normal_text_size"
    android:textColor="@color/black_color" />
```

On retrouve ici différentes vues ainsi qu'un layout qui organise ces vues. Cet extrait de code nous donne l'extrait d'interface suivante :

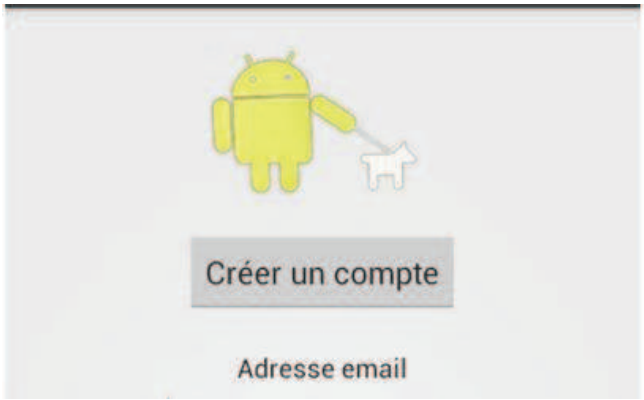


Figure 26: Extrait d'interface graphique

Selon le LinearLayout, l'orientation des vues est vertical, la disposition des différents éléments de ce layout sont dressés un par un de haut en bas selon leur apparition dans le code. Il existe plusieurs types de layout selon le rendu que l'on souhaite avoir, dont voici quelques un :

- *LinearLayout* : met les vues sur une même ligne ou une même colonne selon l'orientation choisie
- *RelativeLayout* : permet de mettre les vues en fonction de la position des autres.
- *TableLayout* : dispose les vues comme dans un tableau

- *FrameLayout* : permet d'afficher une vue unique. Il est possible d'afficher plusieurs élément en jouant sur leur visibilité.

3.2.3. Gestion des évènements

Pour interagir avec notre application, nous devons gérer les évènements sur les vues.

3.2.3.1. Récupération des vues

A chaque vue doit correspondre un identifiant unique afin d'être discernable et utilisable dans un code Java. L'attribut *android:id* se charge d'attribuer cet identifiant. Cet identifiant sera sauvegarder dans la classe R qui conserve les attributs de la vue.

```
<TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

Nous pouvons ensuite caster la vue c'est-à-dire le convertir en un objet hérité de View que l'on pourra manipuler. Voici un exemple pour éclaircir ces propos :

```
TextView monTexte = null;

  @Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

    monTexte = (TextView)findViewById(R.id.text);
    monTexte.setText("Le texte de notre TextView");
  }
```

Dans cette portion de code qui crée une activité, nous déclarons d'abord un objet du type *TextView* que l'on initialise. Le contenu de la méthode *onCreate* nous montre ensuite l'affection d'un fichier *layout* à l'activité, qui sera suivi du cast de la vue identifié *text* en un *TextView.* Cela est indispensable puisque la méthode *findViewById(int id)* retourne un objet de type *View*. L'objet est ensuite manipulable dans le suite du code.

3.2.3.2. Programmation des évènements

La gestion des évènements est fait à travers les listeners. Un listener est un objet qui détecte les évènements qui ne sont autres que les interactions de l'utilisateur avec l'interface graphique (clique sur un bouton, toucher glisser,...). Prenons l'exemple d'un bouton. Par exemple, pour intercepter l'évènement clic sur un Button, on appliquera l'interface *View.OnClickListener* sur ce bouton. Cette interface contient la méthode de *callbackvoid onClick(View vue)* — le paramètre de type *View* étant la vue sur laquelle le clic a été effectué, qui sera appelée à chaque clic et qu'il faudra implémenter pour déterminer que faire en cas de clic. Par exemple pour gérer d'autres évènements, on utilisera d'autres méthodes :

- *View.OnLongClickListener* pour les clics qui durent longtemps, avec la méthode *boolean onLongClick(View vue)*. Cette méthode doit retourner *true* une fois que l'action associée a été effectuée.
- *View.OnKeyListener* pour gérer l'appui sur une touche. On y associe la méthode *boolean onKey(View vue, int code, KeyEvent event)*. Cette méthode doit retourner *true* une fois que l'action associée a été effectuée.

Voici un exemple de code utilisant un listener, après le cast du bouton auquel il est adressé :

```
carnet = (Button) findViewById(R.id.cr);
carnet.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent Carnet = new Intent(Main.this,Carnet.class);
                 startActivity(Carnet);
             }
       });
```

Ceci n'est qu'un aperçu de ce qu'est un listener ainsi qu'une des méthodes d'instanciation de ce dernier. Nous recommandons toujours de parcourir la documentation officielle d'Android pour approfondir le sujet.

4. Stockage de données[4]

4.1. Données internes et données externes

Une application Android a besoin de stocker des données sur le terminal où elle est installée. Non seulement ces données pourront contenir les préférences de l'utilisateur mais aussi être utiliser pour créer des fichiers. Une application Android peut stocker deux types de données selon leurs emplacements.

Les données que l'application stocke dans la mémoire interne du terminal sont les données internes. Ces données sont privées et donc ne sont pas accessibles que par le biais de l'application. Celles-ci sont supprimées du terminal lorsque l'utilisateur supprime l'application correspondante. Ces données se trouvent dans le répertoire */data/data/package_de_l_application* que le système Android créé par défaut dans la mémoire du téléphone. Ces données sont donc présentes tant que l'application est présente dans le téléphone.

Les données qu'une application stocke dans la mémoire externe (carte SD) sont les données externes. Ces données contrairement à celles internes ne sont pas privées et sont accessibles par n'importe qu'elle application. Ainsi, une application pourrait accéder et modifier un fichier utilisé par une autre et cela causerai un disfonctionnement dans cette dernière. La mémoire externe n'est pas toujours aussi disponible, il faut donc la tester préalablement avant d'y écrire un fichier.

4.2. Les bases de données SQLite

Une base de données est un ensemble structuré et organisé permettant de stocker des grandes quantité de données informatiques brute.

SQLite est un système de gestion de base de données (SGBD) open source très compacte supportant la syntaxe SQL. Il fonctionne sans serveur et est donc exécuter dans le même processus que l'application qui l'utilise. SQLite est donc l'idéal pour les systèmes mobiles compte tenu de sa taille et ce mode d'exécution.

Andorid inclut SQLite dans chaque terminal. Son utilisation ne nécessite pas de droits d'administrations ni de configuration particulières. Une base de données SQLite fait partie des données internes d'une application.

La classe qui permet de travailler sur les bases de données SQLite est la classe *SQLiteDatabase*. Elle fournit les méthodes qui permet de manipuler les données de la base (insérer, modifier, supprimer) ainsi que les méthodes permettant d'exécuter des requêtes. La classe *SQLiteOpenHelper* comporte les méthodes requises pour créer et mettre la base de données. C'est par son intermédiaire que se fait la création des tables et de leurs champs dans lesquels les données seront stockées.

Voici un extrait de code de création d'une base de données et de sa table :

```
public class DatabaseHandler extends SQLiteOpenHelper {
 public static final String METIER_KEY = "id";
 public static final String METIER_INTITULE = "intitule";
 public static final String METIER_SALAIRE = "salaire";

 public static final String METIER_TABLE_NAME = "Metier";
 public static final String METIER_TABLE_CREATE =
 "CREATE TABLE " + METIER_TABLE_NAME + " (" +
 METIER_KEY + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
 METIER_INTITULE + " TEXT, " +
 METIER_SALAIRE + " REAL);";

 public DatabaseHandler(Context context, String name, CursorFactory factory, int
version) {
    super(context, name, factory, version);
 }

 @Override
 public void onCreate(SQLiteDatabase db) {
        db.execSQL(METIER_TABLE_CREATE);
        }
 }
```

4.3. Lecture et écriture des fichiers

Android garde les même principes que Java pour l'écriture et la lecture de fichiers. Le cas le plus simple est la manipulation de flux d'octets, faisant intervenir les objets *FileInputStream* pour la lecture et *FileOutputStream* pour l'écriture. Les fichiers sont stockés soit sur la mémoire interne, soit sur la mémoire externe du terminal. La seule différence dans leur manipulation réside dans le fait que pour le cas des fichiers externes, il est impératif de tester l'existence du stockage externe avant d'écrire ou lire le fichier en question. Ces extraits de codes suivant montre l'utilisation de ces objets cités auparavant :

- pour l'écriture :

```
FileOutputStream output = null;
String userName = "Mitantsoa";
```

```
try {
  output = openFileOutput(PRENOM, MODE_PRIVATE);
  output.write(userName.getBytes());
  if(output != null)
    output.close();
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

- pour la lecture :

```
FileInputStream fIn = null;
InputStreamReader isr = null;

char[] inputBuffer = new char[255];
String data = null;

try{
 fIn = context.openFileInput("settings.dat");
    isr = new InputStreamReader(fIn);
    isr.read(inputBuffer);
    data = new String(inputBuffer);
 }catch (Exception e) {
       e.printStackTrace();
 }
```

CHAPITRE 6

PRESENTATION DE L'APPLICATION TOPOCHE

1. But et philosophie

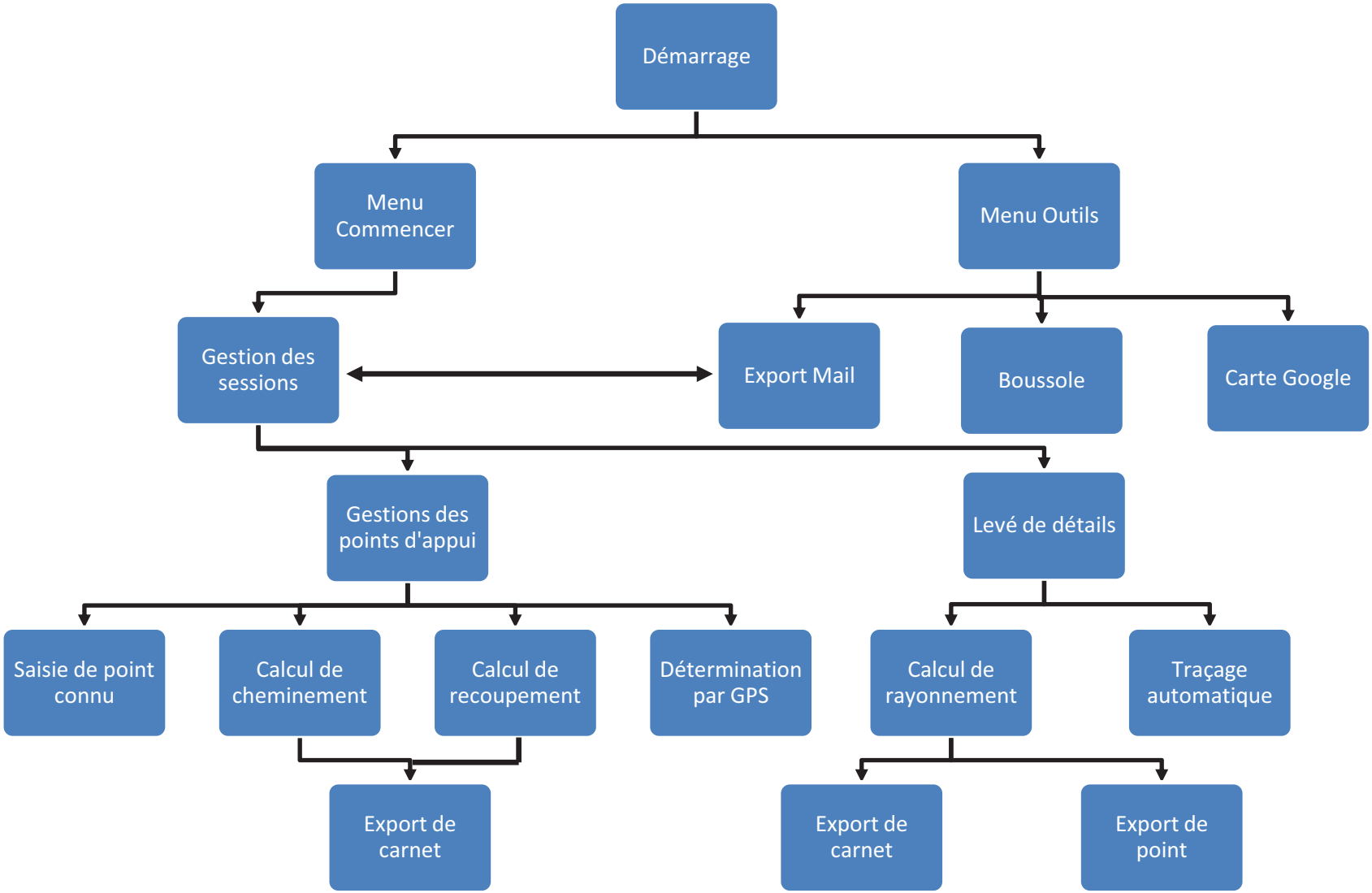
Nous vivons actuellement dans une ère où le numérique et la technologie font partie du quotidien de l'homme et en particulier celui des ingénieurs et techniciens. Auparavant et particulièrement sur notre île, un géomètre n'utilise que des moyens rustiques pour effectuer son lever et faire ses calculs. Les observations sont écrites sur papier et sont ressaisis sur un ordinateur pour être calculer et dessiner. Prenant compte de la durée que peut prendre une session de travail, il nous est venu l'idée de développer une application mobile d'aide aux travaux topographiques de terrain. Cette application serait à la fois un carnet de terrain qui enregistrerait les observations, un logiciel de calcul, et une carte satellite.

Même si l'application automatise presque tous les calculs relatifs à la topographie, elle a pour raison de confirmer et d'affirmer la maitrise des différents procédés et méthodes utilisés en topographie. Cet outil polyvalent installé sur un smartphone tient dans une poche. Il vous permet si vous avez un appareil de mesure de faire de la topographie partout et quand vous le voulez sans grande contrainte. Pour ces raisons, nous avons décidé d'intitulé notre application "ToPoChe" qui n'est autre qu'une combinaison des mots topographie et poche.

2. Conception

L'application ToPoChe a pour objectif de récolter les observations faites sur le terrain et de calculer les coordonnées des points issus de ses dernières. Ce logiciel est développé sur la plateforme Android qu'on retrouve sur les téléphones et tablettes pour être utilisable partout sans contrainte majeur de taille ou d'énergie sur le terrain. Le programme met aussi en avant le récepteur GPS présent sur le terminal, ainsi que l'API Google Maps intégré dans le système mobile à travers un outil de localisation.

L'organigramme sommaire suivant résume le processus de fonctionnement de l'application :

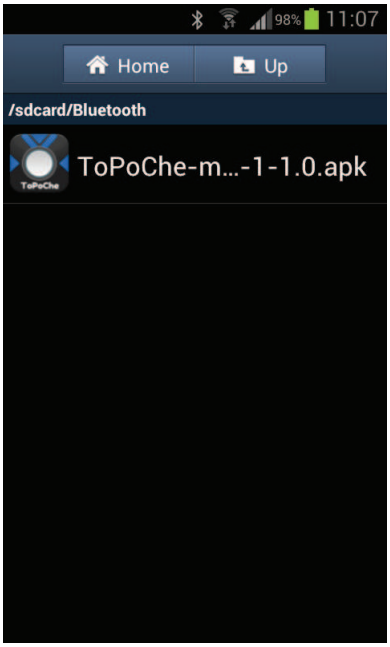


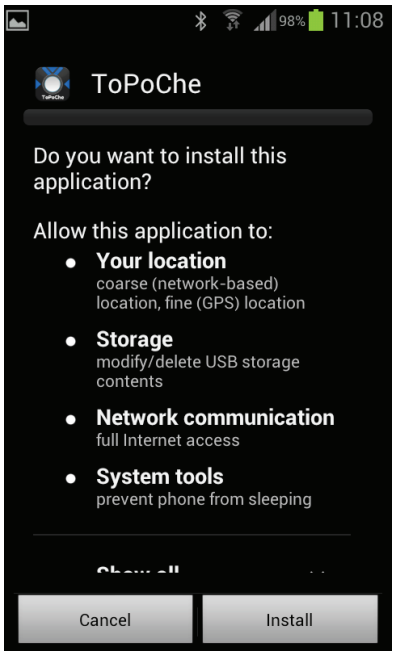
Organigrammme 3 : Organigramme simplifié du processus de l'application

3. Description du logiciel

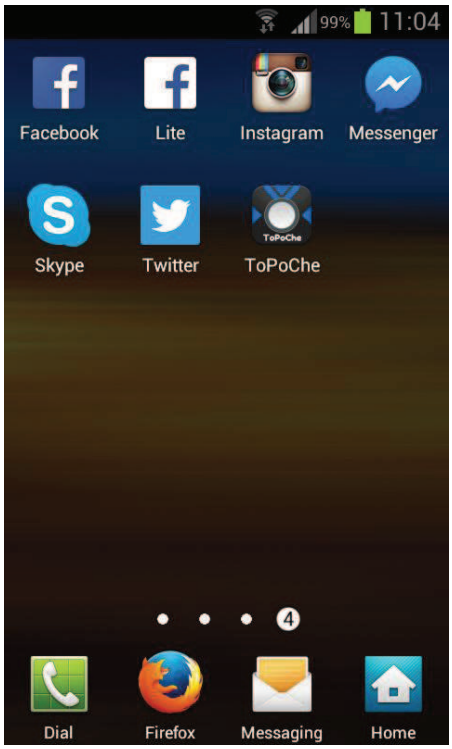
3.1. Démarrage de l'application

Tout d'abord, avant d'utiliser notre application, il faut l'installer sur un terminal fonctionnant sous Android. Le fichier d'installation de celui a pour extension *.apk*. On le copie sur la mémoire du terminal et on clique dessus pour lancer l'installation.





Une fois installée, l'application apparait dans la liste des applications du terminal et est prêt à être utilisé.



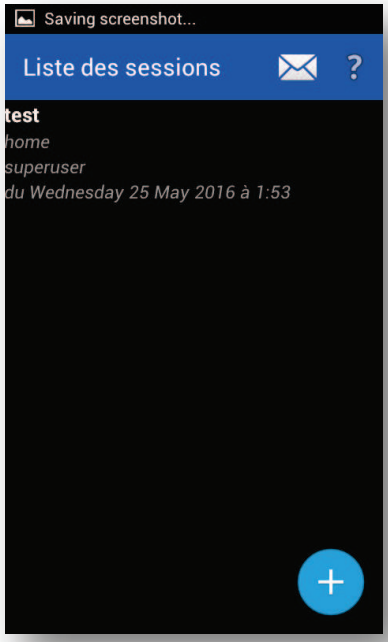
Pour démarrer l'application, il suffit de cliquer sur l'icone la représentant et on accède à l'écran d'accueil suivant.

Un léger chargement s'effectue et on accède ensuite à la page de démarrage. Cet écran regroupe deux boutons qui mènent au deux principaux menus de l'application

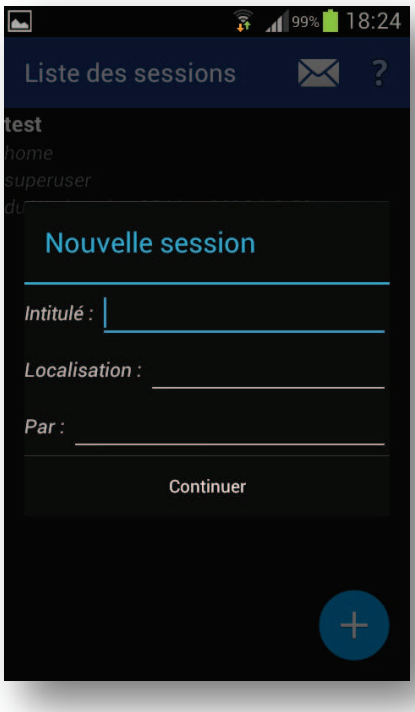
3.2. Menu "Commencer"

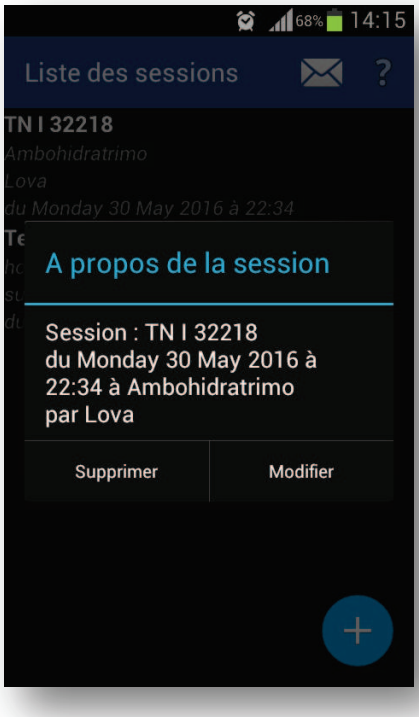
3.2.1. Gestionnaire de session

On accède à cette page après avoir cliquer sur le bouton "Commencer" de l'activité précédente.



Chaque travail sur terrain est représenté par une session dans notre application. Ainsi cette page gère l'ajout, la modification, ou la suppression de sessions. Les sessions sont présentées comme on le voit en liste.





3.2.2. Gestionnaire de point d'appui

Chaque session de travail nécessite des points d'appui pour pouvoir rattacher les points levés et calculer. Cette activité est le point de départ pour toute détermination de point à travers l'option Ajouter représenter par : 

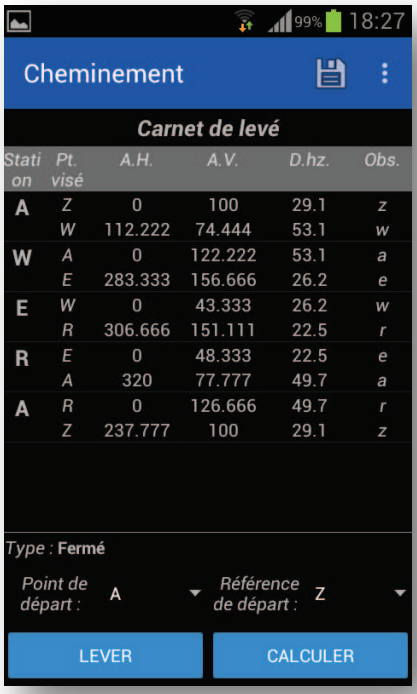
3.2.2.1. Saisie de point connu

Cette option permet la saisie directe des coordonnées de point connu.

3.2.2.2. Calcul de cheminement

Lorsqu'on clique sur l'option Cheminement, on accède à une page similaire à celle qui gère les sessions. Elle permet éditer le type du cheminement ainsi que son nom.

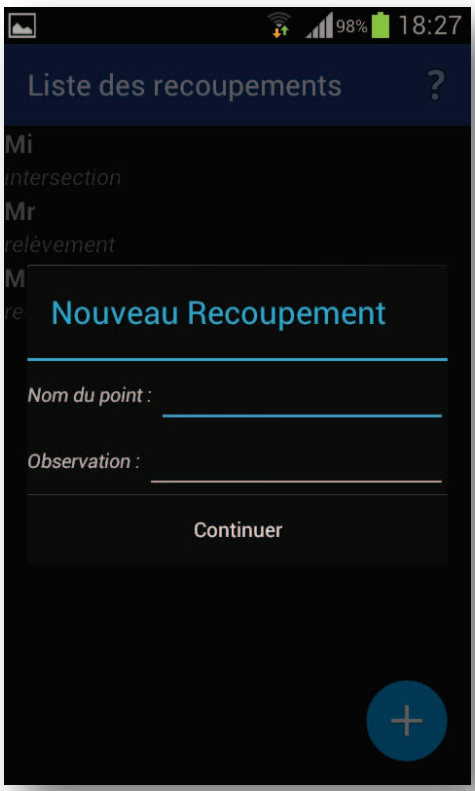
On accède ensuite à la page de lever et de calcul. On y retrouve un carnet de levé, ainsi qu'une boite de dialogue de saisie de donner lorsqu'on appui sur le bouton "Lever"



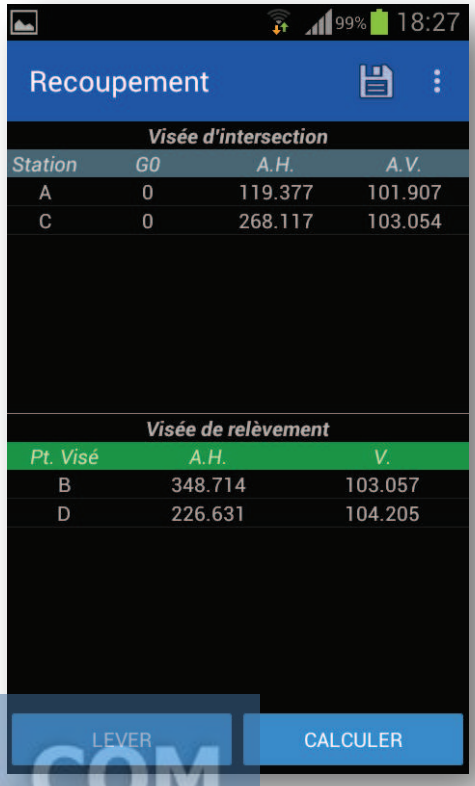
Une fois le levé achevé, on a accès au bouton "Calculer" qui comme son nom l'indique calcul les coordonnées des points du cheminement. Une boite de dialogue s'affiche en fin de calcul pour nous faire part des résultats.

3.2.2.3. Calcul de recoupement

Le principe d'organisation et d'affichage est similaire à celui du calcul de cheminement. Une page gère les sessions de recoupement et la suivante affiche les carnets de levés et permet de calculer les coordonnées du point recoupé.

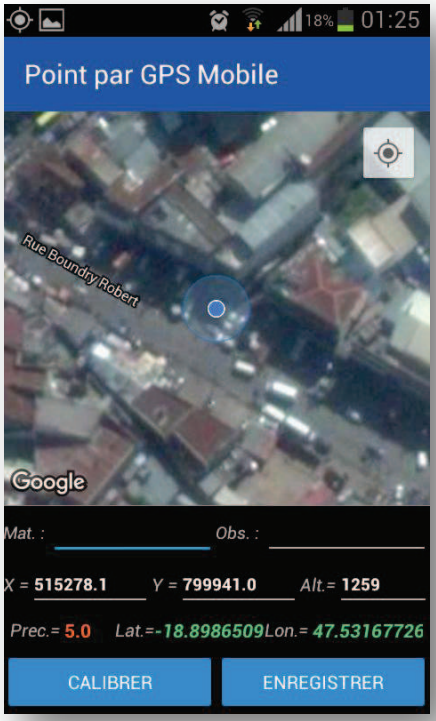


La particularité de ce module et qu'il calcule les coordonnées en intersection ou en relèvement uniquement selon les visées disponibles.



3.2.2.4. Point par GPS mobile

Une des méthodes rapides de détermination de point et la détermination par GPS. Notre application est capable de donner les coordonnées d'un point dans le système Laborde à l'aide du capteur GPS du terminal et un algorithme de calcul de transformation de coordonnées. L'activité montre une portion de carte Google pour permettre à l'utilisateur de bien confirmer sa position.



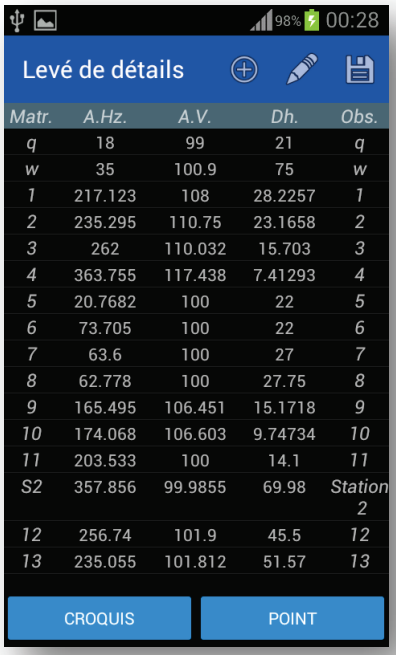
Une fonction de calibration est accessible via le bouton "Calibrer", sachant qu'un signal GPS est variable dans le temps et l'espace et est aléatoirement induit d'erreur.

3.2.3. Levé de détails

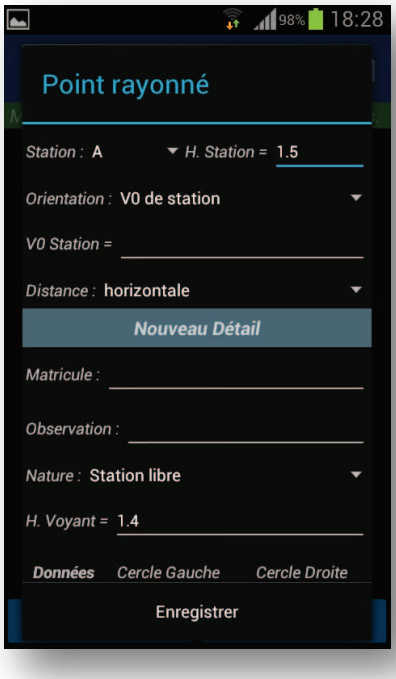
Etant satisfait du nombres de point d'appui, nous pouvons commencer le lever de détails. Pour ce faire on clique sur le bouton "Lever de détails " l'activité qui gère les points d'appuis.

3.2.3.1. Carnet de levé et liste des points rayonnés

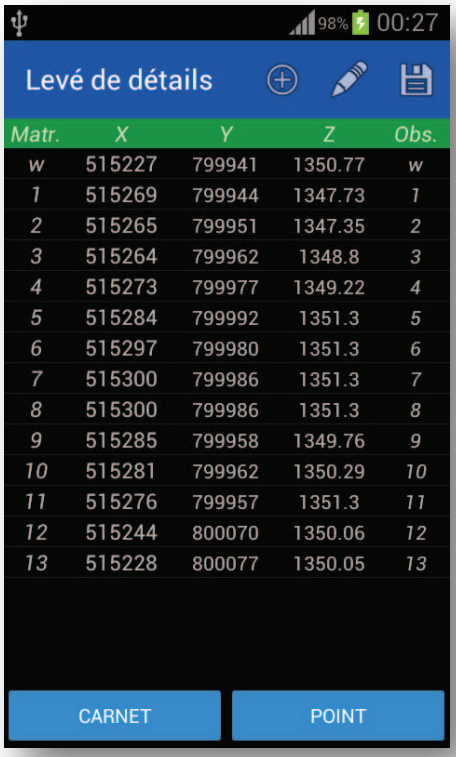
Cette vue liste les points de détails levés.



Pour accéder au formulaire de lever, on clique sur le bouton  de la barre des menus. Une boite de dialogue apparait alors et on y saisit les données d'observations.

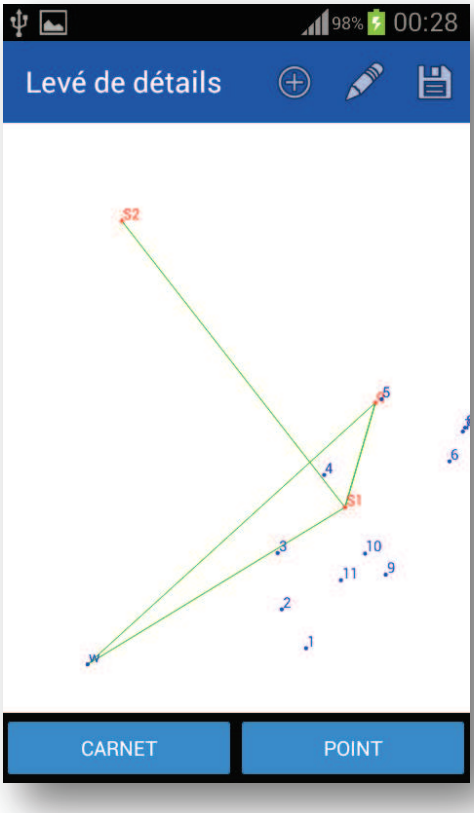


Le bouton "Point" permet d'afficher la liste des points rayonnés



3.2.3.2. Croquis

Notre application a la capacité de dessiner un croquis simple de la session. Le bouton "Croquis/Carnet" permet de basculer entre les vues qui contient le carnet et la vue qui dessine le croquis.

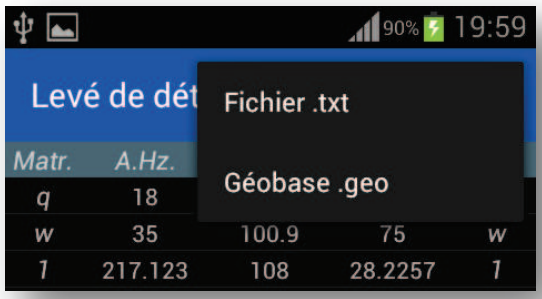


3.2.3.3. Export

La collecte de données et le calcul sont un fait, mais il est aussi nécessaire pour une application de pouvoir exporter les données qu'elles créent afin d'être exploiter sur une autre plateforme(logiciel de CAD ou de SIG). C'est pour cela que notre application possède une fonction d'export pour les rendre utilisables et pallier l'encapsulation de celles-ci.

Les fonctions d'export de trouvent sur la plupart des activités de l'application, et sont symbolisées par l'icone  présente dans la bar des menus.

L'application pour sa première version exporte les fichiers dans le format *.txt* et le format *.geo* compatible dans le logiciel de traitement de données topographiques Covadis. Les fichiers sont générés dans le répertoire "Topoche" que l'application crée dans la mémoire du terminal.



3.3. Menu "Outils"

3.3.1. Menu Export

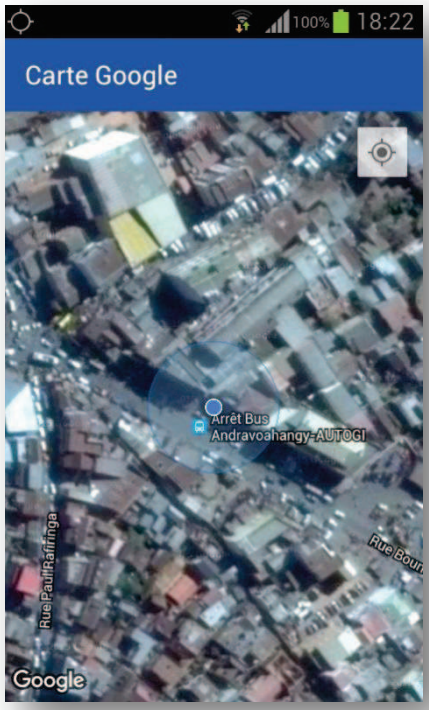
Ce menu permet d'envoyer un mail avec lequel on pourra attacher en pièce jointe les fichiers exportés par l'application. Cette option est pratique le cas où les données doivent parvenir immédiatement à un bureau de traitement éloigné après la collecte de données.

3.3.2. Menu Boussole

Il est parfois utile de s'orienter avec une boussole lors de mission de reconnaissance, ou juste pour se repérer. Pour alléger le topographe, notre application en possède une.

3.3.3. Menu Carte Google

Une vue satellite est un outil efficace pour se localiser dans un lieu encore inexploré. L'application que nous avons développer utilise l'API Google Maps pour offrir une carte numérique en ligne sur laquelle on pourra afficher notre position.



3.3.4. Menu à propos

Ce dernier regroupe les informations concernant le développeur de l'application ainsi que ses contacts pour toutes remarques et/ou suggestions.

4. Perspective d'amélioration

Un logiciel doit constamment évoluer dans le temps et être améliorer par ces développeurs puisque ,comme en topographie, il n'est pas absolu. Des bugs que les développeurs n'ont pas rencontrés durant la codage sont toujours présents dissimulés quelque part dans le logiciel. Apporter des mises à jour et optimisation selon l'ère du temps sont donc des impératifs si nous voulons que notre logiciel soit utilisé durant une longue durée.

4.1. Enrichissement du module de calcul

Les calculs présentés dans l'application sont encore sommaire en terme de topographie. Elle illustre seulement les plus couramment utilisé sur terrain lors de lever planimétrique. Mettre un accent sur les calculs altimétriques (calculs de profil) et les calculs de réduction (distances, angulaires) ainsi que l'amélioration de la précision des coordonnées reçu par le GPS serait un grand atout pour la prochaine version du logiciel.

4.2. Amélioration du module de dessin

Nous avons vu dans le mode d'emploi du logiciel que l'espace destiné au croquis est très basique. De plus celui ne supporte que les dessins en deux dimensions. Intégrer le dessin en trois dimensions et une possibilité d'édition manuelle avec mise en couche des différents éléments selon leurs natures est envisageable dans la prochaine mise à jour de l'application.

Conclusion

Dans cet ouvrage, nous avons pu voir comment exploiter les nouvelles technologies, notamment les appareils intelligents fonctionnant sous le système Android, pour nous aider dans nos travaux de terrains. L'objectif est de tirer avantage de ces appareils presque objets du quotidien dont on ignore souvent les capacités.

L'application développé à l'issu de ce mémoire automatise les calculs que l'on rencontre souvent en topographie. Sa conception la rendant simple d'utilisation pour tout topographe jeune tant qu'expérimenté la place en tant qu'outil et compagnon indispensable pour chaque descente, puisque réduit considérablement le temps de travail autant sur le terrain qu'au bureau grâce notamment aux fichiers que peut générer celle-ci (.geo , .txt , .png).

Pour réaliser ce logiciel, il nous a fallu maîtriser les procédés de calcul topographique et aussi apprendre le langage de programmation Java, langage de programmation d'application fonctionnant sous Android. Bien que les difficultés rencontrées furent nombreuses, nous sommes satisfait du résultat que nous avons obtenus durant la réalisation de ce projet. Certes, comme en topographie, le produit que nous avons développé n'est pas absolu et peut encore contenir des bugs. L'apport de mise à jour régulière est un impératif.

Enfin, cette application peut être considéré parmi les premiers produits destinés au appareil mobile et à la topographie à Madagascar. Nous espérons que cet ouvrage et ce travail puisse inspirer les élèves de la mention Information Géographique et Aménagement du territoire, ainsi que tous ingénieurs topographes à approfondir ce domaine encore inexploré, vu la capacité des smartphones de nos jours.

Références

Webographie :

[1] : http://www.commentcamarche.net/contents/java-2132469458

[2] : https://fr.wikipedia.org/wiki/Java_(langage)

[3] : https://fr.wikipedia.org/wiki/Android

[4] : https://openclassrooms.com/courses/developpez-une-application-pour-android

[5] : https://developer.android.com/index.html

[6] : https://console.developers.google.com/

[12] : http://developer.android.com/sdk/index.html

Forum : http://stackoverflow.com/questions/tagged/android

Bibliographie :

[7] : Jean-Baptiste HENRY, «Cours de Topographie et Topométrie Générale », Chapitre 2, Université Louis Pasteur Strasbourg

[8] : Serge MILLES, « Topographie et topométrie modernes », Tome1, Paris 1999

[9] : Serge MILLES, « Topographie et topométrie modernes », Tome2, Paris 1999

[10] : Misan'ny Farany Nirina ANDRIANARISON, Tout sur la projection Laborde et l'utilisation du GPS à Madagascar, Mémoire de fin d'études, 2008

[11] : Tolotra Nantenaina MAMILALA, Elaboration d'un logiciel de calcul topographiques, Mémoire de fin d'études, 2010

Annexes

Annexe 1 : Extrait de code source Java

```
private double[] calculRecoupement(List<Teboka> pointrel, double[] lecrel,double[]
harel,double[]hvrel, double[]vrel,List<Teboka> pointint,double[]goint,double[]lecint,double[]
haint,double[]hvint, double[]vint){

    double X,Y,Z;
    double[][]matAInt,matARel;
    double []DInt,DRel ;

    if(pointrel.size()<=pointint.size()){
        //intersection
        double gi1 = (goint[0]+lecint[0])*Math.PI/200;
        double gi2 = (goint[1]+lecint[1])*Math.PI/200;
        double dxi = pointint.get(0).getX()-pointint.get(1).getX();
        double dyi = pointint.get(0).getY()-pointint.get(1).getY();
        Y = pointint.get(0).getY()+ (dxi-dyi*Math.tan(gi2))/(Math.tan(gi2)-Math.tan(gi1));
        X = pointint.get(0).getX()+(Y-pointint.get(0).getY())*Math.tan(gi1);
    } else {
        //relevement
        double hab = Refy.controleAngle(lecrel[1] - lecrel[0]);
        double hac = Refy.controleAngle(lecrel[2] - lecrel[0]);
        double ctab = 1 / Math.tan(GPSLaborde.gradtorad(hab));
        double ctac = 1 / Math.tan(GPSLaborde.gradtorad(hac));
        double dxab = Teboka.dx(pointrel.get(0), pointrel.get(1));
        double dxac = Teboka.dx(pointrel.get(0), pointrel.get(2));
        double dxbc = Teboka.dx(pointrel.get(1), pointrel.get(2));
        double dyab = Teboka.dy(pointrel.get(0), pointrel.get(1));
        double dyac = Teboka.dy(pointrel.get(0), pointrel.get(2));
        double dybc = Teboka.dy(pointrel.get(1), pointrel.get(2));
        double tgrla = (dxab * ctab - dxac * ctac + dybc) / (dyab * ctab - dyac * ctac - dxbc);
        double grla = Math.atan(tgrla);
        double tgrlb = (tgrla + Math.tan(GPSLaborde.gradtorad(hab))) / (1 - (tgrla *
(Math.tan(GPSLaborde.gradtorad(hab)))));
        double grlb = Math.atan(tgrlb);
        double dxrl = pointrel.get(0).getX() - pointrel.get(1).getX();
        double dyrl = pointrel.get(0).getY() - pointrel.get(1).getY();
        Y = pointrel.get(0).getY() + (dxrl - dyrl * Math.tan(grlb)) / (Math.tan(grlb) -
Math.tan(grla));
        X = pointrel.get(0).getX() + (Y - pointrel.get(0).getY()) * Math.tan(grla);
    }        //fin point approché

    Teboka Mo = new Teboka();
    Mo.setX(X);
    Mo.setY(Y);
    double xmo = Mo.getX();
    double ymo = Mo.getY();

                                //debut matrice int
    if(pointint.size()==0){
        matAInt = new double[0][0];
        DInt = new double [0];
    } else {
        matAInt = new double[pointint.size()][3];
        for (int i = 0; i < pointint.size(); i++) {
            Teboka pi = pointint.get(i);
            matAInt[i][0] = Math.cos(GPSLaborde.gradtorad(Teboka.gisement(pi, Mo))) /
Teboka.distance(pi, Mo);
            matAInt[i][1] = -Math.sin(GPSLaborde.gradtorad(Teboka.gisement(pi, Mo))) /
Teboka.distance(pi, Mo);
            matAInt[i][2] = 0;
        }
        DInt = new double[pointint.size()];
        for (int i = 0; i < pointint.size(); i++) {
            Teboka pi = pointint.get(i);
            DInt[i] = GPSLaborde.gradtorad(Refy.controleAngle(goint[i] + lecint[i]) -
Teboka.gisement(pi, Mo));
        }
    }
                                //fin matrice int
```

```
                                //debut matrice rel
    if(pointrel.size()==0){
        matARel = new double[0][0];
        DRel = new double [0];
    } else {
        matARel = new double[pointrel.size()][3];
        for (int i = 0; i < pointrel.size(); i++) {
            Teboka pi = pointrel.get(i);
            matARel[i][0] = -Math.cos(GPSLaborde.gradtorad(Teboka.gisement(Mo, pi))) /
Teboka.distance(Mo, pi);
            matARel[i][1] = Math.sin(GPSLaborde.gradtorad(Teboka.gisement(Mo, pi))) /
Teboka.distance(Mo, pi);
            matARel[i][2] = -1;
        }
        Teboka p0 = pointrel.get(0);
        double goo = Refy.controleAngle(Teboka.gisement(Mo, p0) - lecrel[0]);
        DRel = new double[pointrel.size()];
        for (int i = 0; i < pointrel.size(); i++) {
            Teboka pi = pointrel.get(i);
            DRel[i] = GPSLaborde.gradtorad(Refy.controleAngle(goo + lecrel[i]) -
Teboka.gisement(Mo, pi));
        }
    }
                                //fin matrice rel

    double[][] matA = concatD(matARel,matAInt);
    double[] D = concat(DRel,DInt);

    RealMatrix coeff = new Array2DRowRealMatrix(matA,false);
    RealVector constants = new ArrayRealVector(D, false);
    DecompositionSolver solver = new SingularValueDecomposition(coeff).getSolver();
    RealVector solution = solver.solve(constants);
    double dx = solution.getEntry(0);
    double dy = solution.getEntry(1);

    Teboka def = new Teboka();
    def.setX(xmo+dx);
    def.setY(ymo+dy);

                                //altitude intersection
    double zi = 0;
    if(pointint.size()==0) {
        zi = 0;
    }else {
        for (int i = 0; i < pointint.size(); i++) {
            zi = zi + pointint.get(i).getZ() + (haint[i] - hvint[i] +
Teboka.distance(pointint.get(i),def)/Math.tan(vint[i] * Math.PI / 200));
        }
    }
                                //altitude relevement
    double zr = 0;
    if(pointrel.size()==0) {
        zr = 0;
    }else {
        for (int i = 0; i < pointrel.size(); i++) {
            zr = zr + pointrel.get(i).getZ() - (harel[i] - hvrel[i] +
Teboka.distance(pointrel.get(i),def)/Math.tan(vrel[i] * Math.PI / 200));
        }
    }

    if(pointint.size()==0){
        Z = zr/pointrel.size();
    }else if(pointrel.size()==0){
        Z = zi/pointint.size();
    }else{
        Z = (zr/pointrel.size()+zi/pointint.size())/2;
    }

    double coord[] = {xmo,ymo,dx,dy,xmo+dx,ymo+dy,Z};
    return coord;
}
```

Annexe 2 : Extrait de code source XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout

        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center|left">

        <TextView
            style="@style/textitalic"
            android:text="Intitulé : "/>

        <EditText
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:id="@+id/editTextNomChem"
            android:inputType="text"/>

    </LinearLayout>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center|left">

        <TextView
            style="@style/textitalic"
            android:text="Type : "/>

        <Spinner
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:id="@+id/spinner_type_chem_session"/>

    </LinearLayout>

</LinearLayout>
```

Annexe 3 : Les avantages du système Assisted-GPS

L'*Assisted GPS*, ou A-GPS, est une technique pour améliorer le positionnement par satellites. L'A-GPS a été introduit pour améliorer la réactivité du GPS, d'où son nom de « GPS assisté ».

Un système A-GPS, ou AGPS ou Assisted GPS ou WAG (Wireless Assisted GPS), utilise les récepteurs GPS de l'opérateur pour aider le terminal mobile à connaître quels signaux GPS il doit suivre. Grâce à cette assistance, la recherche de signal effectuée par le terminal est grandement réduite. La durée nécessaire pour la première connexion ou TTFF (Time To First Fix) passe de plusieurs minutes à seulement quelques secondes. De plus, contrairement aux récepteurs GPS traditionnels, le récepteur A-GPS intégré dans le terminal est en mesure de détecter et démoduler des signaux de très faible magnitude.

L'A-GPS permet de diminuer la consommation d'énergie du terminal GPS, l'envoi des éphémérides par le réseau cellulaire est moins gourmand que le téléchargement de ces données depuis les satellites. Cela permet aussi de réaliser des positionnements rapides à l'allumage. Et avec EGNOS *(*European Geostationary Navigation Overlay Service*)* et sa correction différentielle, l'A-GPS permet d'augmenter la précision : on atteint 3 à 5 mètres en extérieur, qu'on est à 10 mètres avec le système GPS. Et avec les données d'assistance fréquentielle, la sensibilité est améliorée. Ce système A-GPS est donc plus performant que celui fourni en standard sur certain appareil, ce dernier ne téléchargeant que les éphémérides, il gagne donc en rapidité de calcul et en consommation d'énergie mais pas d'amélioration de la sensibilité et de la précision**.**

Seul bémol du A-GPS, son obligation d'être lié au réseau d'un opérateur(avec forfait data ajusté) quand le GPS est totalement autonome et gratuit.

Annexe 4 : Plateforme de développement Android Studio

1. Configuration minimale de l'ordinateur

Toute personne désireuse de développer une application Android doit avoir un ordinateur muni d'un système d'exploitation avec au minimum :

- 2 Go de mémoire RAM
- 500 MB pour installer Android Studio et 1.5 GB de plus pour le Android SDK sur le disque dur
- Processeur double cœur
- Java Development Kit 7 ou plus installé
- Un écran avec une résolution minimum de 1280*800

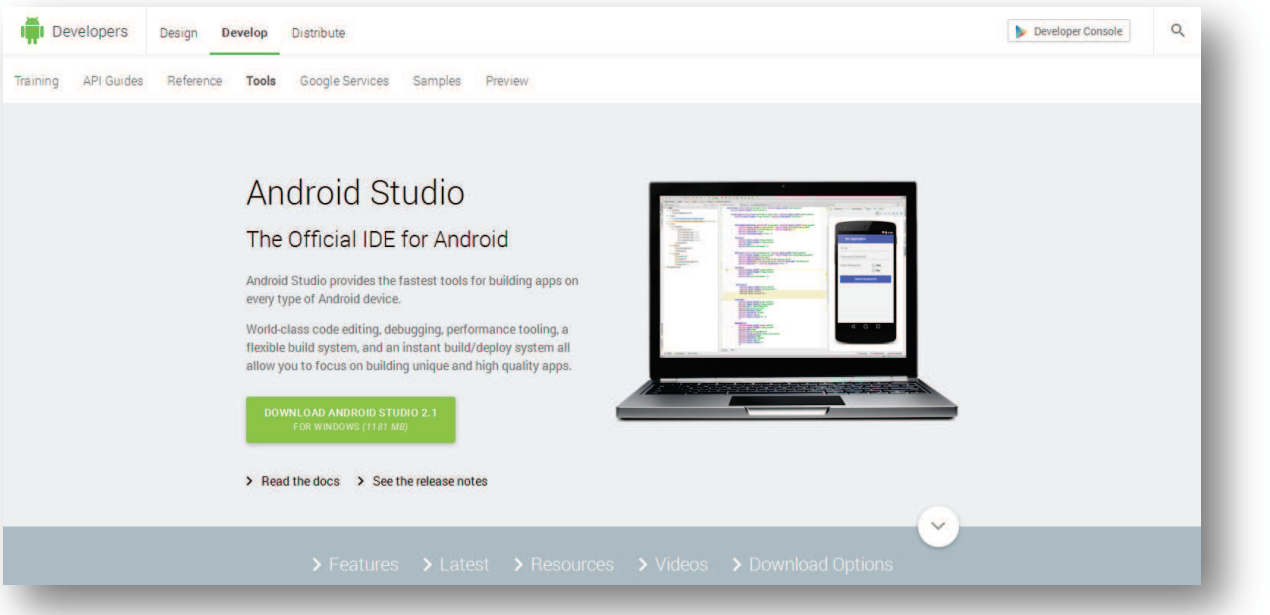
2. Présentation du logiciel Android Studio

2.1. Téléchargement

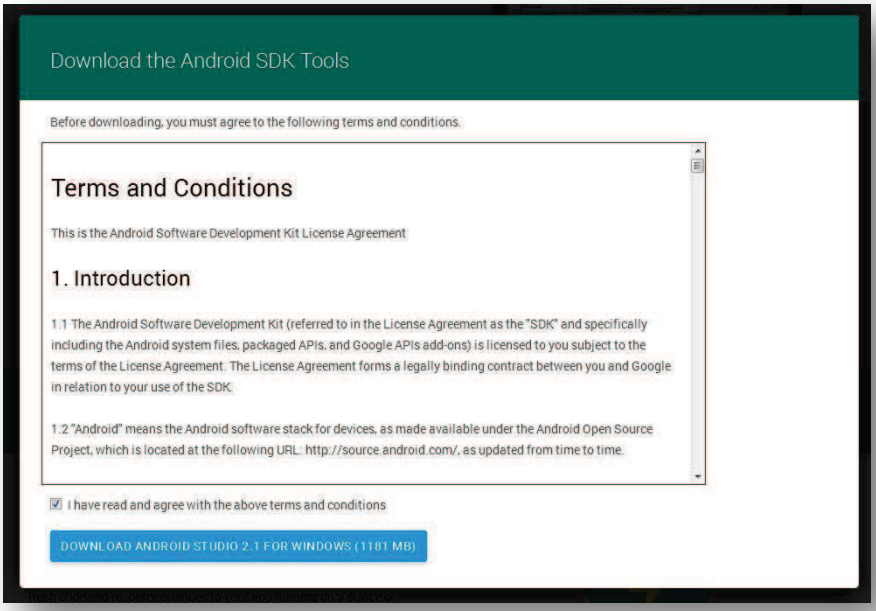
Android Studio est l'environnement de développement pour les applications Android. Il permet principalement d'éditer les fichiers Java et les fichiers de configuration de l'application.

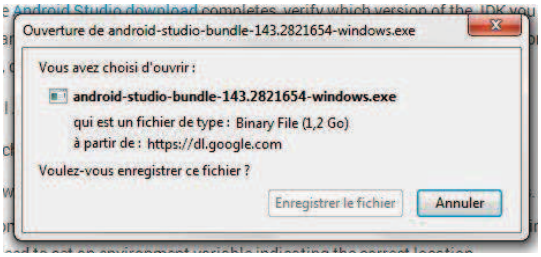
Il est constitué entre autres des outils pour gérer le développement d'applications multilingues et permet de visualiser la mise en page des écrans sur des écrans de résolutions variées simultanément. Le logiciel est soumis à la license Apache 2.0

Pour le télécharger, il faut se rendre sur le site internet dédié aux développeur Android[12]. On accède alors à la page suivante :



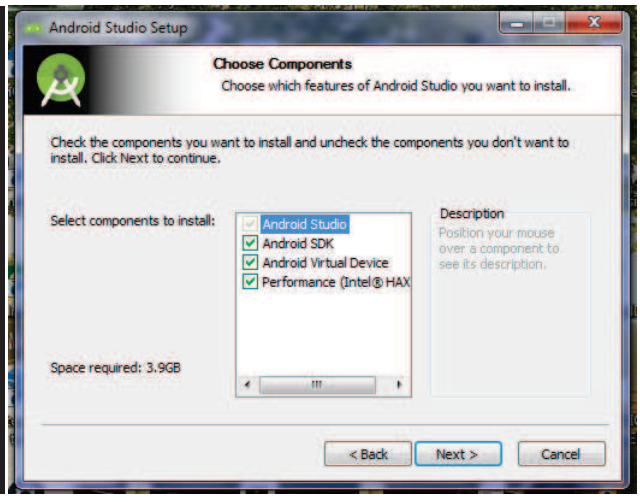
On clique ensuite sur le bouton suivant et le téléchargement commence après avoir accepté les conditions d'utilisation :

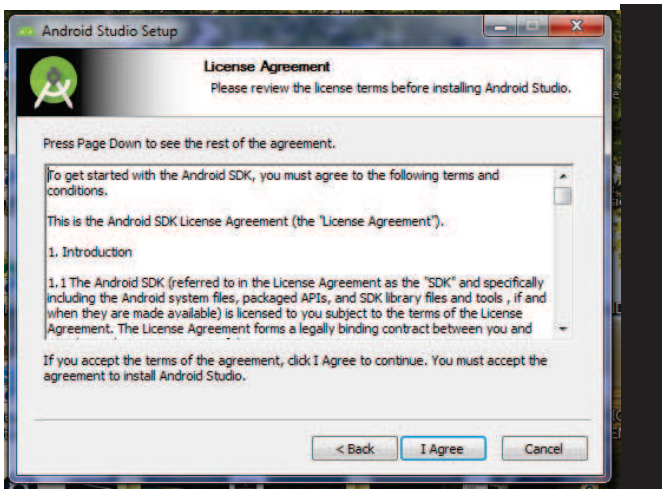


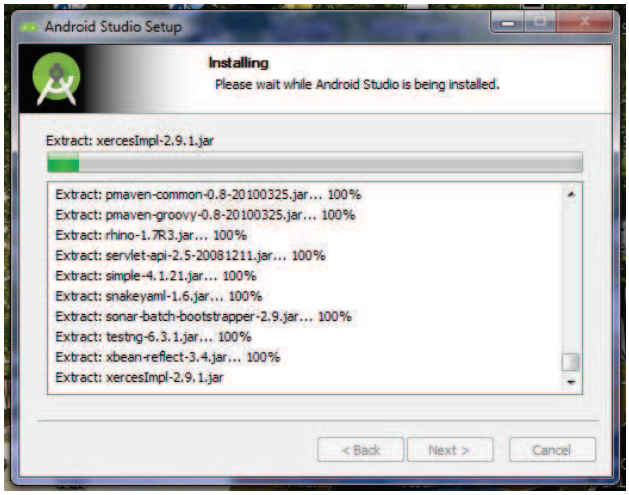


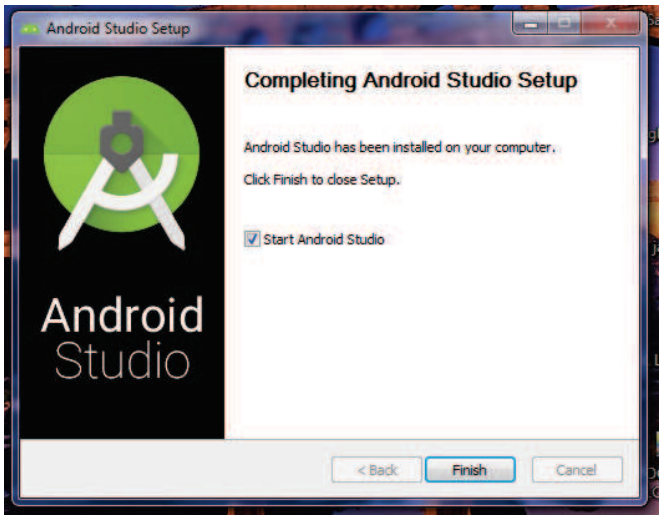
2.2. Installation

On exécute ensuite le fichier téléchargé et le logiciel s'installe qui installe en même temps le Software Development Kit (SDK) Android. Il faut que l'ordinateur soit connecté à internet pendant l'installation.

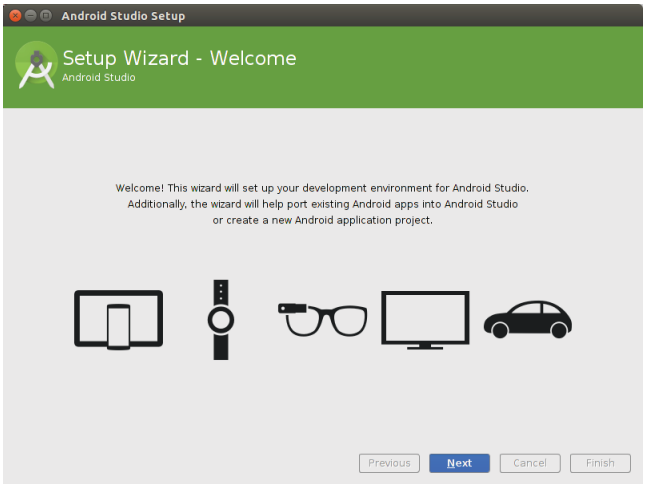


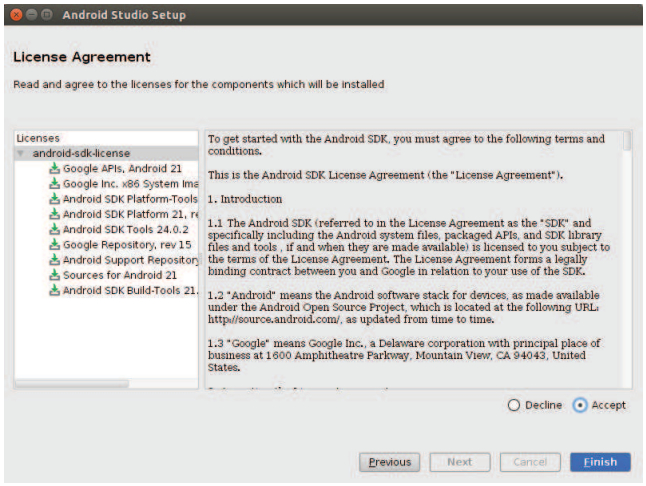


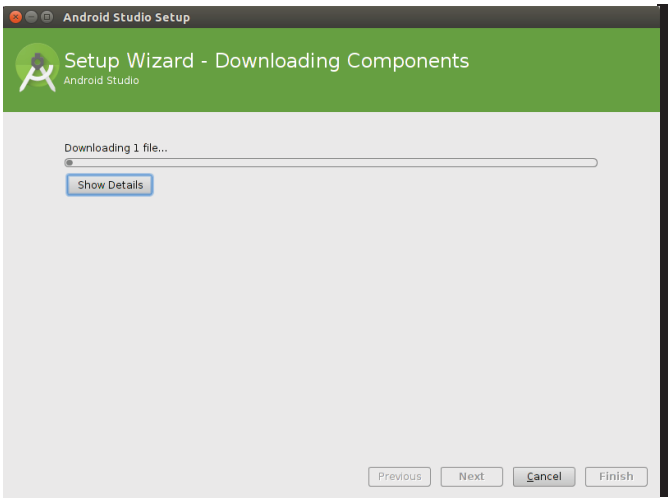


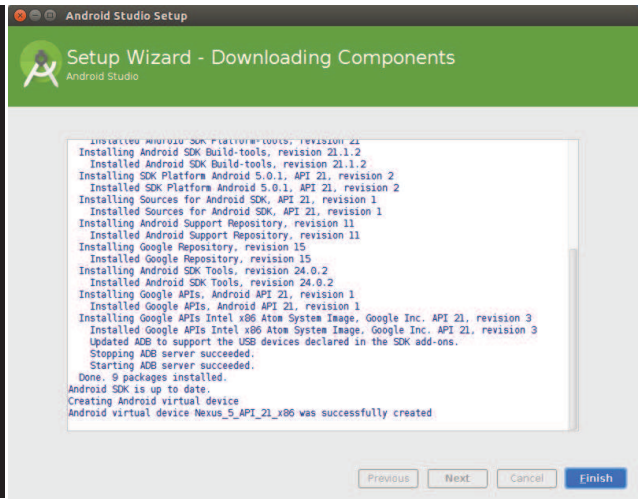


Une fois installé, le logiciel se lance et entame sa phase d'initialisation.

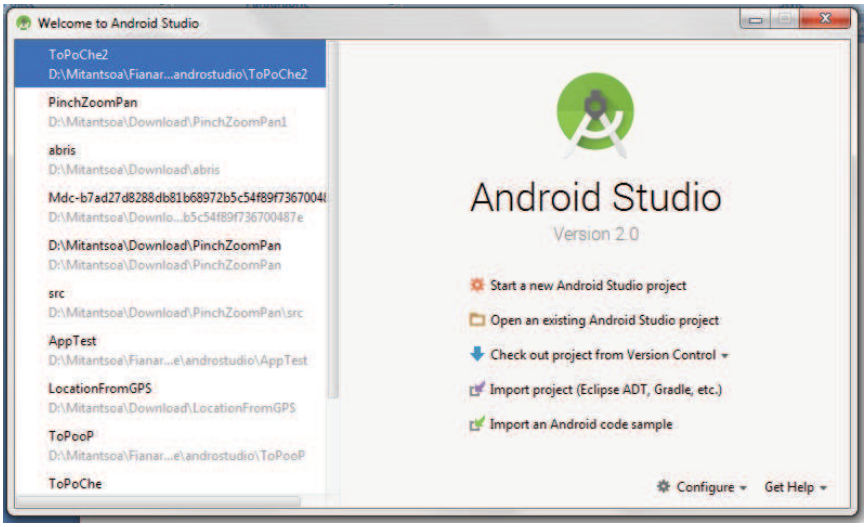






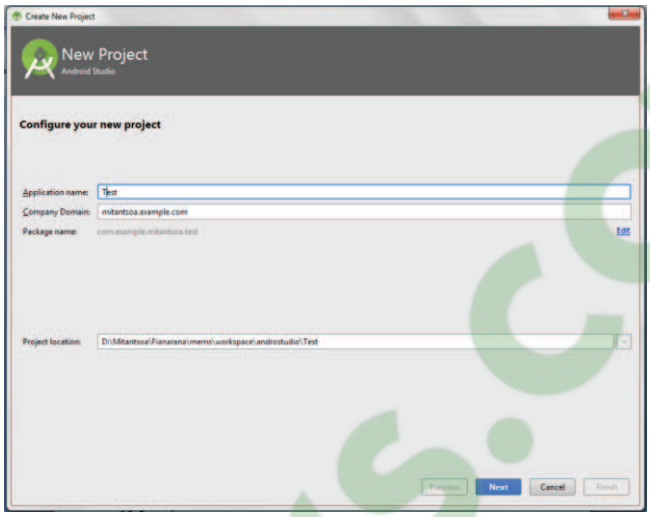


Enfin, la fenêtre de démarrage s'affiche et le logiciel et prêt à fonctionner.



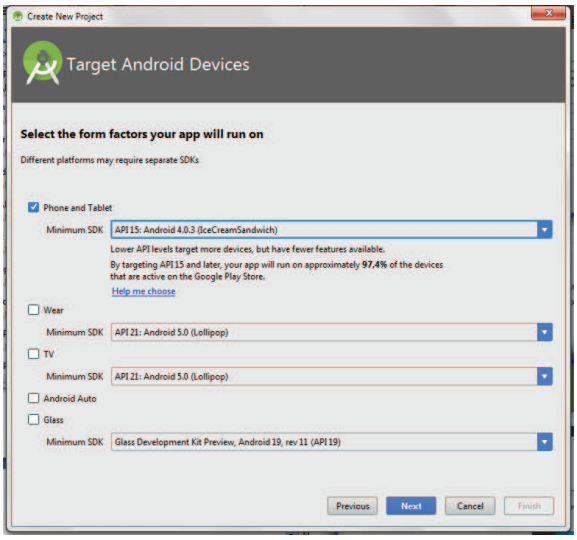
3. Espace de développement

Pour illustrer ce paragraphe, nous allons créer un nouveau projet que nous allons nommé *Test*. On clique sur le bouton *"Start a new Android Studio project"*. La fenêtre suivante apparait et on complète les champs requis.

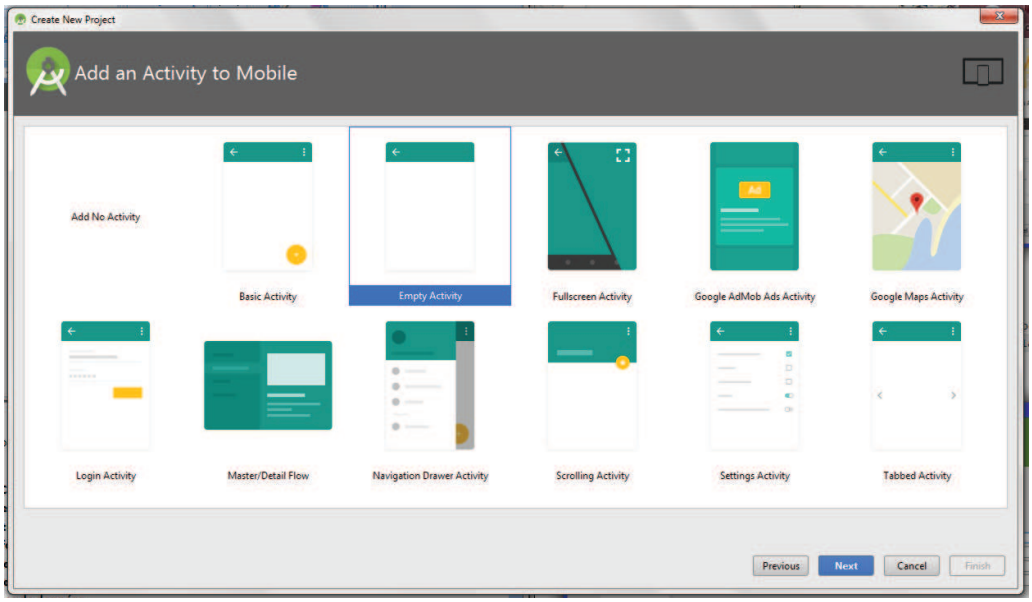


- *Application Name* : il s'agit du nom de l'application
- *Company Domain* : nom de domaine de notre entreprise pour constituer ce champ, Il permet à Android Studio de déduire automatiquement un Package Name c'est-à-dire une chaîne de caractères qui sera utilisée pour déterminer dans quel package se trouvera notre projet. Le package agira comme une sorte d'identifiant pour notre application sur le marché d'applications, alors il doit être unique. De plus, il ne pourra pas être changé une fois votre application publiée.
- *Project location* : correspond à l'emplacement du disque dur où les fichiers du projet seront créés.

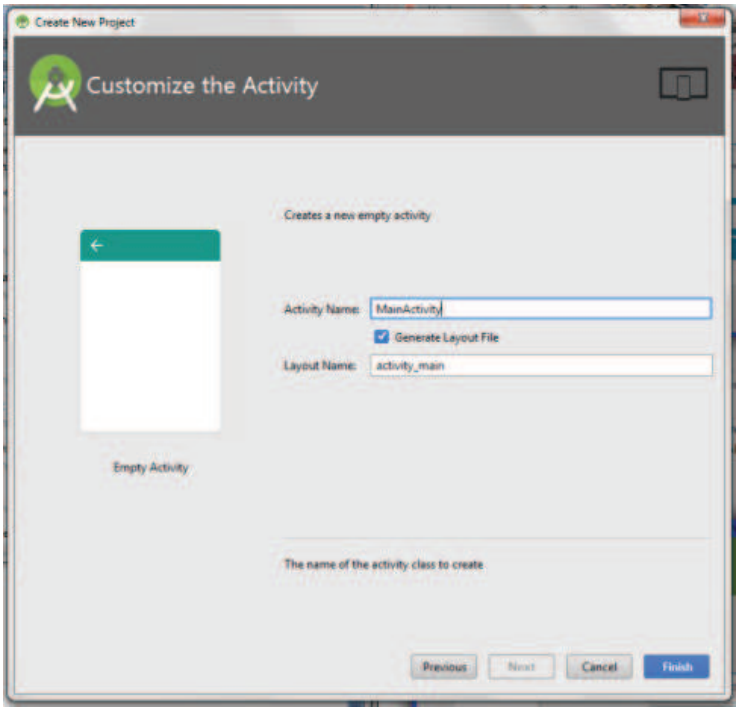
En cliquant sur *Next,* on accède à une nouvelle fenêtre qui permet de définir le matériel de destination de notre application ainsi que la version d'Android minimum que doit utilisé ce même matériel.



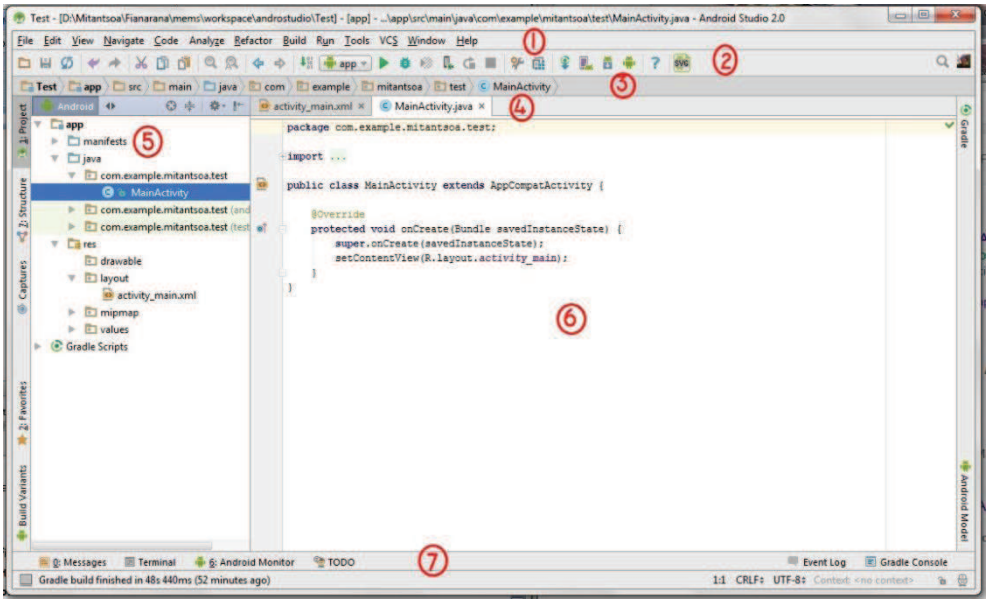
Une fois achevé, on clique encore sur *Next*. La fenêtre qui suit ajoute une nouvelle activité à notre application. On définit alors le type d'interface graphique sur lequel on va travailler notre application.

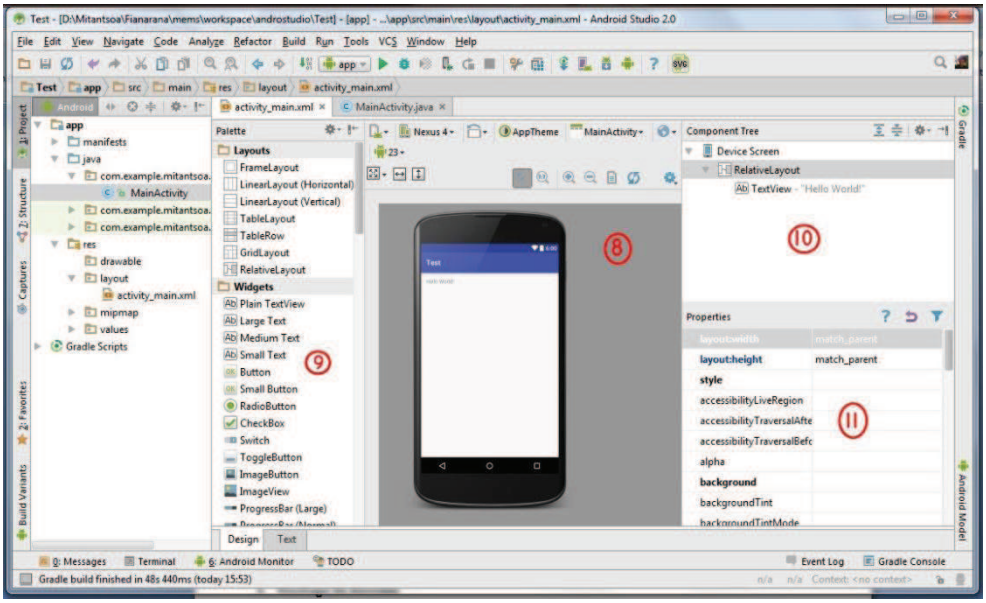


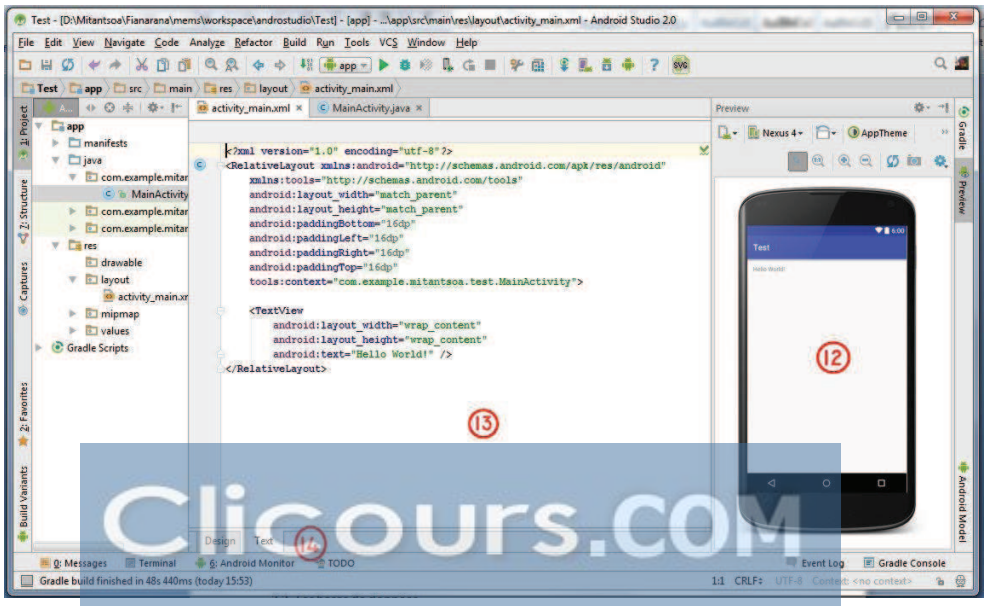
La dernière fenêtre qui apparait après validation permet de nommer l'activité de départ de l'application et de créer le projet.



On accède enfin à la fenêtre de développement proprement dite. Cette fenêtre offre différents types d'affichage selon le type de fichier sur lequel on travaille. Les figures suivantes montrent ces variations.



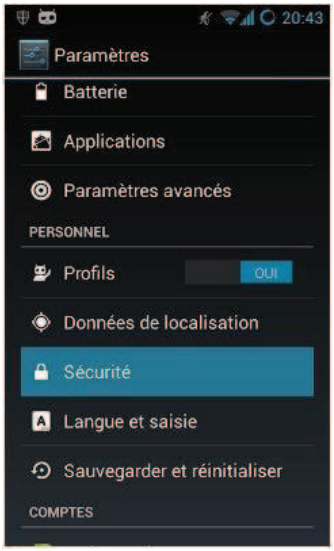


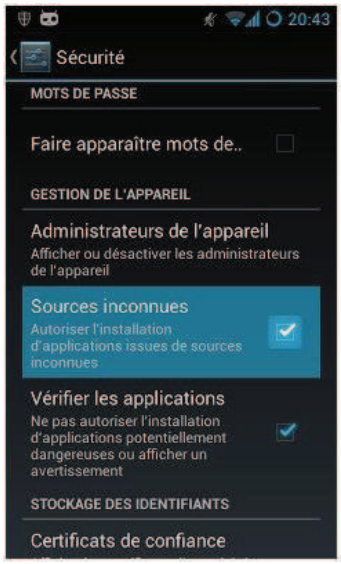


1 : barre de menus
2 : barre d'outils
3 : barre de navigation
4 : onglet de navigation
5 : arborescence du projet
6 : fenêtre de codage Java
7 : barre d'état
8 : éditeur manuelle d'interface graphique
9 : liste des palettes
10 : arborescence des widgets
11 : éditeur de propriétés des widgets
12 : aperçu de l'interface graphique
13 : fenêtre de codage XML
14 : onglet de navigation

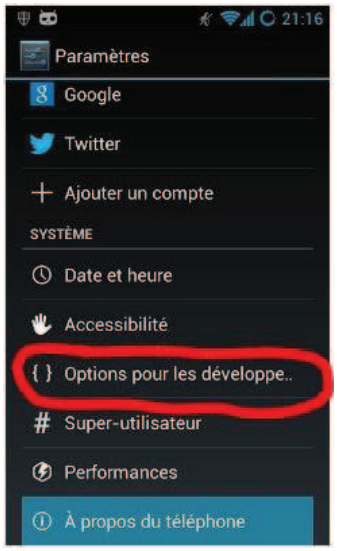
4. Test du code et débogage

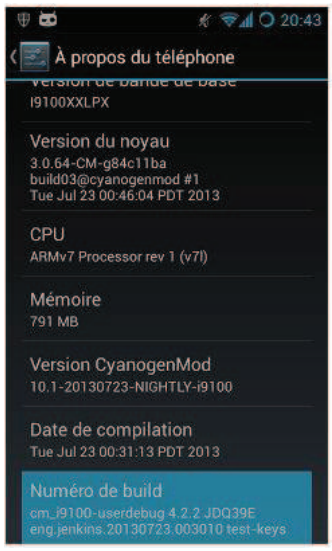
Il existe deux façon de tester une application sur Android Studio. On peut avoir recourt à un appareil virtuel ou alors à un véritable appareil. Pour chaque cas, il faut activer le mode développeur de l'appareil. Pour ce faire, il faut entrer dans les paramètres du téléphone et ouvrir le menu *Sécurité*. On autorise ensuite les applications de sources inconnues à être installé



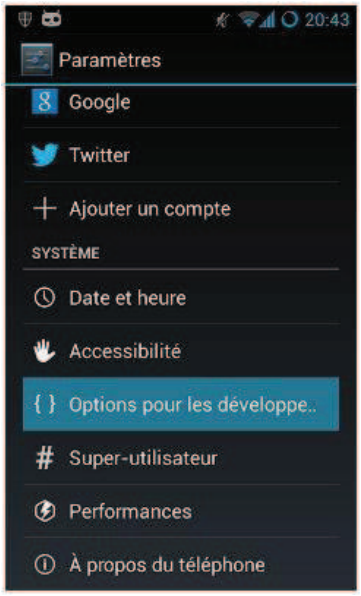


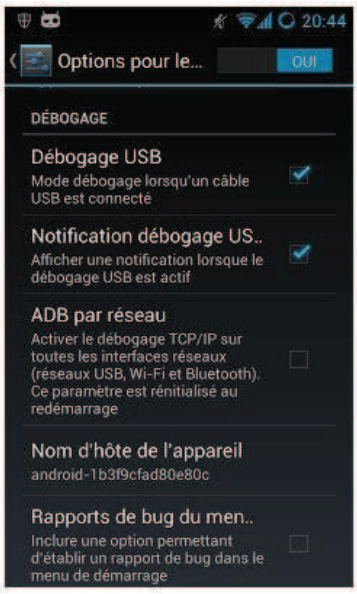
Maintenant, il faut activer le mode développeur. Si ce mode n'est pas encore disponible comme dans l'illustration suivante, il faut l'activer en appuyant sept fois sur le *Numéro de build* du menu *A propos du téléphone*.





Enfin, il faut activer l'option Débogage USB dan le menu *Options pour les développeurs* et notre périphérique sera prêt pour les tests.





HARIMANJATO Mitantsoa Fabrice

Rue Boundry Robert, II M 6 Bis Mahavoky
ANTANANARIVO -101
Tel : +261346494639
e-mail : fabriceharimanjato@gmail.com

Développement d'une application sur mobile, aide aux travaux topographiques de terrains

Résumé :

La topographie à Madagascar ne profite pas pleinement des avantages qu'offre les innovations technologiques actuelles. Le présent mémoire traite de l'élaboration d'une application mobile multifonction d'automatisation de calcul topographique, de localisation et de sauvegarde de données d'observation. Cette application a pour but d'optimiser la durée de travail tant sur terrain qu'au bureau, en exploitant le potentiel des terminaux fonctionnant sous Android. Sauvegarde numérique des données et exploitation de la technologie GPS sont mis à profit dans la conception de ce logiciel mobile.

Mots Clés : *Calculs Topographiques, Carnet de terrain, Android, Java, GPS*

Abstract :

Topography in Madagascar does not profit fully of advantages that the current technology innovations offer. The present report treats development of a multifunction mobile application for topographic automation of calculation, localization and observation data storage. The purpose of this application is to optimize the work period as well on field as at the office, by exploiting the potential of the terminals using Android system. Digital safeguard of the data and exploitation of GPS technology are the main feature of this mobile software.

Keywords : *Topographic Calculation, Field notebook, Android, Java, GPS*

Nombres de pages : 93
Nombres de tableaux : 02
Nombres de figures : 26

Encadreurs:

Mr RABETSIAHINY
Maître de conférence à l'Ecole Supérieure Polytechnique d'Antananarivo

Mr RABEMALAZAMANANA
Enseignant à l'Ecole Supérieure Polytechnique d'Antananarivo