On Mobile Augmented Reality Applications based on Geolocation

Thibaud Michel

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel du 25 mai 2016

Préparée au sein de **L'Institut National de Recherche en Informatique et en Automatique (INRIA), Laboratoire d'Informatique de Grenoble (LIG), Laboratoire de recherche Grenoble, Images, Parole, Signal, Automatique (GIPSA-LAB)**
et de l'école doctorale **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

On Mobile Augmented Reality Applications based on Geolocation

Présentée par

Thibaud MICHEL

Thèse dirigée par **Nabil LAYAÏDA**
et codirigée par **Hassen FOURATI et Pierre GENEVÈS**

Thèse soutenue publiquement le **10 Novembre 2017**,
devant le jury composé de :

Madame Valérie RENAUDIN	DR	IFSTTAR	Rapporteur
Monsieur Takeshi KURATA	Prof.	AIST, Japan	Rapporteur
Monsieur Oliver RUEPP	Ph.D	Apple Inc., U.S.	Examinateur
Monsieur Pierre GENEVÈS	CR	CNRS	Co-Directeur de thèse
Monsieur Hassen FOURATI	MCF	GIPSA-Lab	Co-Directeur de thèse, Invité
Monsieur Nabil LAYAÏDA	DR	Inria	Directeur de thèse
Madame Laurence NIGAY	Prof.	LIG	Président

Abstract

Applications for augmented reality can be designed in various ways, but few take advantage of geolocation. However, nowadays, with the many cheap sensors embedded in smartphones and tablets, using geolocation for augmented reality (Geo AR) seems to be very promising. In this work, we have contributed on several aspects of Geo AR: estimation of device positioning and attitude, and the impact of these estimations on the rendering of virtual information.

In a first step, we focused on smartphone attitude estimation. We proposed the first benchmark using a motion lab with a high precision for the purpose of comparing and evaluating filters from the literature on a common basis. This allowed us to provide the first in-depth comparative analysis in this context. In particular, we focused on typical motions of smartphones when carried by pedestrians. Furthermore, we proposed a new parallel filtering technique for limiting the impact of magnetic perturbations with any attitude estimation algorithm used in this context. We showed how our technique compares and improves over previous works.

In a second step, we studied the estimation of the smartphone's position when the device is held by a pedestrian. Although many earlier works focused on the evaluation of localisation systems, it remains very difficult to find a benchmark to compare technologies in the setting of a commodity smartphone. Once again, we proposed a novel benchmark to analyse localisation technologies including WiFi fingerprinting, WiFi trilateration, SHS (Step and Heading System) and map-matching.

In a third step, we proposed a method for characterizing the impact of attitude and position estimations on the rendering of virtual features. This made it possible to identify criteria to better understand the limits of Geo AR for different use cases.

We finally proposed a framework to facilitate the design of Geo AR applications. We show how geodata can be used for AR applications. We proposed a new semantics that extends the data structures of OpenStreetMap. We built a viewer to display virtual elements over the camera livestream. The framework integrates modules for geolocation, attitude estimation, POIs management, geofencing, spatialized audio, 2.5D rendering and AR. Three Geo AR applications have been implemented using this framework. TyrAr is an application to display information on mountain summits and cities around the user. AmiAr allows one to monitor lights, shutters, tv in a smart apartment. Venturi Y3 is an AR-Tour of Grenoble with audio description and experiences.

Résumé

Les applications de réalité augmentée (RA) peuvent être conçues de différentes manières, mais encore très peu tirent parti de la géolocalisation. Pourtant, aujourd'hui, avec la multitude de capteurs embarqués dans nos smartphones et nos tablettes, l'utilisation de la RA Géolocalisée (RA Geo) semble très prometteuse. Dans ce travail de thèse, nous avons contribué sur plusieurs aspects fondamentaux de la RA Geo: l'estimation de la position du dispositif, l'estimation de son orientation, ainsi que sur l'impact de ces estimations sur le rendu des informations virtuelles.

Dans un premier temps, nous avons étudié l'estimation de l'orientation du téléphone. Nous avons réalisé le premier benchmark dans un laboratoire de capture de mouvements afin de comparer et d'évaluer les différents filtres de la littérature sur une base commune. Celui-ci a permis de produire la première analyse comparative approfondie des différentes approches. Nous avons aussi proposé un nouveau filtre dont le but est de minimiser l'effet des perturbations magnétiques omniprésentes en intérieur et avons quantifié son apport par rapport aux techniques existantes.

Dans un deuxième temps, nous avons étudié l'estimation de la position du téléphone lorsqu'il est porté par un piéton. Il existe de nombreux travaux sur les technologies de localisation mais il est difficile de trouver un benchmark qui évalue les différentes technologies dans le cas d'un smartphone grand public. Ainsi, à nouveau, nous avons produit le premier benchmark pour analyser de façon comparative les différentes techniques de localisation qui peuvent être utilisées avec un smartphone.

Dans un troisième temps, nous avons proposé une méthode pour caractériser l'impact des estimations d'orientation et de positionnement sur le rendu des informations virtuelles. Ceci a permis d'identifier des critères permettant de comprendre plus précisément les limites de faisabilité de différent cas d'utilisation de la RA Geo.

Enfin, nous avons proposé un nouveau framework pour faciliter la conception d'applications de la RA Geo. Nous montrons comment les données de cartographie peuvent être utilisées et enrichies à l'aide d'une nouvelle sémantique utilisant OpenStreetMap. Nous avons créé un visualiseur permettant d'afficher des éléments virtuels sur le flux de la caméra. Ce framework intègre différents modules pour la localisation, l'orientation, la gestion des points d'intérêts, le geofencing, l'audio spatialisé, et le rendu 2D ou RA. Finalement, trois exemples d'applications d'AR Géo ont été réalisés à partir de ce framework. TyrAr est une application pour visualiser des informations sur les sommets et les villes environnantes. AmiAr permet de contrôler certains objets de domotique dans un appartement connecté. Venturi Y3 est une visite guidée de la ville de Grenoble avec des expériences de RA avec de l'audio.

Acknowledgements

First of all, I would like to thank my supervisor: Nabil Layaïda and my co-supervisors: Pierre Genevès and Hassen Fourati which allowed me to agreeably integrate Tyrex and Necs teams.

Besides, I would also like to thank the members of my jury, my reviewers: Valérie Renaudin and Pr. Takeshi Kurata as well as my examiners: Pr. Laurence Nigay and Oliver Ruepp to have reviewed my work, provided me encouraging and constructive feedback.

This work was carry out thanks to the support of LabEx PERSYVAL-Lab (ANR–11-LABX-0025), EquipEx KINOVIS (ANR-11-EQPX-0024) and EquipEx AmiQual4HOME (ANR-11-EQPX-0002).

Furthermore, I would like to thank all my colleagues which helped me technically in my work:

- J.F. Cuniberto for the design of the smartphone handler.
- J. Dumont and M. Heudre for showing me how the Vicon and Qualisys system work.
- C. Roisin, L. Carcone, N. Gesbert, T. Calmant, G. Dupraz-Canard, D. Graux, L. Jachiet, J. Zietsch for the help in recording sensors data.
- J. Zietsch for having shared with me a part of this project by studying the WiFi-Fingerprinting approach.
- R. Pincent and N. Bonnefond for helping me to setup the UWB system and sharing with me your experience with smart objects.
- M. Razafimahazo for providing the iOS app to record smartphone's sensors, and mainly for long discussions about implementation of AR browser on real devices.

and all others colleagues from Tyrex and Necs teams and more generally from Inria.

On a more personal note, I would like to thank my family and principally Fanny for their unconditional support.

Contents

Abstract i

Résumé iii

Acknowledgements v

Table of Contents x

List of Figures xiii

List of Tables xvi

Introduction 1

1 State of the Art 5
- 1.1 Coordinate Systems . . . 6
  - 1.1.1 Smartphone . . . 7
  - 1.1.2 World Geodetic System (WGS) . . . 7
  - 1.1.3 Earth-Centered, Earth-Fixed (ECEF) . . . 8
  - 1.1.4 Local Tangent Plane (LTP) . . . 9
  - 1.1.5 OpenGL . . . 9
- 1.2 Attitude Estimation . . . 10
  - 1.2.1 Attitude Representation . . . 10
  - 1.2.2 Allan Variance of the Smartphone IMU . . . 11
  - 1.2.3 Attitude Estimation . . . 14
  - 1.2.4 Evolution of Techniques over the Years . . . 16
- 1.3 Device Geolocation . . . 18
  - 1.3.1 Geolocation Techniques . . . 18
  - 1.3.2 Map to Improve Location Estimates . . . 26
  - 1.3.3 Fused Approaches . . . 27
  - 1.3.4 Surveys and Evaluations of Geolocation Techniques . . . 28
- 1.4 Geo AR Browsers . . . 29
  - 1.4.1 Formats and Authoring . . . 29
  - 1.4.2 Frameworks . . . 30

1.4.3 Evaluation of Geo AR Systems 34

2 Attitude Estimation: Benchmarks and Improvements 35
2.1 Experimental Protocol in a Motion Lab 36
2.1.1 Ground Truth 36
2.1.2 Typical Smartphone Motions 37
2.1.3 Introducing Magnetic Perturbations 37
2.1.4 Different Devices 39
2.1.5 Common Basis of Comparison and Reproducibility 40
2.2 Attitude Estimation Algorithms Selected for Comparison 41
2.3 Design of a New Algorithm for Limiting Magnetic Perturbations Impact 42
2.4 A Deep Analysis of Results 43
2.4.1 Importance of Calibration 44
2.4.2 The Difficulty with Noises for Kalman Filters 45
2.4.3 Bias Consideration 46
2.4.4 Behaviors during Typical Smartphone Motions 47
2.4.5 Impact of Magnetic Perturbations 47
2.4.6 Pitch and Roll in Augmented Reality Scenario 48
2.4.7 Comparison with Device-Embedded Algorithms 50
2.4.8 Empirical Computational Complexity 51
2.4.9 Relevant Sampling Rates 51
2.4.10 Parameter Adjustment for a Balance Between Stability and Precision . . 52
2.5 Conclusions 54

3 An Experimental Protocol to Evaluate Device Positioning Approches 57
3.1 A Set of Applications to Record Sensors Data and Create a Ground Truth 58
3.1.1 Senslogs - A Core App to Record Data from Built-in Sensors 58
3.1.2 GTR4SL - Ground Truth Recorder for Sensor Localization 61
3.1.3 Fingerprinting Offline App 62
3.2 An Experimental Protocol to Score and Analyze Navigation Algorithms 63
3.2.1 A Testbed for Navigation Algorithms with Smartphone 63
3.2.2 Analyze on Vector Maps 64
3.3 Technologies Evaluated 64
3.3.1 WiFi Fingerprinting 65
3.3.2 WiFi Trilateration 65
3.3.3 Step and Heading System 65
3.3.4 SHS + Map-Matching 66
3.3.5 GNSS 66
3.3.6 UWB 66
3.4 Benchmark: A Trade Of Between External Data and Technologies 66
3.4.1 Testbed Context 66
3.4.2 Preprocessing phase 67
3.4.3 Context: Indoor vs Outdoor 68
3.4.4 Smartphone Orientation: Fixed vs Free 69
3.4.5 Starting Position: Known vs Unknown 70

3.4.6 Map: With vs Without . . . 70
3.4.7 WiFi: Sampling Rates and Fingerprints . . . 70
3.4.8 SHS . . . 72
3.5 Conclusions and Perspectives . . . 73

4 An Evaluation Method for Augmented Reality 75
4.1 Introduction . . . 76
4.2 An Evaluation Method to Calculate Distance and Angle Errors . . . 77
4.2.1 Attitude Estimation Model . . . 77
4.2.2 Position Estimation Model . . . 78
4.2.3 Attitude + Position Estimation Model . . . 79
4.3 Projected Distance on the Screen . . . 80
4.4 Scores from our Benchmarks: Examples on Different Use Cases . . . 81
4.4.1 Use Case 1: An Application to Identify Mountains and Cities . . . 82
4.4.2 Use Case 2: An Application to Discover the History of a City . . . 83
4.4.3 Use Case 3: Make 3D Models Appear and Turn Around in an Indoor Environment . . . 84
4.4.4 Use Case 4: Identify and Interact with Objects in a Building using UWB 84
4.5 Conclusion . . . 85

5 On a Geo AR Browser Framework 87
5.1 OSM for AR Documents . . . 88
5.1.1 A Format over OSM Specifications . . . 88
5.1.2 JOSM: A Fast Authoring Tool for Geo AR . . . 91
5.2 Geo AR Viewer: From Reality to Virtual World . . . 91
5.2.1 Positioning Features in the OpenGL Scene . . . 92
5.2.2 Orientation: from Sensors to OpenGL . . . 93
5.2.3 Camera stream: Field Of View and Aspect Ratio . . . 95
5.2.4 Scheduling and Updating the OpenGL Scene . . . 96
5.3 AR Browser Framework . . . 97
5.3.1 Mapbox to handle Indoor Vector Maps . . . 98
5.3.2 Core of the Framework . . . 99
5.3.3 AR Browser Applications . . . 101
5.4 Conclusions and Perspectives . . . 105

General Conclusions & Perspectives 107

Bibliography 111

Appendices 123

Appendix A Contributions **125**
A.1 Pseudo-code of our SHS approach 125
A.2 Description of Outputs from Senslogs App 126
A.3 Source Code of the JOSM Map Paint Style for Augmented Reality 128

List of Figures

0 Geo Augmented Reality. 1

1.1 Geo Augmented Reality Overview. 6
1.2 Smartphone frame representation. 7
1.3 World Geodetic System representation. 7
1.4 Earth-Centered, Earth-Fixed (ECEF) frame representation. 8
1.5 Local Tangent Plane (LTP) frames representation. 9
1.6 OpenGL frame representation. 10
1.7 The Smartphone-Frame SF (dashed line) and ENU-Frame EF (solid line). 10
1.8 Typical Allan deviation plot for a system [Hou, 2004] 12
1.9 Allan variance of gyroscope signal. 13
1.10 Allan variance of accelerometer signal. 14
1.11 Allan variance of magnetometer signal. 15
1.12 Reference vectors when the smartphone is static and in the absence of magnetic deviations. 16
1.13 General schema for attitude estimation. 16
1.14 Overview of indoor technologies in dependence on accuracy and coverage. [Mautz, 2012] 19
1.15 WiFi fingerprinting system flow. [He and Chan, 2016] 20
1.16 An example of WiFi trilateration with 3 Access Points (AP) 21
1.17 Step and Heading System (SHS) overview 22
1.18 Block diagram of the foot motion tracking algorithm that produces the foot orientation quaternion, foot position, foot velocity, and gait phase in [Fourati, 2015] . . 24
1.19 Estimated velocity for standard INS vs proposed method in [Lakmal and Samarabandu, 2016] 24
1.20 Example of point to network approach for map matching 26
1.21 Touring Machine system overlays AR information in outdoor environments. . . . 31

2.1 Kinovis room at Inria, Grenoble, France. 37
2.2 The eight typical motions for a smartphone. 38
2.3 Magnetic boards for building structure and heaters simulation. 39
2.4 Magnitude of magnetic field measurements and Earth's magnetic field during our simulation with magnetic boards. 39
2.5 Magnitude of magnetic field measurements and Earth's magnetic field in the indoor environment of Inria building in Grenoble. 43

2.6 Pseudo-code for limiting the impact of magnetic perturbations. 44
2.7 Sample run of the reprocessing technique (red) when a magnetic perturbation occurs, in comparison to ground truth (black) and earlier techniques. 49
2.8 From default frame to camera landscape frame (rotation of 90°around x-axis then another rotation of 90°around z-axis) 49
2.9 Relative performance in terms of CPU cost (lower is better). 51
2.10 Spectrum of possibilities in terms of stability/precision in AR with few magnetic perturbations 54
2.11 Spectrum of possibilities in terms of stability/precision in AR with high magnetic perturbations 55

3.1 Screenshots of Senslogs app 59
3.2 Pseudo-code of approach developed to continuously scan WiFi signals. 60
3.3 Screenshots of GTR4SL app 61
3.4 Screenshots of fingerprinting app 62
3.5 Pseudo-code of approach developed to continuously scan WiFi signals. 63
3.6 Common interface for navigation algorithms 63
3.7 Screenshot of our vector map to analyze navigation algorithms. Blue line is the ground truth. Red line is estimated positions. Green lines show distances between estimated and reference positions. 64
3.8 Satellite View (Google Maps) of the two places where benchmarks took place. In red the $15\ 000\ m^2$-building and in blue the $5\ 000\ m^2$-clear space area. 67
3.9 Precision error of WiFi-Fingerprinting approach in function of the number of fingerprints recorded. 71
3.10 The eight typical motions for a smartphone. 72
3.11 Precision error of SHS approaches in function of the time considered from the beginning, when smartphone is fixed in an indoor context. 73

4.1 Representation of errors due to a poor estimation of a virtual feature position. v is the distance between the estimated feature (P') and the real position of the feature (P). The projection of this distance on the screen is named e. 76
4.2 Evaluation model and simulation where $f_{\text{pos}} = 0$ and f_{att} is fixed 77
4.3 Evaluation model and simulation where $f_{\text{att}} = 0$ and f_{pos} is fixed 78
4.4 Evaluation model where f_{att} and f_{pos} are fixed 79
4.5 Projection of distance error on the screen 80

5.1 Hierarchy of features and documents handled by our format. 88
5.2 Comparison of JOSM with and without map paint style for AR 90
5.3 An example of AR application which does not respect camera aspect ratio. Camera stream should render the screen with right angles but here we observe a distortion. 91
5.4 Representation of the 3 frames which deal with Rajawali: ECEF (in green), OpenGL Camera (in red) and a 3D Model (in blue). 93
5.5 OpenGL camera: a succession of rotations from OpenGL frame to ECEF frame. . 94
5.6 Camera feed scaled to fill the size of the view and keep aspect ratio. 95
5.7 Virtual spheres placed at a predefined position on a custom target to verify FOV. . 96

5.8 Pseudo-code of features update triggered by ”device location changed”, ”device attitude changed” or ”screen is rotated” events. 97
5.9 Mapbox indoor process. 98
5.10 Screenshots of Mapbox GL JS with our indoor plugin 99
5.11 Proposed Geo AR Browser Overview. 100
5.12 Screenshots of Tyr-AR application 102
5.13 Screenshots of Smart Home AR application 103
5.14 Screenshots of Venturi Y3 application 104
5.15 Our AR framework at the center of the process for building Geo AR applications. 105
6.1 Geo AR with a UAV 109

List of Tables

1.1 List of Geo AR frameworks which allow to customize camera pose estimation. . 33

2.1 Statistics on Magnitude of External Accelerations for each motion 38
2.2 Statistics on Magnitude of Magnetic Field with low and high magnetic perturbations 39
2.3 Sensors specifications with the max. sampling rate 40
2.4 Precision of attitude estimation according to calibration with all motions 45
2.5 Precision of attitude estimation according to sensor noises without magnetic perturbations. 45
2.6 Precision of attitude according to bias estimation without magnetic perturbations. 46
2.7 Precision of attitude estimation according to typical motions without magnetic perturbations. 47
2.8 Precision of attitude estimation according to typical motions with magnetic perturbations. 48
2.9 Precision of attitude estimation according to Augmented Reality motions with magnetic perturbations. 50
2.10 Precision according to device with all motions and with/without magnetic perturbations. 50
2.11 Precision according to sampling with all motions and with/without magnetic perturbations. 51

3.1 Statistics on WiFi scan interval 60
3.2 Average time spent for the offline phase with each navigation technology in our $15\ 000\ m^2$-building. 68
3.3 Average (AVG) and Standard Deviation (STD) of precision error of navigation algorithms inside and outside. 69
3.4 Comparison of precision error of navigation algorithms in an indoor environment when device is fixed or free. 69
3.5 Precision error of navigation algorithms without the knowledge of starting position with a fixed smartphone in an indoor context. 70
3.6 Precision error of WiFi approaches with different static devices in an indoor context. 71
3.7 Precision error of WiFi approaches with different moving devices in an indoor context. 71
3.8 Precision error of SHS approaches with different attitude algorithms when smartphone is fixed in an indoor context. 72

4.1 Scores of use case 1: an application to identify mountains and cities 83
4.2 Scores of use case 2: an application to discover the history of a city 84
4.3 Scores of use case 3: make 3D models appear and turn around in an indoor environment 84
4.4 Scores of use case 4: identify and interact with objects in a building using UWB . 85

5.1 AR scores for Smart Home AR application after adjusting z-axis with the two finger swipe 103
5.2 AR scores for Smart Home AR application after 1 minute-drift on z-axis 104
5.3 Updated list of Geo AR frameworks with our contribution 106

Introduction

First, the scientific problem addressed is presented. We then introduce the context in which this thesis takes place. Finally, we present the organization of the present dissertation.

Motivation & Problem

The term Augmented Environments refers collectively to ubiquitous computing, context-aware computing, and intelligent environments. The goal of our research on these environments is to introduce personal Augmented Reality (AR) devices, taking advantage of their embedded sensors. Augmented Reality (AR) is a live view of real world environment where virtual objects (features) are shown over the camera image of an handheld device. We believe that personal AR devices such as mobile phones or tablets will play a central role in augmented environments. These environments offer the possibility of using ubiquitous computation, communication, and sensing to enable the presentation of context-sensitive information and services to the user. AR applications often rely on 3D content and employ specialized hardware and computer vision techniques for both tracking and scene reconstruction and exploration.

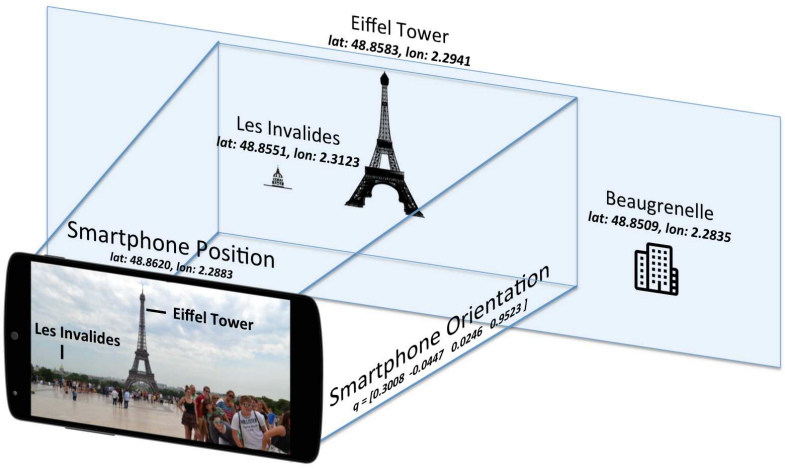


Figure 0: Geo Augmented Reality.

Geo Augmented Reality (Geo AR) is an AR method which allows the user to visualize augmented features exclusively thanks to the device position and orientation (Figure 0). In contrary to AR based on vision, this technique does not use image processing. Compared to AR based on vision, Geo AR is less computational expensive and do not rely on a specific lightning[Messelodi

et al., 2015] and knowledge of the context. GPS, WiFi, Bluetooth or any kind of location sensors can be used to determine device position. The orientation of the device can be computed by an attitude estimation filter using the onboard inertial sensors (gyroscope, accelerometer and magnetometer).

Recently, several companies have started to develop applications which use Geo AR, including Layar[1], Google Sky Map[2], Peak.AR[3]. Based on user feedbacks [Dünser et al., 2012], AR applications using a geolocation approach could be much immersive if they are more accurate. However, no study evaluated the impact of positioning and attitude estimation on the Geo AR rendering.

In this work, we have contributed on several aspects of Geo AR: estimation of device positioning and attitude, and the impact of these estimations on the rendering of virtual information. Precision of attitude estimation is crucial in Geo AR, as features should be seen at the right place on the screen. Precision of device positioning is vital as well, because the pair <position, orientation> (pose) defines where features will be displayed on the screen.

Context

This work is the result of a collaboration between two research teams: Tyrex and NeCS.

The Tyrex team aims at developing a vision of a web where content is enhanced and protected, applications made easier to build, maintain and secure. It seeks to open new horizons for the development of the web, enhancing its potential, effectiveness, and dependability. In particular, the team aims at making contributions by obtaining fundamental results, by building advanced experimental applications showcasing these results and by contributing to web standards. One fundamental problem is a lack of formalisms, concepts and tools for reasoning simultaneously about content or data, programs, and communication aspects. Our main scientific goal is to establish a unifying development framework for designing advanced (robust, flexible, rich, efficient and novel) applications.

The research field of NeCS team deals with feedback systems controlled over networks, but also concerns systems that naturally exhibit a network structure (e.g., traffic, electrical networks, etc.). The team also works on the multi-sensors fusion and estimation in navigation including attitude estimation and geolocation.

Thesis Outline

The rest of the dissertation is divided as follows into five chapters.

Chapter 1

In chapter 1, we focus on the state-of-the-art principles used for Geo AR. We show how the Geo AR approach works and how it is linked to the different modules. Then, we present an

[1] `https://www.layar.com/`

[2] `https://play.google.com/store/apps/details?id=com.google.android.stardroid`

[3] `https://peakar.salzburgresearch.at/`

overview of attitude estimation: we provide a background, analyze sensors and introduce state-of-the-art filters. We also give an overview of positioning estimation: we present the state-of-the-art techniques and show the limits of existing evaluations. Finally, we describe the stack of how Geo AR browsers are built from authoring to rendering.

Chapter 2

In Chapter 2, we studied the smartphone attitude estimation. We observed that there was no common way to evaluate the different techniques that can be used in this specific context. We built the first benchmark which uses a motion lab with a high precision to compare and evaluate filters from the literature on a common basis. This allowed us to provide the first in-depth comparative analysis in this context. In particular, we focused on typical motions of smartphones when carried by pedestrians. Furthermore, we proposed a new technique for limiting the impact of magnetic perturbations with any attitude estimation algorithm used in this context. We showed how our technique compares and improves over previous works.

Chapter 3

In Chapter 3, we studied the smartphone positioning when the device is held by a pedestrian. GNSS (Global Navigation Satellite System) is a widespread technology which is mostly reliable when used outdoor: accuracy is greatly reduced when the user is indoor or when buildings obstruct the satellite reception (for example in a city center). Many works focused on indoor localization but it is still difficult to find a benchmark to compare technologies that can be used on a commodity smartphone. Hence, we developed our own benchmark: GTR4SL (Ground Truth Recorder for Smartphone Localization). This benchmark allowed us to analyze several technologies: WiFi fingerprinting, WiFi trilateration, PDR and map-matching.

Chapter 4

We proposed a method for characterizing the impact of attitude and position estimations on the rendering of virtual features. This made it possible to identify criteria to better understand the limits of Geo AR for different use cases. From data we collected with our both benchmarks, we applied our method on 4 AR use cases: an application to identify mountains and cities, an application to discover the history of a city, make 3D models appear and turn around in an indoor environment, identify and interact with objects in a building using UWB. Then, we conclude on their feasibility.

Chapter 5

We finally proposed a framework to facilitate the design of AR applications based on geolocation. We show how geodata can be used for AR applications. We proposed a new semantics that extends the data structures of OpenStreetMap. We built our own viewer to display virtual elements over the camera live stream. The particularity of this viewer is that it uses ECEF (Earth-Centered, Earth-Fixed) frame to place virtual objects in the scene. In order to navigate through the scene, the 6 DoF (Degrees of Freedom) of the camera are defined by the device position and attitude. To go from an AR experience to another, it is often useful for the user to use a 2D view. There exists

many 2D renderers for outdoor (Google Maps, Bing, Mapbox, ...) but only few for indoor. We proposed an extension to the open-source project Mapbox to display a multi-floor vector map, using OpenStreetMap specifications. Finally, a specific effort was put on software engineering aspects in order to provide a modular framework. We proposed a simple way to design AR applications based on geolocation. The framework contains modules for geolocation, attitude estimation, POIs management, geofencing, spatialized audio, 2.5D rendering and AR. We show three applications which have been implemented using this framework:

- TyrAr, an application to display information on mountains summits and cities around the user.
- AmiAr, an application to monitor lights, shutters, tv in a smart appartment.
- Venturi Y3 is an AR-Tour of Grenoble with audio description and experiences.

Chapter 1

State of the Art

Contents

1.1 Coordinate Systems 6
1.1.1 Smartphone 7
1.1.2 World Geodetic System (WGS) 7
1.1.3 Earth-Centered, Earth-Fixed (ECEF) 8
1.1.4 Local Tangent Plane (LTP) 9
1.1.5 OpenGL 9
1.2 Attitude Estimation 10
1.2.1 Attitude Representation 10
1.2.2 Allan Variance of the Smartphone IMU 11
1.2.3 Attitude Estimation 14
1.2.4 Evolution of Techniques over the Years 16
1.3 Device Geolocation 18
1.3.1 Geolocation Techniques 18
1.3.2 Map to Improve Location Estimates 26
1.3.3 Fused Approaches 27
1.3.4 Surveys and Evaluations of Geolocation Techniques 28
1.4 Geo AR Browsers 29
1.4.1 Formats and Authoring 29
1.4.2 Frameworks 30
1.4.3 Evaluation of Geo AR Systems 34

In recent years, we observed an increasing availability of geolocated data accessible via the web (OpenStreetMap [OSM], Wikimedia, etc). Thanks to these geo data providers, it is possible to fill AR experiences with millions of Points of Interest (POIs) and features[1]. Sometimes, data does not exist in geo databases, it has to be created by human users via an authoring tool. POIs and virtual features are sent to the Geo AR framework then rendered by the AR viewer. Features

[1] `https://taginfo.openstreetmap.org/reports/database_statistics`

displayed by the AR Viewer are not fixed, they are updated in function of the device position and orientation. The estimated position is provided by a positioning algorithm and the estimated attitude is given by an attitude filter. An overview and links between these concepts are represented in Figure 1.1.

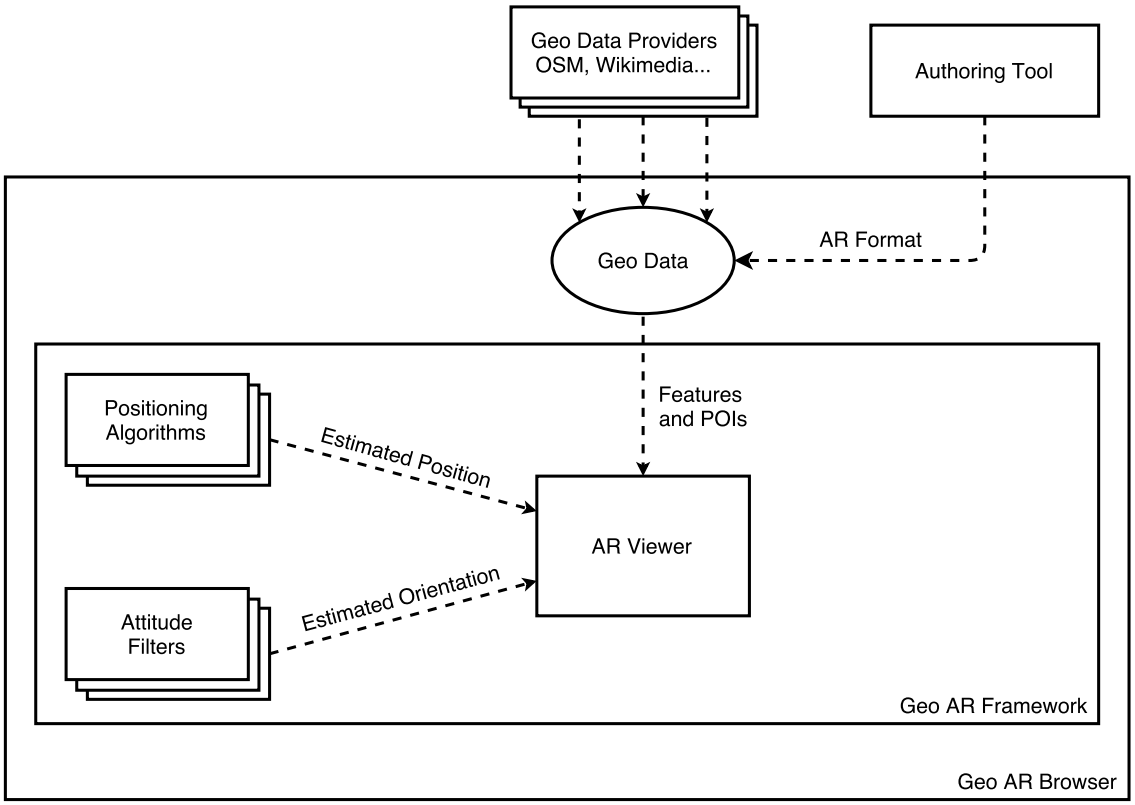


Figure 1.1: Geo Augmented Reality Overview.

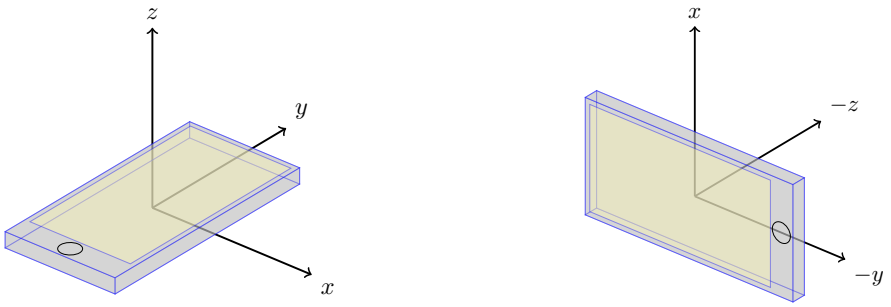
The orientation of the smartphone is often given with respect to ENU (East-North-Up) frame. The position of the smartphone is often provided using geodetic coordinates. The coordinate system of AR viewer is also different.

First, in Section 1.1, we introduce the five coordinate systems that we use in the manuscript. Second, we introduce the specificities of attitude estimation on smartphones in Section 1.2. Then, in Section 1.3, we review positioning techniques available within smartphones. Finally, we present AR frameworks and how are rendered features in Section 1.4.

1.1 Coordinate Systems

In this manuscript we deal with different research domains including attitude estimation, positioning, rendering. Each of them uses a conventional representation for expressing frames and coordinate systems. In this section, we give an overview of all the coordinate systems that are used in the manuscript. Relations between geographic coordinate systems are also introduced.

1.1.1 Smartphone

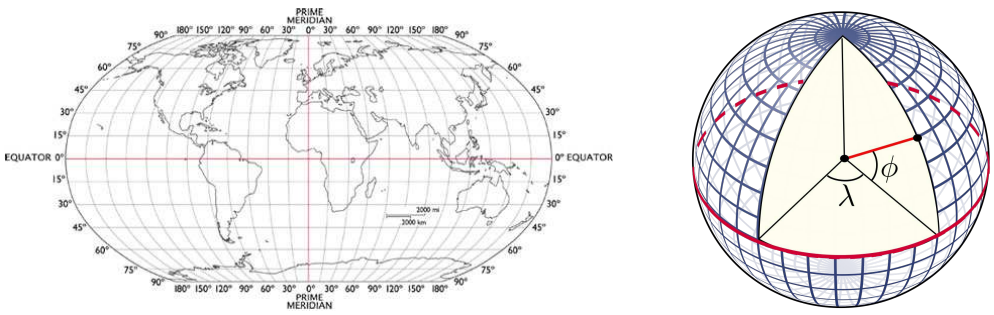


(a) Layed. **(b)** Camera looking forward.

Figure 1.2: Smartphone frame representation.

The major smartphone manufacturers SDK (Android, iOS, Windows) have all agreed on the same representation of the smartphone frame. When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors: accelerometer, gyroscope and magnetometer. The device's natural (default) orientation for smartphones is portrait. However, the natural orientation for many tablet devices is landscape. And the sensor coordinate system is always based on the natural orientation of a device. In Figure 1.2, frame is represented when the smartphone is layed and when smartphone camera is looking forward (AR).

1.1.2 World Geodetic System (WGS)



(a) Meridians an parallels projected on a world map. **(b)** Latitude and longitude on a sphere.

Figure 1.3: World Geodetic System representation.

The World Geodetic System (WGS) is a standard for use in cartography, geodesy, and navigation including GPS. It comprises a standard coordinate system for the Earth, a standard spheroidal reference surface. The latest revision is WGS84 (EPSG:4326), established in 1984 and last revised in 2004. The WGS 84 datum surface is an ellipsoid with major radius $R_{\text{major}} = 6378137\ m$ and the polar semi-minor axis $R_{\text{minor}} = 6356752.3142\ m$. The latitude ($\phi$) of a point on Earth's

surface is the angle between the equatorial plane and the straight line that passes through that point and through the center of the Earth. Lines joining points of the same latitude trace circles on the surface of Earth called parallels. The north pole is 90° N; the south pole is 90° S. The 0° parallel of latitude is designated the equator. The longitude (λ) of a point on Earth's surface is the angle east or west of a reference meridian to another meridian that passes through that point. The prime meridian of WGS84 is close to the Greenwich meridian (102 meters east at the latitude of the Royal Observatory, UK). Figure 1.3 exhibits meridians and parallels on the world map and on a sphere.

A geographic coordinate system is a coordinate system used in geography that enables every location on Earth to be specified by a set of numbers. As WGS is not sufficient to express every location on Earth, the height (h) information is given. Along with the latitude (ϕ) and longitude (λ), the height (h) provides the three-dimensional geodetic coordinates.

1.1.3 Earth-Centered, Earth-Fixed (ECEF)

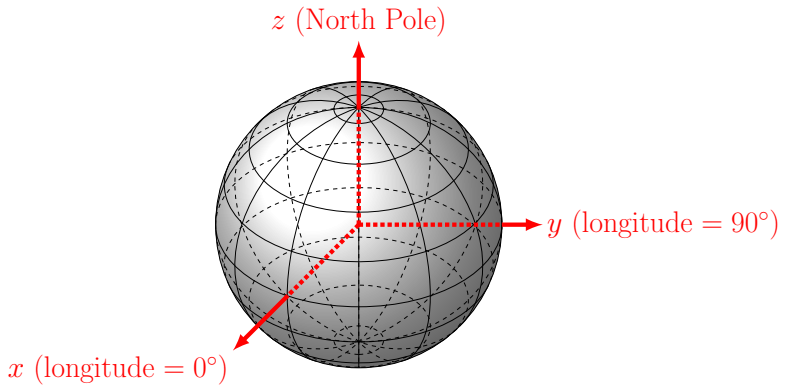


Figure 1.4: Earth-Centered, Earth-Fixed (ECEF) frame representation.

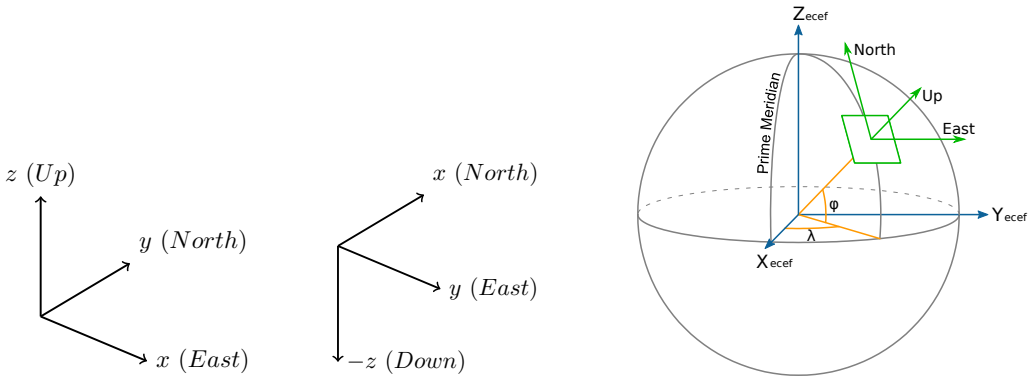
ECEF (Earth-Centered, Earth-Fixed) is a three dimensional geographic and Cartesian coordinate system. The point $(0, 0, 0)$ is defined as the center of mass of the earth. The z-axis extends through true north, where true north is the direction along the earth's surface towards the geographic North Pole. The x-axis intersects the sphere of the earth at 0° latitude (the equator) and 0° longitude (prime meridian). This means that ECEF rotates with the earth, and therefore coordinates of a point fixed on the surface of the earth do not change (see Figure 1.4).

The formula to convert geodetic coordinates (latitude (ϕ), longitude (λ), and height (h)) to ECEF coordinates is given below:

$$\begin{cases} x = (N(\phi) + h) * cos(\phi) * cos(\lambda) \\ y = (N(\phi) + h) * cos(\phi) * sin(\lambda) \\ z = (\frac{R^2_{\text{minor}}}{R^2_{\text{major}}} * N(\phi) + h) * sin(\phi) \end{cases} \tag{1.1}$$

$$\text{where, } N(\phi) = \frac{R^2_{\text{major}}}{\sqrt{R^2_{\text{major}} * cos^2(\phi) + R^2_{\text{minor}} * sin^2(\phi)}}.$$

1.1.4 Local Tangent Plane (LTP)



(a) ENU. **(b)** NED. **(c)** ENU with respect to ECEF.

Figure 1.5: Local Tangent Plane (LTP) frames representation.

Local Tangent Plane (LTP) is a Cartesian geographical coordinate system for representing state vectors that is commonly used by vehicles (cars, planes...). It consists of three numbers: one represents the position along the northern axis, one along the eastern axis, and one represents vertical position. Two major conventions came out from LTP: ENU (East, North, Up) and NED (North, East, Down). The origin of these coordinate systems are usually chosen to be the object's center of gravity. In ENU, the x-axis points to the East, the y-axis points to the North and the z-axis points to the sky perpendicular to the reference ellipsoid. In the same way as ENU, NED coordinates are defined by: x-axis points to the North, the y-axis points to the East and the z-axis points to the center of the Earth. NED coordinates are mostly employed by aerial vehicles (planes, spacecrafts, UAV) to manage positive z values and comply with the right-hand rule. Whereas ENU coordinates are mostly used by ground objects (cars, smartphones). That is why for the rest of the manuscript we will use ENU convention.

Finally, to convert coordinates from ENU frame to ECEF frame, firstly the translation from geodetic coordinates (Eq. 1.1) is applied, following by the rotation R_{ENU}^{ECEF}:

$$R_{ENU}^{ECEF} = R_z(-\frac{\pi}{2} + \lambda)\, R_x(-\frac{\pi}{2} - \phi), \tag{1.2}$$

In Figure 1.5, we provide a representation of ENU, NED and ENU with respect to ECEF frame.

1.1.5 OpenGL

Finally, the last coordinate system we introduce here is the one found in OpenGL. Conventionally, the x-axis points to the right side, the y-axis points to the up and the z-axis points backward. The default orientation of the camera is through z-axis and up vector through y-axis (see Figure 1.6).

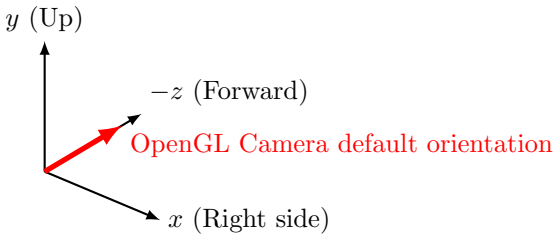


Figure 1.6: OpenGL frame representation.

1.2 Attitude Estimation

Attitude Estimation was first introduced by Wahba in 1965 [Wahba, 1965]. Since then, much research has been carried out on estimation of attitude for a multitude of objects. For example, in 1982, Shuster et al. [Shuster et al., 1982] proposed a Kalman filter for spacecraft attitude estimation. Later, in 2008, Euston et al. [Euston et al., 2008] introduced a complementary filter for attitude estimation of a UAV (Unmanned Aerial Vehicles). Recently, thanks to the miniaturization of sensors, one can estimate attitude of a smartphone, but, how precisely? This is one central topic explored in this dissertation. First, in Section 1.2.1, we provide a background concerning attitude representation. Then, we analyze and characterize the triad of MEMS (Micro-Electro-Mechanical Systems): accelerometer, gyroscope and magnetometer found in smartphone with Allan variance in Section 1.2.2. In Section 1.2.3 we describe how smartphone attitude can be estimated thanks to the triad of sensors we introduced before. Finally, in Section 1.2.4 we present an evolution of the field over the years.

1.2.1 Attitude Representation

The smartphone attitude is determined when the axis orientation of the Smartphone-Frame SF $(\mathrm{SF}_x,\ \mathrm{SF}_y,\ \mathrm{SF}_z)$ is specified with respect to the ENU Frame EF $(\mathrm{EF}_x,\ \mathrm{EF}_y,\ \mathrm{EF}_z)$, see Figure 1.7.

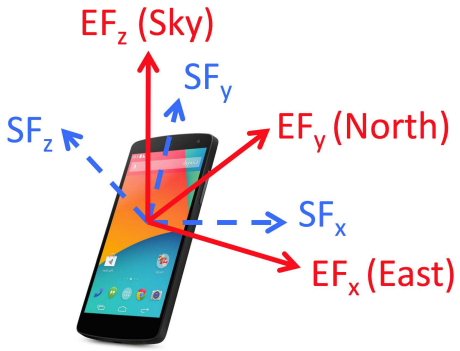


Figure 1.7: The Smartphone-Frame SF (dashed line) and ENU-Frame EF (solid line).

Based on the literature, the attitude can be expressed with four different mathematical representations [Sachs, 2010]. Euler angles (yaw, pitch, roll), rotation matrices, quaternions or axis/angle.

A unit-norm quaternion, which defines the rotation between SF and EF, is defined by:

$$q = {}^{E}_{S}q = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T \in \mathbb{R}^4, \tag{1.3}$$

where q_w and $\begin{bmatrix} q_x & q_y & q_z \end{bmatrix}$ are respectively the scalar and the vector parts of the quaternion.

To express a vector $v = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$ from EF to SF, Hamilton product [Kuipers, 1999] is used (Eq. (1.4)). Conversely, from SF to EF, Eq. (1.5) is used.

$${}^{S}v_q = q^{-1} \otimes {}^{E}v_q \otimes q, \tag{1.4}$$

$${}^{E}v_q = q \otimes {}^{S}v_q \otimes q^{-1}, \tag{1.5}$$

where v_q is the quaternion form of v (Equation (1.6))

$$v_q = \begin{bmatrix} 0 & v_x & v_y & v_z \end{bmatrix}^T. \tag{1.6}$$

The well-known kinematic equation can be used to describe the variation of the attitude in term of quaternion:

$$\dot{q} = \frac{1}{2}\, q \otimes \omega_q, \tag{1.7}$$

where ω_q is the quaternion form of angular velocity. More details about quaternion algebra can be found in [Kuipers, 1999].

The Euler angles representation is composed of three main rotations: a rotation φ around the x-axis (roll angle), a rotation θ around the y-axis (pitch angle) and a rotation ψ around the z-axis (yaw angle). More details about Euler angles properties can be found in [Diebel, 2006].

A rotation matrix for attitude estimation is a 3×3 matrix defining three unit vectors yielding a total of 9 parameters [Shepperd, 1978].

The axis–angle representation parameterizes a rotation by three quantities, a unit vector e indicating the direction of an axis of rotation, and an angle θ describing the magnitude of the rotation about the axis.

Each representation has some drawbacks. In our context, Euler angles cannot be used due to the well-known gimbal-lock problem [Diebel, 2006], when the device is in a pocket or held for phoning, the yaw angle can vary widely. Quaternions avoid the singularity problem, they provide basic primitives with cheap computation cost, but they are less human understandable. All the algorithms that we have implemented in Java/Matlab and benchmarked in Section 2.4 use the quaternion algebra. A simple mathematical transformation between quaternions and Euler angles can be found in [Diebel, 2006].

1.2.2 Allan Variance of the Smartphone IMU

The sensors configuration of a smartphone is composed of a triad of MEMS sensors consisting of a 3-axis gyroscope, a 3-axis accelerometer and a 3-axis magnetometer. The outputs of these low-cost sensors are not perfect and suffer from several problems: noise, bias, scale factor, axis misalignment, axis non-orthogonality and local temperature. In the following, we will analyze the Allan variance on each sensor embedded in a smartphone (a Google Nexus 5) to provide a model of the ouput of each sensor. Allan variance is a measure of frequency stability often used

to characterize MEMS noises [Hou, 2004, Zhang et al., 2008, Hansson and Tufvesson, 2011]. Figure 1.8 exhibits a typical Allan deviation plot for a system. Allan variance analysis allows us to model sensors noise and bias. More information on Allan variance characteristics can be found in [Zhang et al., 2008].

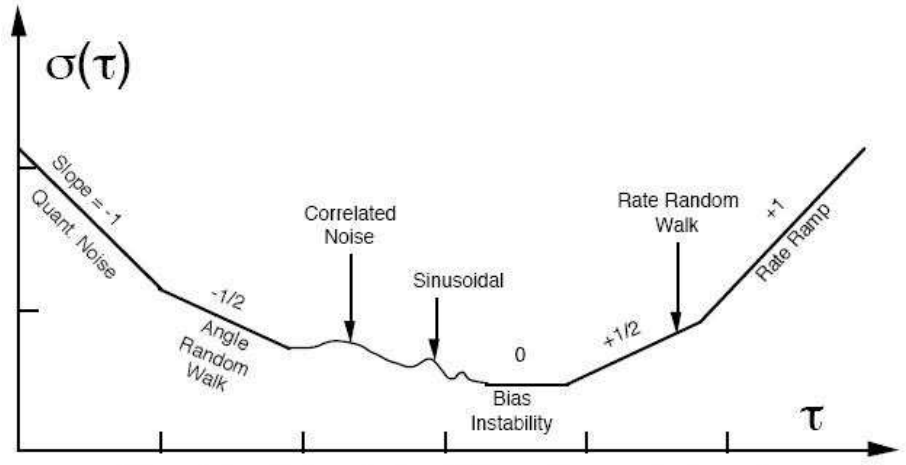


Figure 1.8: Typical Allan deviation plot for a system [Hou, 2004]

Gyroscope

The 3-axis gyroscope measures the angular velocity of the smartphone in $rad.s^{-1}$: ${}^{S}\omega = \begin{bmatrix} {}^{S}\omega_x & {}^{S}\omega_y & {}^{S}\omega_z \end{bmatrix}^T$. An Allan variance study is applied to the gyroscope signal. The results are shown in Fig. 1.9 and three main noises are identified: an Angular Random Walk (ARW) given by the $-\frac{1}{2}$ slope part, a Bias Instability (BI) given by the 0 slope curve part and a Rate Random Walk (RRW) given by the $+\frac{1}{2}$ slope part. The widely used continuous time model for a gyroscope can be written as:

$$ {}^{S}\omega = {}^{S}\omega_r + {}^{S}\omega_b + {}^{S}\omega_n, \tag{1.8} $$

where

${}^{S}\omega$ is the angular rate measured by the gyroscope.

${}^{S}\omega_r$ is the true angular rate.

${}^{S}\omega_b$ is the gyroscope bias, where its derivative ${}^{S}\dot{\omega}_b$ is modeled by a random walk noise ${}^{S}\dot{\omega}_b = {}^{S}\omega_{b_n}$, its standard deviation (BI) is noted ${}^{S}\sigma_{\omega_{b_n}}$. The gyroscope bias leads after integration (see Eq. (1.7)) to an angular drift, increasing linearly over time.

${}^{S}\omega_n$ is the gyroscope white noise, its standard deviation (ARW) is noted ${}^{S}\sigma_{\omega_n}$.

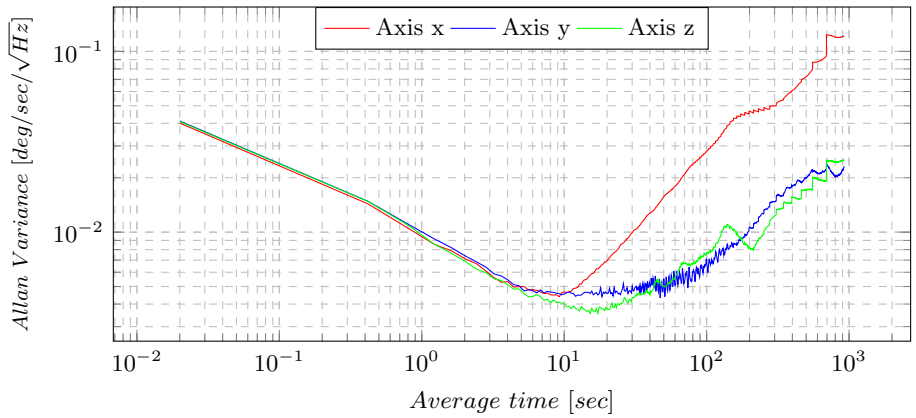


Figure 1.9: Allan variance of gyroscope signal.

Accelerometer

The 3-axis accelerometer measures the sum of the gravity and external acceleration of the smartphone in $m.s^{-2}$: ${}^{S}a = \begin{bmatrix} {}^{S}a_x & {}^{S}a_y & {}^{S}a_z \end{bmatrix}^T$. In the same way as the gyroscope, we used the Allan variance on the accelerometer signal. The results are shown in Fig. 1.10 and three main noises are identified: a Velocity Random Walk (VRW) given by the $-\frac{1}{2}$ slope part, a Bias Instability (BI) given by the 0 slope curve part and a Correlated Noise (CN) given by the oscillations (mostly on x-axis and z-axis). The continuous time model for accelerometer can be written as:

$$ {}^{S}a = {}^{S}a_r + {}^{S}a_b + {}^{S}a_n, \tag{1.9} $$

where

- ${}^{S}a$ is the sum of the gravity and external acceleration of the body measured by the accelerometer.
- ${}^{S}a_r$ is the true sum of the gravity and external acceleration of the body.
- ${}^{S}a_b$ is the accelerometer bias, where its derivative ${}^{S}\dot{a}_b$ is modeled by a Gauss-Markov noise: ${}^{S}\dot{a}_b = \beta {}^{S}a_b + {}^{S}a_{b_n}$, the standard deviation of ${}^{S}a_{b_n}$ (BI) is noted ${}^{S}\sigma_{a_{b_n}}$.
- ${}^{S}a_n$ is the accelerometer white noise, its standard deviation (VRW) is noted ${}^{S}\sigma_{a_n}$.

An uncalibrated accelerometer in a static phase provides a magnitude of acceleration close to g. In [Frosio et al., 2013], the authors provide an accelerometer calibration algorithm based on a minimum of 7 static phases. This calibration allows to remove the bias and misalignment by normalizing the acceleration vector in multiple smartphone orientations. Finally the acceleration magnitude is close to g.

Magnetometer

The 3-axis magnetometer measures the magnetic field of the smartphone in micro-tesla (μT): ${}^{S}m = \begin{bmatrix} {}^{S}m_x & {}^{S}m_y & {}^{S}m_z \end{bmatrix}^T$. The Allan variance is used on magnetometer signal and the results

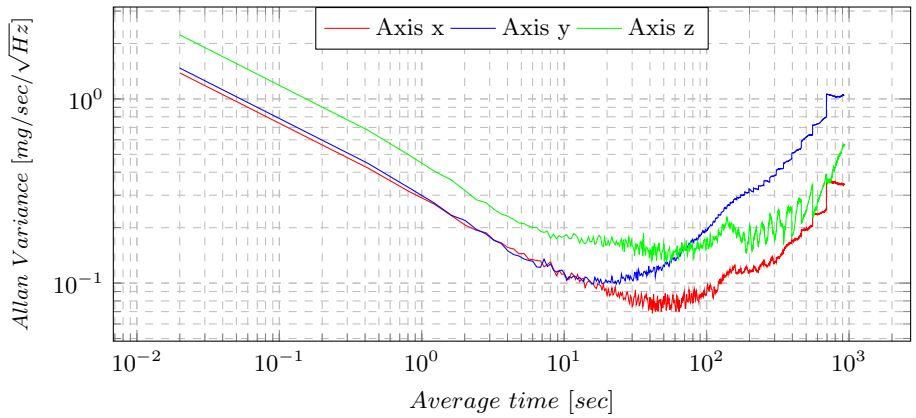


Figure 1.10: Allan variance of accelerometer signal.

are shown in Fig. 1.11 where three main noises are identified: an Angle Random Walk (ARW) given by the $-\frac{1}{2}$ slope part, a Bias Instability (BI) given by the 0 slope curve part and a Correlated Noise (CN) given by the oscillations. The continuous time model for magnetometer can be written as:

$$^{S}m = {}^{S}m_r + {}^{S}m_b + {}^{S}m_n, \tag{1.10}$$

where

^{S}m is the magnetic field measured by the magnetometer.

$^{S}m_r$ is the true magnetic field.

$^{S}m_b$ is the magnetometer bias, where its derivative $^{S}\dot{m}_b$ is modeled by a Gauss-Markov noise: $^{S}\dot{m}_b = \beta {}^{S}m_b + {}^{S}m_{b_n}$, the standard deviation of $^{S}m_{b_n}$ (BI) is noted $^{S}\sigma_{m_{b_n}}$.

$^{S}m_n$ is the magnetometer white noise, its standard deviation (ARW) is noted $^{S}\sigma_{m_n}$.

The Allan variance analysis and results are similar to those proposed in other works where MEMS are used [Renaudin and Combettes, 2014, Hou, 2004].

1.2.3 Attitude Estimation

The problem of finding the optimal attitude estimation solution was formulated by Wahba in 1965 [Wahba, 1965]. Wahba's problem seeks to find a rotation matrix between two coordinate systems from a set of vector observations (minimum two vectors known in a fixed frame and in a body frame).

In our case, the two coordinate systems are the Smartphone Frame (SF) and the Earth Frame (EF) as shown in Figure 1.7. A typical Inertial Measurement Unit (IMU) in a smartphone can provide two vector observations expressed in two frames:

- acceleration in SF provided by an accelerometer noted ^{S}acc and its projection in EF noted ^{E}acc.

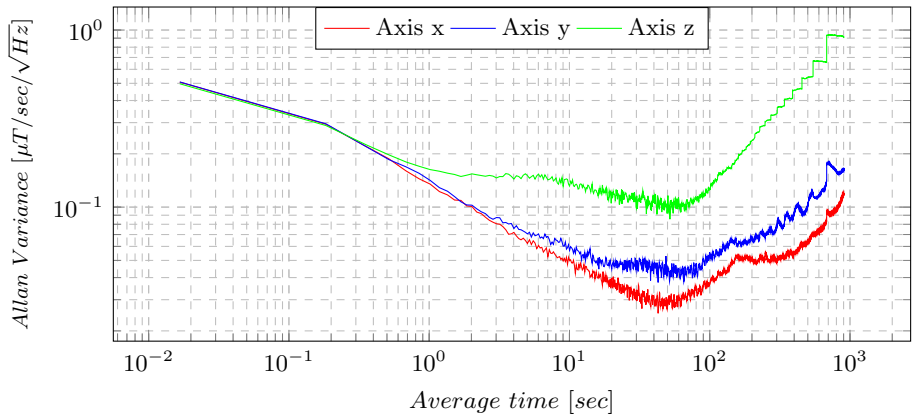


Figure 1.11: Allan variance of magnetometer signal.

- magnetic field in SF provided by a magnetometer noted ${}^{S}\text{mag}$ and its projection in EF noted ${}^{E}\text{mag}$.

These 2 observation vectors can be modeled as following:

$$ {}^{S}\text{acc}_q = q^{-1} \otimes {}^{E}\text{acc}_q \otimes q, \tag{1.11} $$

$$ {}^{S}\text{mag}_q = q^{-1} \otimes {}^{E}\text{mag}_q \otimes q. \tag{1.12} $$

If the smartphone is in static phase (not translating), $\text{acc}^{ext} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ and

$$ {}^{E}\text{acc} = \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T. \tag{1.13} $$

In absence of magnetic deviations, $\text{mag}^{ext} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ and

$$ {}^{E}\text{mag} = \begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T, \tag{1.14} $$

where m_x, m_y and m_z can be obtained using the WMM [(NGA) and the U.K.'s Defence Geographic Centre (DGC), 2015].

Figure 1.12 shows these two vectors: ${}^{E}\text{acc}$ in blue and ${}^{E}\text{mag}$ in green.

In addition to accelerometer and magnetometer, the gyroscope is used to estimate variation of attitude. Unfortunately, the gyroscope bias leads after integration (Equation (1.7)) to an angular drift, increasing linearly over time. Since the use of only gyroscope is not enough for attitude estimation, accelerometer and magnetometer are used to get an absolute quaternion and compensate the drift. The crux in solving an attitude estimation problem then consists in combining inertial and magnetic sensor measurements in a relevant manner. Figure 1.13 illustrates the whole approach, where K is the fusion gain between data merged from accelerometer-magnetometer fusion and gyroscope integration. This gain is adjusted depending on sensors reliability.

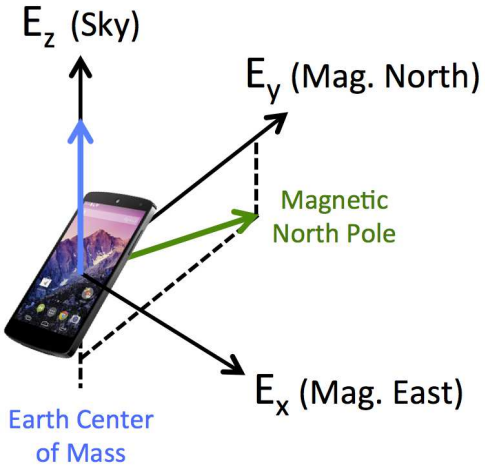


Figure 1.12: Reference vectors when the smartphone is static and in the absence of magnetic deviations.

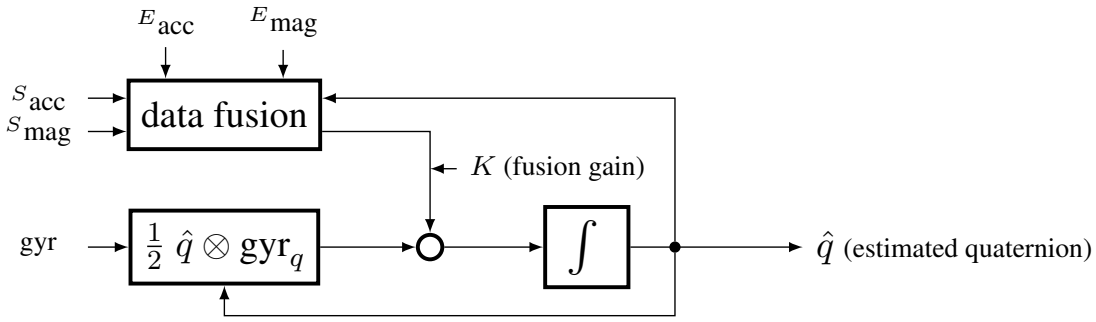


Figure 1.13: General schema for attitude estimation.

1.2.4 Evolution of Techniques over the Years

Since 1965, a multitude of solutions have been proposed to resolve attitude estimation problem, such as TRIAD [Black, 1964], QUaternion ESTimator (QUEST) [Shuster and Oh, 1981], Singular Value decomposition method (SVD) [Markley, 1988], Kalman Filters (KF) [Harada et al., 2004, Rehbinder and Hu, 2004, Choukroun et al., 2006, Lee et al., 2012b, Valenti et al., 2016], Extended Kalman Filters (EKF) [Marins et al., 2001, Sabatini, 2006, Zhu et al., 2007, Munguia and Grau, 2011, Renaudin and Combettes, 2014], Unscented Kalman Filters (UKF) [Crassidis and Markley, 2003], Adaptive Kalman Filters (AKF) [Suh, 2010, Makni et al., 2014], Particle Filters [Oshman and Carmi, 2006] and more recently Observers [Fourati et al., 2011, Mahony et al., 2008, Martin and Salaün, 2010, Madgwick et al., 2011]. A survey and an analysis of these methods can be found in [Markley and Mortari, 2000]. In 2007, Crassidis et al. [Crassidis et al., 2007] provide another survey with a focus on nonlinear attitude estimation methods. In this dissertation, we further focus on algorithms that use measurements from the 3 sensors that are now commonly found on smartphones: gyroscopes, accelerometers and magnetometers, and attempt to leverage on these measurements to provide precise attitude estimation of smartphones carried by pedestrians.

Most algorithms developed so far rely on a common assumption: the external acceleration is negligible. However, when used in the context of smartphone carried by a pedestrian, this assumption is questionable (we have experimentally observed high external accelerations: see e.g. second column of Table 2.1). Specifically, the relation between ${}^{S}\text{acc}$ and ${}^{E}\text{acc}$ given by Equation (1.11) holds only if no external acceleration is applied on the smartphone. Assumption of external acceleration is not a new problem, in [Harada et al., 2004, Rehbinder and Hu, 2004, Sabatini, 2006, Lee et al., 2012b] authors propose to discard accelerometer measurements in the update phase of their KFs. They set values of covariance matrix to infinity when:

$$\underbrace{\left|\|{}^{S}\text{acc}\| - \|{}^{E}\text{acc}\|\right|}_{\mu} > \gamma_{\text{acc}}. \tag{1.15}$$

In [Munguia and Grau, 2011] and [Li and Wang, 2013], the authors explain how they adjust the covariance matrix in function of the left term of Eq. (1.15). In [Suh, 2010] and [Makni et al., 2014], authors use KF residual errors to detect external acceleration. The technique proposed in [Suh, 2010] needs time to let residual matrix converge in a static phase to identify bias before estimating external accelerations. Finally, in [Renaudin and Combettes, 2014], authors only perform the update phase of their KF during periods considered as Quasi Static Field (QSF). During QSF, a low variance is given to measurements and ${}^{E}acc$ is adjusted during these phases. To the best of our knowledge, the use of a detector à la (1.15) has not been investigated yet with an observer-based filter.

Most algorithms found in the literature do not consider magnetic perturbations. However, in the pedestrian context, the smartphone is often exposed to ferromagnetic objects, and this is known to yield a bad attitude estimation [Afzal et al., 2011, Bachmann et al., 2004]. Few papers are concerned with magnetic perturbations for attitude estimation on a smartphone carried by pedestrians. In [Madgwick et al., 2011], authors consider the impact of magnetic perturbations on the North-East plane, abstracting over other possible impacts. In [Harada et al., 2004] and [Sabatini, 2006], authors set the covariance matrix of magnetic measurements to infinity when:

$$\left|\|{}^{S}\text{mag}\| - \|{}^{E}\text{mag}\|\right| > \gamma_{\text{mag}}. \tag{1.16}$$

In [Harada et al., 2004], in addition to the detector (1.16), Harada et al. use the following property to detect magnetic perturbations:

$$\theta({}^{S}\text{acc}, {}^{S}\text{mag}) - \theta({}^{E}\text{acc}, {}^{E}\text{mag}) > \gamma_{\theta}, \tag{1.17}$$
$$\text{where: } \theta(v_1, v_2) = \arccos\frac{v_1 \cdot v_2}{\|v_1\| \cdot \|v_2\|}.$$

Similarly to how external accelerations are treated, in [Renaudin and Combettes, 2014], the authors use a QSF detector based on the variance of measurements.

In Chapter 2 of this manuscript, we will introduce a new technique (Section 2.3) for limiting the impact of magnetic perturbations on attitude estimation algorithms that are executed on smartphones carried by pedestrians. In addition, we conduct extensive practical experiments with several (and typical) motions of smartphones carried by pedestrians, and show how our approach compares and improves over previous works in this context. To the best of our knowledge, our systematic comparison of attitude estimation algorithms is the first in this context. Our experiments include 126 datasets with several typical motions, several devices, realistic magnetic perturbations, and a fine-grained analysis.

In the next section, in the same manner than for attitude estimation filter, we will present algorithms which can be used for Geo AR.

1.3 Device Geolocation

Geolocation is the identification or estimation of the real-world geographic location of an object, it involves the generation of a set of geographic coordinates. The first aspects of geolocation appeared a long time ago (2 000+ years) when ancient Greeks triangulated their geographical location using only the stars. In the 1920's, in many countries, the loop antennas (using direction finding technique) were used to determine the position of airships, aircrafts and ships locally. For many years, geolocation has been a world-wide challenge and particularly during the wars. In 1933, the U.S. Naval Research Laboratory proposed the use of a pulse radar technique to be able to detect aircraft and ships. Later, in 1978, United States government announced the deployment of 11 Navstar GPS (Global Positioning System) satellites for military use. In 1983, a public access to GPS was released by the U.S. for commercial aircraft to improve air navigation safety. Finally, in 1989, Magellan Navigation, Inc introduced NAV 1000, the first hand-held GPS device. Over the years, sensors like GPS, accelerometer, magnetometer, gyroscope, WiFi... have been miniaturized. Therefore, in 2000's and beginning of 2010's, we observed a multitude of new sensors embedded in recent smartphones. These sensors allow researchers to develop geolocation techniques already used in other fields like robotic or army.

In order to provide accurate Geo AR experiences, we browsed the literature to find the best geolocation techniques with smartphones. The first problem encountered is that each technique will not work in the same manner for different context. For instance, GNSS (Global Navigation Satellite System) exhibits a good precision when used in a clear outdoor area thanks to the well coverage of satellites. However, if GNSS is used inside buildings, satellite wave propagation will not rely on a line-of-sight, therefore precision is degraded. Conversely, localization based on WiFi works relatively well in apartment buildings thanks to the multitude of access points. Whereas, in a field, with this approach and without any access point, there is no possibilities to estimate user location.

In 2012, Mautz et al. [Mautz, 2012] provided a survey of indoor positioning technologies. A figure taken from [Mautz, 2012] represents technologies accuracy in function of their coverage is given (Figure 1.14). However, not all the technologies presented in this work are available in smartphones. Moreover, precision errors of each technology span different orders of magnitude (e.g: centimeter-level, meter-level...), this information is not always enough accurate for our work.

Geolocation techniques which can be used with a smartphone are introduced below.

1.3.1 Geolocation Techniques

In this section, we present geolocation techniques from the literature which can be implemented on a smartphone. We also included accuracy claimed by authors for each technique.

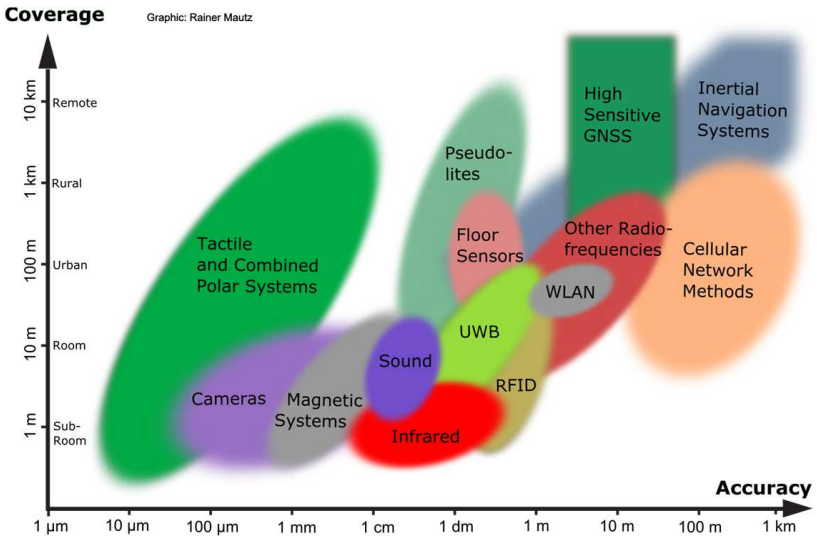


Figure 1.14: Overview of indoor technologies in dependence on accuracy and coverage. [Mautz, 2012]

GNSS

In 2017, GNSS sensors are present in almost all new manufactured smartphones and are more and more reliable. Usually, smartphones support U.S. GPS (Global Positioning System) as well as GLONASS (Russian Global Navigation Satellite System). On their technology, U.S. government claims an horizontal accuracy of less than 1.9 meters 95% of the time [National Coordination Office for Space-Based Positioning, Navigation, and Timing, 2017]. In order to compete with others GNSS providers, European Union launched the Galileo project in 2005. Galileo is intended to provide horizontal and vertical position measurements within 1-meter precision, and better positioning services at higher latitudes than other positioning systems. However, they expect to reach the "Full Operational Capability" in 2019.

GNSS seems to have unanimous support for outdoor positioning when it is used in a clear space. Nevertheless, GNSS relies on time of flight values, thus, when the line of sight between receiver and a satellite is obstructed by an object, for instance a building, accuracy is degraded. In [Ben-Moshe et al., 2011], authors show that average precision error in a urban canyon is about $15\ m$. Finally, inside a building, GNSS becomes unreliable with errors $> 60\ m$ [Peterson et al., 1997].

WiFi Fingerprinting

In the recent years, research based on the early RADAR system [Bahl and Padmanabhan, 2000], using the Wi-Fi fingerprinting technology has spread across multiple directions to solve the current challenge of obtaining a precise position estimation indoors [He and Chan, 2016]. For WiFi fingerprinting it is assumed that each location in a building can be identified by its unique Fingerprint. A WiFi Fingerprint (FP) is the (in theory) unique combination of the Received Signal Strength Indicator (RSSI) and its according Access point (AP, represented by its unique MAC address). This system does not require line of sight (LoS), the knowledge about the exact APs

locations and therefore neither angle nor distance measurements [Beder and Klepal, 2012]. WiFi Fingerprinting is usually divided into two phases: An offline (site survey) phase and an online (navigation) phase. In the offline phase predefined Reference Points (RP) with a known location are visited. For each RP at least one FP is created by storing the received RSSI Values from the surrounding APs in a database. Therefore each RP is represented by a FP. During the online phase, the user (client) performs a WiFi scan and then stores the received RSSI values together with the corresponding AP in a vector. The vector is then used together with the pre-created DB by the geolocation algorithm to calculate a position estimation. Each FP is compared with the scan of the client resulting in score which represents the similarity of the two. Either the minimum or maximum score can be used to identify the corresponding RP and therefore the location. Figure 1.15 summarizes the system flow.

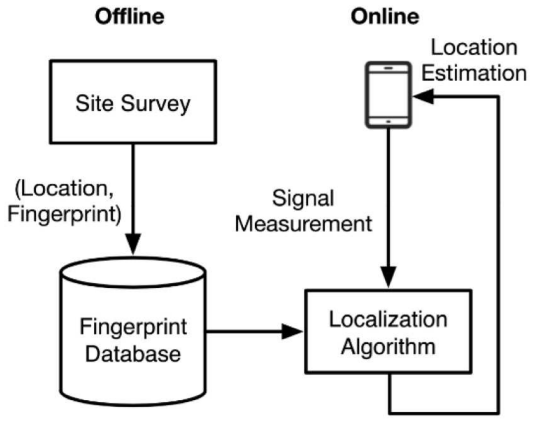


Figure 1.15: WiFi fingerprinting system flow. [He and Chan, 2016]

Generally, in the literature, authors evaluate WiFi fingerprinting approaches with a precision of around $1 - 3\ m$ [Lymberopoulos et al., 2015]. WiFi fingerprinting has also its limits:

- It relies on non-moving access points. If WiFi antennas are moved, position estimation is altered.
- Offline phase is tedious, as FPs should be recorded at known location.
- Due to the nature of the WiFi signal being an electromagnetic wave, damping and reflection effects have a strong influence on the radio distribution and therefore the accuracy [Karimi, 2013].

WiFi Trilateration

A second technique to determine user position with WiFi signals consists in using trilateration with RSSI from WiFi AP. Trilateration is a widely used technique to determine position with radio signals: GPS, Bluetooth, UWB, etc. Trilateration is the process of determining locations of points by measurement of distances, using the geometry circles (Figure 1.16). This technique is often mistaken with triangulation, which consists in calculating a position thanks to one distance and 2 angles.

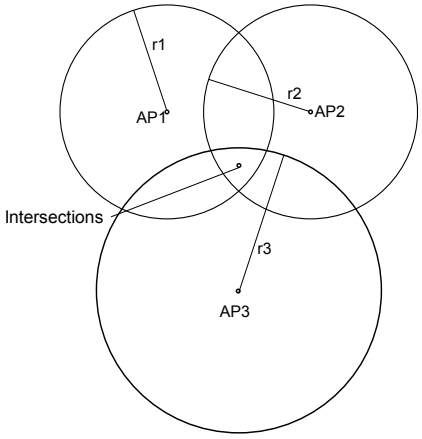


Figure 1.16: An example of WiFi trilateration with 3 Access Points (AP)

Distances between the smartphone and WiFi access points are not known by the system but thanks to the attenuation of WiFi signal, distance can be estimated. Free-space path loss model (FSPL) (Eq. 1.18) is a well known radio propagation model that predicts the path loss of a signal through free space (usually air) [Rappaport et al., 1996].

$$PL(dB) = 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}\left(\frac{4\pi}{c}\right), \tag{1.18}$$

where,

f is the signal frequency (in hertz),

d is the distance from the transmitter (in meters)

c is the speed of light in a vacuum, $2.99792458 * 10^8\ m.s^{-1}$.

This model has been adapted for indoor localization by taking into account walls textures and multiple floors.

One model is the log-distance path loss model [Rappaport et al., 1996]:

$$PL(dB) = PL_0 + 10\gamma \log_{10}\frac{d}{d_0} + X_g, \tag{1.19}$$

where,

PL_0 is the path loss at the reference distance d_0. Unit: Decibel (dB)

d_0 is the reference distance, usually 1 meter.

γ is the path loss exponent.

X_g is a normal (or Gaussian) random variable with zero mean, reflecting the attenuation (in decibel). This random variable may have Gaussian distribution with σ standard deviation in dB.

γ and σ are dependent on frequency of transmission and building type.

An other model is the ITU indoor propagation model [Series, 2012]:

$$PL(dB) = 20 \log f + N \log d + P_f(n) - 28 \tag{1.20}$$

where,

N is the distance power loss coefficient.

n is the number of floors between the transmitter and receiver.

$P_f(n)$ is the floor loss penetration factor.

Once again, N and $P_f(n)$ depend on frequency band and building type.

Only few studies tried to evaluate the precision of WiFi trilateration. On a device which is not a smartphone, in [Aboodi and Wan, 2012], authors exhibit an accuracy of 2.6 m using the FSPL model.

SHS

The Step and Heading System (SHS) is a Pedestrian Dead Reckoning (PDR) technique which consists in detecting user's step, estimating step size and user direction. In a simple implementation, the user holds the phone in front of him and each step causes position to move forward a "step length" distance in the direction measured by the compass (see Figure 1.17). The main problem of a PDR technique remains the knowledge of the starting position.

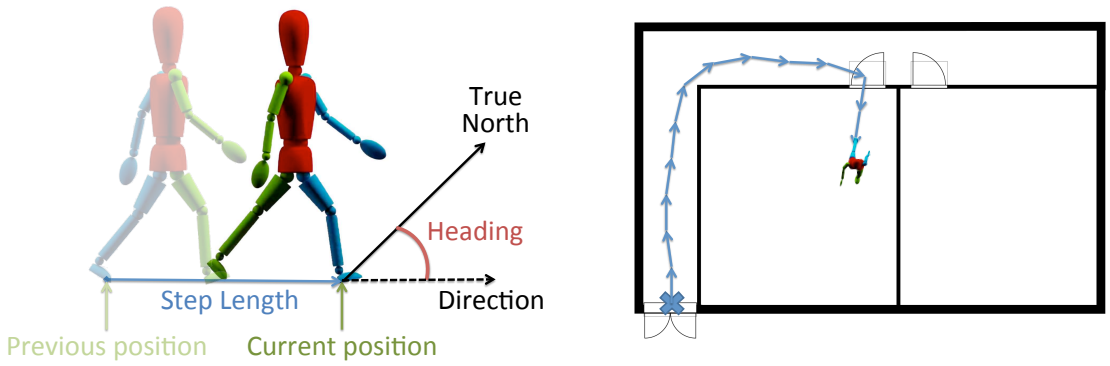


Figure 1.17: Step and Heading System (SHS) overview

Step detection researches started a long time ago (1965) with commercial pedometers that were originally used by sports and physical fitness enthusiasts. Pedometers were often worn on the belt and thanks to the analysis of the gait cycle, step detection becomes possible [Harle, 2013]. Nevertheless, challenges changed when unconstrained smartphone arrived on the market. The hand motion can be decoupled from the general motion of the user and the characteristics of the inertial signals can differ depending on the carrying modes. Therefore, in [Susi et al., 2013], authors developed algorithms for characterizing the gait cycle of a pedestrian using a hand-held device. In [Brajdic and Harle, 2013], authors applied walk detection and step counting algorithms

to smartphone sensor data. The results favour the use of standard deviation thresholding and windowed peak detection with error rates of less than 3%.

Step length estimation has been widely investigated when the device is fixed on the body, however with an unconstrained smartphone the challenge remains complex. The algorithms can be categorized in two groups, namely algorithms based on biomechanical models and algorithms based on empirical relationships. In [Jahn et al., 2010], authors provide a theoretic evaluation of the estimators' performance and present a comparison based on measurement data. They obtained median values of the errors varying between 1.3 % and 3.6 % with a method based on the correlation of the vertical acceleration during one step with the length of the step. Renaudin et al. in [Renaudin et al., 2012] proposed another method by analyzing sensor's signal in the frequency domain. Before computing the distance walked over a step, the carrying mode of the device, i.e., texting, phoning or arm swing, and the user motion are identified. Their experimental protocol demonstrated error over traveled distance between 2.5% and 5%.

Finally, the last step consists in finding the user walking direction in the horizontal plane, defined by geographical north and geographical east (e.g: north $= 0°$, east $= 90°\ldots$). In order to solve this problem, one approach consists in estimating smartphone orientation (see Section 1.2.3), then finding the misalignment between hand-held device and walking direction [Combettes and Renaudin, 2015]. As we saw in Section 1.2.3, attitude estimation is challenging when the device is not static and exposed to magnetic perturbations. A bad attitude estimation has a direct impact on the final position estimated by the SHS. Renaudin et al. proposed an attitude filter to reduce impact of magnetic perturbation and external accelerations in [Renaudin and Combettes, 2014]. A further study on attitude filters is given in Section 1.2.4. Moreover, the misalignment problem is often misunderstood in the literature because when the smartphone is held in texting mode, rotation between smartphone frame and frame defined by user walking direction is really small. Consequently, if misalignment is neglected, SHS will work well when smartphone is in texting mode, however, once the smartphone orientation changes (phoning, pocket...), SHS will no longer work well. A comparison and evaluation of misalignment technique is given by Combettes et al. in [Combettes and Renaudin, 2015].

We did not find any evaluation of the full process of SHS with hand-held devices or smartphones. Most of the time SHS is fused with another technology.

Inertial Navigation System (INS)

An Inertial Navigation System (INS) is a system that tracks position by estimating the full 3D trajectory of the sensor at any given moment. Tracking position then involves subtracting the gravitational signal from the vertical accelerometer signal and performing double integration on the remaining 3D acceleration.

Inevitably, measurement errors are present within the sensor data, and the double integration results in a drift (potentially cubic growth in time). The average error in position obtained by such a system after 60 seconds is over $150\ m$ [Woodman, 2007]. This drift can be lowered to $5\ m.\text{min}^{-}1$ with the help of magnetometers under certain conditions. This method has been extensively studied [Al Nuaimi and Kamel, 2011, Yun et al., 2012, Fourati, 2015] when the device is placed on a shoe, because, from the human walking model, two gait phases are extracted: stance and swing. Therefore, during the stance phase, the foot is static and the device velocity is considered to be 0 (See Figure 1.18). This information allow to considerably reduce the drift from

INS system.

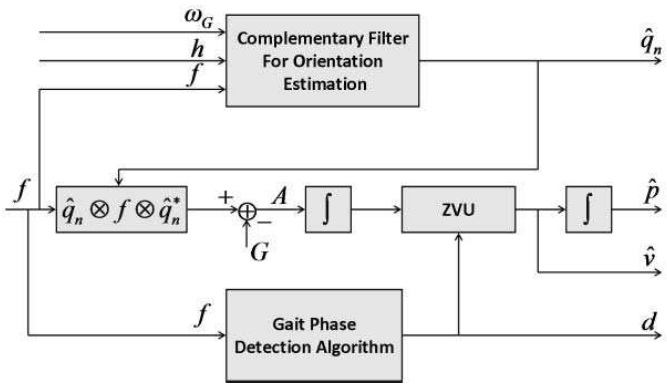


Figure 1.18: Block diagram of the foot motion tracking algorithm that produces the foot orientation quaternion, foot position, foot velocity, and gait phase in [Fourati, 2015]

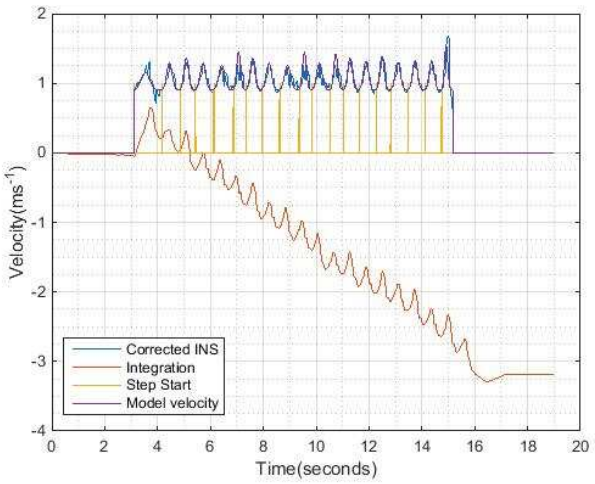


Figure 1.19: Estimated velocity for standard INS vs proposed method in [Lakmal and Samarabandu, 2016]

Unfortunately, zero velocity update cannot be directly applied to a hand-held smartphone because device is never static when walking (see Figure 1.19). In [Lakmal and Samarabandu, 2016], authors proposed a method based on a Gaussian velocity model (see Figure 1.19). They claim that the proposed method is more robust than SHS with a gain of about 20%.

Bluetooth Low Energy (BLE)

The introduction of low-cost, low-power Bluetooth Low Energy (BLE) beacons for proximity detection also provides a new signal that can be used to fine-grained positions estimation. In the same manner than WiFi, it is possible to use BLE with trilateration or with fingerprinting. In

[Faragher and Harle, 2014], authors compared WiFi fingerprinting with BLE fingerprinting. They conclude on several aspects:

- The low bandwidth of BLE makes more RSSI fluctuations than WiFi.
- Smoothing the BLE RSSI measurements by batch filtering multiple beacon measurements per fingerprint is necessary.
- Positioning accuracy increases with the number of beacons per fingerprint, up to a threshold of around 6–8.
- Active WiFi scanning and WiFi network access can cause errors in BLE signal strength measurements.

They obtained an accuracy of less than $2.6\ m$ 95% of the time, whereas WiFi exhibits an average positioning error of less than $8.5\ m$.

Magnetic

While a system like SHS suffers from magnetic perturbations, magnetic field positioning methodology can take advantage of these anomalies [Li et al., 2012]. The goal is to identify a position by a unique signature, where signature comes from a magnetic fingerprint. Unfortunately, magnetic field intensity data only consists of three components. Since the magnetic north is generally unknown, even with the help of the accelerometer to detect the direction of the gravity, only two components can be extracted, i.e. the horizontal intensity and the vertical intensity. Authors claimed to obtain a centimetric error (between $4\ cm$ and $13\ cm$) when building's level is known. Moreover, in a multiple buildings setup, when the correct building was unknown, using the geomagnetic field fingerprint alone could place the test point in the wrong building.

Visible Light Communication (VLC)

Visible Light Communication (VLC) is a data communication variant which uses information from fluorescent lamps. The luminaires (modified to allow rapid, on-off keying) transmit their identifiers and/or locations encoded in human-imperceptible optical pulses. A camera-equipped smartphone, using just a single image frame capture, can detect the presence of the luminaires in the image, decode their transmitted identifiers and/or locations, and determine the smartphone's location and orientation relative to the luminaires. In [Kuo et al., 2014], authors demonstrate a decimeter-level accuracy in both carefully controlled and more realistic human mobility scenarios.

Ultra-Wideband (UWB)

One new wireless technology that is reaching market is Ultra-Wideband (UWB) radio. UWB is used to communicate between devices, similar to Bluetooth and WiFi, but with a higher data rate. UWB has also been designed specifically to transmit in a way that enables much more precise distance measurements and location positioning. Consumer smartphones are not equipped with a UWB receiver yet. However a French start-up (BeSpoon) [2] has demonstrated that UWB technology can be successfully integrated into a smartphone. The SpoonPhone is currently only

[2] `http://spoonphone.com`

a prototype, but is being sold to hardware manufacturers and software developers for R&D and evaluation purposes. BeSpoon announced a location accuracy of a few inches.

1.3.2 Map to Improve Location Estimates

Longer-term tracking of pedestrians has been achieved by incorporating external measurements and environment information. This is usually done with vectorial maps. For years, navigation systems in cars have used map information to correct location provided by GNSS signals [Kim and Kim, 2001]. This approach has been adapted to pedestrian navigation. For instance, the knowledge of footpaths network can enhance SHS or GNSS accuracy outdoor. Another example is the use of building maps for indoor navigation to avoid an estimated position to cross a wall. We split map-matchings techniques into two categories: point to network and particle filters.

Point to Network

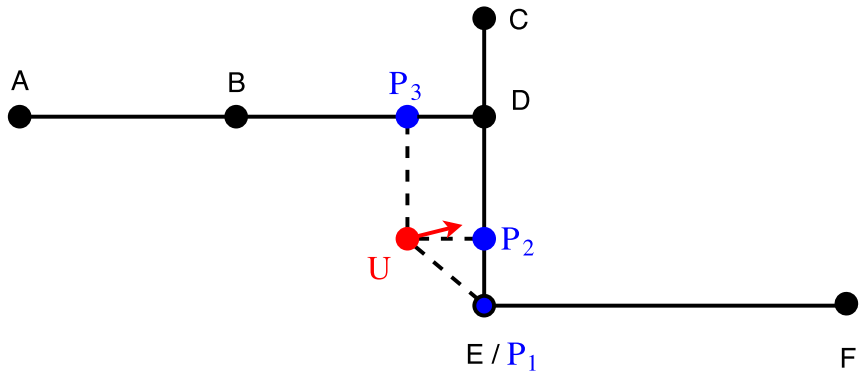


Figure 1.20: Example of point to network approach for map matching

One natural way to proceed is to match the user estimated position U to the "closest" node in the network (point-to-point) [Bernstein and Kornhauser, 1998]. Here, we consider the "closest point" using Euclidean metric. To see this, consider the example shown in Figure 1.20. Our example is composed of 6 nodes (A, B, C, D, E and F) and 5 paths (AB, BD, CD, DE, EF). Using the point to point technique, user estimated position will be projected on node E (P_1).

The next most natural way to proceed is to attempt to identify the path which is closest to U (point-to-curve). Lets consider $A(A_x, A_y)$ and $B(B_x, B_y) \in \mathbf{R}^*$ the both extremities of a path. The distance (d) between $C(C_x, C_y)$ and $[AB]$ is defined by:

$$d = \sqrt{\frac{[(A_y - B_y)C_x + (B_x - A_x)C_y + (A_xB_y - B_xA_y)]^2}{(A_y - B_y)^2 + (B_x - A_x)^2}} \tag{1.21}$$

In our example, user estimated position will be projected on path [DE] (P_2).

Moreover, a condition is added to match closest segments only if user's heading (θ_{heading}) is close enough (θ) to the direction of the segment (θ_{segment}), see Eq. 1.22.

$$\theta = (\theta_{\text{heading}} - \theta_{\text{segment}} + 180) \bmod 360 - 180 \tag{1.22}$$

If we consider $\theta = 15°$ in our example, location will be projected on [BD] footpath (P_3).

Finally, to avoid a projection on far away footpath, if the distance between estimated position and closest segment is greater than a threshold (γ), projection is not applied. Usually γ is lowered for indoor positioning (compared to outdoor) due to the density of networks.

For instance, the Footpath application [Link et al., 2011] uses network from OpenStreetMap project and adapts point-to-curve map-matching on a moving window of five steps.

Particle Filter Localization

Particle filter localization, also known as Monte Carlo localization is an algorithm (initially for robots) to localize using a particle filter [Thrun et al., 2001]. Given a map of the environment, the algorithm estimates the position and orientation of the smartphone as it moves within the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, i.e., a hypothesis of where the user is. The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning the smartphone has no information about where it is and assumes it is equally likely to be at any point in space. Whenever the moves (updates from a positioning system) from smartphone are detected, algorithm shifts the particles to predict its new state after the movement. When the smartphone detects that particles are crossing a wall (thanks to the map information), particles are resampled based on recursive Bayesian estimation.

This approach is often coupled with more than one localization technique. A different weight is applied on particles in function of the positioning technique accuracy [Masiero et al., 2014].

1.3.3 Fused Approaches

Recently, in the literature, we observed lots of new projects which deal with two or more localization techniques. The advantage is that such a system is (usually) more reliable than a single technique, however it can take a long time to setup all the techniques (e.g.: fingerprinting, building map...). Now, we present three works which try to make the most of the sensor fusion for geolocation.

In [Ebner et al., 2015], authors integrate different sensor modalities, namely WiFi, barometer, BLE, step-detection and turn-detection for localization of pedestrians within buildings over multiple floors. To model the pedestrian's movement, which is constrained by walls and other obstacles, they propose a state transition based upon random walks on graphs. The graph corresponds to all valid movements represented by a grid of $20\ cm$ squares. Squares are only placed where they do not intersect walls of the corresponding floorplan. The evaluation of the system within a $4000m^2$ sized building with 4 floors shows an average accuracy of $4\ m$. They also commented on the impact of each technology on the final accuracy.

Authors of [Guimarães et al., 2016] provide a system which deals with multiple gait-model based filtering techniques for movement quantification in combination with a fused positioning mechanism. Magnetic field fluctuations, WiFi readings and movement data are incrementally matched with a feature spot map containing multi-dimensional spatially-related features that characterize the building. Authors claimed to obtain an overall median localization errors between $1.11\ m$ and $1.68\ m$.

Finally, in [Chen et al., 2015], Chen et al. propose a sensor fusion framework for combining WiFi trilateration, SHS and landmarks within a Kalman Filter. WiFi trilateration uses the log-

distance path loss model introduced in Section 1.3.1. Authors do not consider misalignment in their SHS, this means that the smartphone should be held in texting mode. The landmarks investigated in this work include turns, elevators, escalators, stairs and doors. The sensors involved in the identification of these landmarks are the accelerometer, magnetometer, gyroscope, barometer and WiFi. With this approach, they claim to reduce WiFi trilateration precision errors by 65-75% and PDR precision errors by 43-52% to achieved an overall accuracy of 1-2 m.

1.3.4 Surveys and Evaluations of Geolocation Techniques

In the three previous sections, we introduced techniques and approaches for estimating the position of a smartphone. Now, we review surveys and evaluations of geolocation techniques that we found in the literature.

In [Koyuncu and Yang, 2010] and [Al Nuaimi and Kamel, 2011], authors provide information on a subset of techniques (UWB, GPS, Ultrasonic, PDR...) and their accuracy. Comparisons in both studies use precision error claimed by the authors of each paper, this means that techniques were not compared within a common benchmark. Furthermore, both works do not rely on smartphone sensors. In studies from [Khoury and Kamat, 2009] and [Curran et al., 2011], several approaches (radio based) are compared on a common benchmark. Besides, Curran et al. show that on a same technique (WiFi fingerprinting), precision from 2 different approaches can be totally different (from $\approx 2\ m$ to $\approx 25\ m$). Once again results do not rely on smartphone's sensors.

The first smartphone based evaluation occurred in 2015 thanks to the fourth competition of EvAAL[3] (Evaluating Ambient Assisted Living systems through competitive benchmarking) which is a competition that takes place in parallel of the IPIN conference[4]. Unfortunately, the number of participants is very low compared to the number of technologies (4 the first year and 6 the second year). In the same way, the Microsoft Indoor Localization Competition [5] [Lymberopoulos et al., 2015] is held every year since 2014 in parallel of the IPSN conference. In 2016, during the third year of the competition one track introduced "unmodified commercial off-the-shelf devices such as laptops, phones, and tablets". These two competitions have limitations:

- The competition relies on only one track (path followed by the pedestrian), one day and in one (sometimes two) buildings. During the last years, EvAAL team worked on diversification of the competition track: indoor, outdoor, more than one building where entrance is not at the same elevation...
- Systems are evaluated by their authors. We observed some participants following non natural movements/displacements. For instance, some of them did not walked directly to the reference position but they walked by doing right angle. Others shake their smartphone to simulate one more step... To counter these unwanted behaviors, EvAAL staff created a new track in 2016: "Smartphone-based (off-site)". Competitors only have access to smartphone raw data (and a little more information like fingerprints).

Results from the competition are very encouraging, but there are insufficient details in the testing methodology to be able to extrapolate them to a larger population outside the building of

[3] http://evaal.aaloa.org/

[4] ipin-conference.org

[5] https://www.microsoft.com/en-us/research/event/microsoft-indoor-localization-competition-ipsn-2014/

the competition. Recently, respectively in 2013 and 2016, two projects (Evarilos [6] and PerfLoc [7]) highlighted the problems of this kind of competitions and then proposed a methodology and open datasets for benchmarking indoor localization applications. Evarilos project mainly focus on RF technologies [Van Haute et al., 2013] like WiFi, UWB, Bluetooth, infrared, etc. While, PerfLoc also handles inertial sensors (accelerometer, gyroscope, magnetometer) and GPS fixes. Both provide open datasets including reference positions and raw data from 2 offices and 1 open-space (Evarilos [Van Haute et al., 2015]) and four large buildings (PerfLoc [Moayeri et al., 2016]). Moreover, PerfLoc claims that their data recorded follow the methodology provided by the newer ISO 18305:2016 standard [ISO 18305, 2016].

From our point of view, the approaches of these two projects (and especially PerFloc) is the right approach to do geolocation benchmarks but:

- At the time of writing, neither project presented results, they only provided a methodology.
- Four buildings is probably not enough representative of the reality.
- They do not include outdoor use cases.

For these reasons in Chapter 3 we provide a novel benchmark for comparing a subset of techniques: GNSS, SHS, SHS + Point to Curve, WiFi fingerprinting, WiFi trilateration and UWB.

Estimation of device position together with estimation of attitude are the two main components of a Geo AR system. In next section, we will present AR browsers which deal with attitude and position of a smartphone.

1.4 Geo AR Browsers

The number of Augmented Reality applications is growing a lot. All these applications are based on a proprietary AR browser or they reused a framework.

In [Billinghurst et al., 2015], Billinghurst et al. summarized almost 50 years of research and development in the field of AR. They provide a history of important milestones in Augmented Reality, a description of tracking and display technologies and finally an evaluation of the AR systems. This study mainly concerns visual tracking technologies, but they also commented on location based AR.

In this section, we introduced a state of the art of how a Geo AR browser work.

1.4.1 Formats and Authoring

The primary interest of Geo Augmented Reality is to augment the reality with virtual features (timetables of a restaurant, 3D model of an ancient building...). This features are placed in the physical world at known locations. From the Geography Markup Language (GML) [Cox et al., 2002], a feature is an application object that represents a physical entity, e.g. a building, a river, or a person. GML is the XML grammar defined by the Open Geospatial Consortium (OGC) to express geographical features. They introduced three different ways to describe the position of a feature (GML geometries):

- **Point** A single position.

[6] http://www.evarilos.eu
[7] https://perfloc.nist.gov

- **LineString** A list of positions, connected to form a line.
- **Polygon** A list of positions, connected to form a planar area.

For instance, the position of Mont Blanc summit is a Point: *latitude: 45.8326753, longitude: 6.8651661*. Therefore, a data format designed for Geo AR consists in representing a list of features which are described by a geometry and a set of metadata, including a name, a description, pictures, etc.

The Augmented Reality Markup Language (ARML) [Lechner, 2015] is a data standard to describe and interact with AR scenes. It has been developed within the Open Geospatial Consortium (OGC) by a dedicated Working Group. ARML consists of both an XML grammar to describe the location and appearance of virtual objects in the scene. Definitions of features and geometries from GML have been reused in the Augmented Reality Markup Language (ARML). The default coordinate reference system for geometries is geodetic (WGS84 + altitude in meters) (”longitude latitude altitude”). Within ARML, it is possible to associate a descriptions, images, 3D models... to a feature, then describes how they will be rendered in the scene. In addition to geo-referenced features, the language is also capable to handle features from vision tracking.

However, ARML failed to win unanimous support within the AR community. This can be explained by the difficulty and complexity to generate ARML features. Historically, ARML was driven by Wikitude and then adopted by Metaio and Layar. Each of them built a proprietary web interface to generate ARML features but none provides ARML for features based on geolocation. A workaround with Wikitude consists in uploading a KML file to the developer zone.

Other AR languages like KARML [MacIntyre et al., 2011] or Argon-AFrame [MacIntyre, 2017] have been proposed. However, none of them provide an interface for the generation of features.

In 2014, Scioscia et al. [Scioscia et al., 2014] introduce an OSM format based on ontologies. They also provide an extension of JOSM editor to help developers to generate their own format.

Finally, most of developers which deal with Geo AR frameworks use content providers [Belimpasakis et al., 2010, Nicholls and Powertech, 2013, Matuszka et al., 2014].

1.4.2 Frameworks

At the beginning of Augmented Reality, the first systems consisted of a Head-Mounted Display (HMD) linked to a computer [Sutherland, 1968]. Gradually, these technologies become to be wearable. The first demonstration of AR operating in an outdoor environment is the Touring Machine (see Figure 1.21) by Feiner et al. from Columbia University [Feiner et al., 1997]. In 2000, the Naval Research Laboratory presents BARS (Battlefield Augmented Reality System), a new equipment for soldiers [Yohan et al., 2000]. Thanks to an HMD, soldiers have access to a large amount of pieces of information (such as goals, waypoints, and enemy locations). In 2003, Piekarski et al. extended an existing desktop game and developed it into the ARQuake system [Piekarski and Thomas, 2002]. In a same time, Kourogi et al. explore an indoor prototype of a wearable augmented reality system with personal positioning based on walking locomotion analysis [Kourogi and Kurata, 2003a, Kourogi and Kurata, 2003b]. These systems are really expensive and are not comfortable to wear. For these reasons, during years, AR was exclusively used by the researchers and the military.

Since 2011, we observed that AR applications start to be developed for smartphones and consequently they enrich the scope of possible applications (e.g. tourism [Kounavis et al., 2012], GIS

Figure 1.21: Touring Machine system overlays AR information in outdoor environments.

visualization [Nicholls and Powertech, 2013]...). This became possible thanks to the evolution of smartphones computing power and the miniaturization of several sensors (camera, accelerometer, gyroscope...). For instance, in [Bernardos Barbolla et al., 2011], authors explore the applicability of mobile AR to hospitality environments (hotels and similar establishments). Their RF positioning systems (based on WiFi, ZigBee and Bluetooth) provide enough accuracy to perform reliable estimation of the zone/area where the user is staying (error varies between 2 and 3 meters). However, they conclude that this solution is not suitable for indoor AR. After the several major earthquakes which hit the city of Christchurch in 2010 and 2011, HIT Lab NZ developed an application [Lee et al., 2012a] to provide information about destroyed buildings and historical sites that were affected by the earthquakes. The geo-located content is provided in a number of formats including 2D map views, AR visualization of 3D models of buildings on-site, immersive panorama photographs. During the same year, in Italy, Brondi et al. [Brondi et al., 2012] developed LiTe, an augmented view of Piazza dei Miracoli, one of the most famous artistic sites in Italy. Unfortunately, authors concluded on a limited AR experience due to the poor accuracy of GPS data and inertial sensors.

From the multitude of new applications, some projects have specialized in the creation of frameworks for Geo AR applications. This is especially the case with the three professional libraries: Metaio, Wikitude and Layar [Madden, 2011]. The main problem for us is that these libraries are source-closed and not enough flexible to plug our navigation and attitude estimation algorithms. On the other hand, in some projects, authors do not provide a framework but they give explanations on architecture and implementation of a Geo AR framework. For example, Geiger et al. in the AREA project [Geiger et al., 2014, Pryss et al., 2016] explain how POIs from ECEF coordinates are projected on a 2D screen.

Unfortunately, all the projects introduced above are not suitable for testing our algorithms. Therefore, we selected a set of Geo AR frameworks which are open-source or those where the library allows us to modify the pose estimation. Most projects do not provide a description of how they technically work. Thus, we had to browse the source code for each of them. A description for each is reported below and a summary is presented in Table 1.1.

Argon 1 & 2 [Speiginer et al., 2015] is a javascript framework for adding augmented reality content to web applications, and thus by extension it is usable on smartphones. Argon is an inde-

pendent open-source project, supported by the Augmented Environments Lab at Georgia Tech and by Mozilla. Authors created a full eco-system around AR: formats (KARML & HTML), frameworks (Argon 1 & 2) and authoring tools. Contents from Argon 1 were provided in KARML. KARML is a format based on KML (location-based XML format for Google Earth and Maps) extended to include AR-relevant elements. Due to problems of flexibility, KARML has been replaced by an HTML5-based approach in Argon 2. We did not find any public authoring tool to generate KARML or HTML5-based approach in the eco-system of Argon. However, screenshots from their projects[8] make us think that they have it internally. More recently (2016-2017), authors have been working on a package to provide a collection of entities and components for integrating Argon and AFrame. A-Frame is an open-source web framework for building virtual reality experiences originally developed by Mozilla VT team. In its second version, to handle geo virtual scene, Argon starts to deal with WebGL thanks to three.js and cesium.js libraries, this enables the use of more tools for rendering. Geo location and attitude estimation used here are provided by JS APIs (GeoLocation and Orientation Sensor). *Geolocation* specifications [9] from W3C recommendations was completed in 2013 (first version) and 2016 (second version). However, *Orientation Sensor* specifications [10] from W3C are still in draft. Argon also offers the possibility of using computer vision tracking of images and objects (with the Vuforia AR SDK). AR needs lots of resources from smartphone in order to work well. During our study, we observed that an AR browser via a webview is less immersive.

android-augment-reality-framework [11] is both an application and a library for Geo AR. In the framework, authors make available several data flows connectors to retrieve information from projects like Twitter and Wikipedia. Pieces of information is displayed on 2D Android Views with a label. Field of view is not taken into account in the rendering and device camera aspect ratio is not respected. Location estimation is provided by GPS and attitude estimation by a fusion of accelerometer and magnetometer at $10\ Hz$. This project has not been updated since 2014.

BeyondAR [12] is an Android open source framework which allows the user to set objects around the world, he only needs to take care of the content. Developer uses its own OpenGL rendering, but capabilities are really few (images and labels). Camera aspect ratio is not respected, consequently we can observe distortions. However, from the source code of the library, we can read that it is not recommended to use POIs $> 5\ km$ due to the frame conversion. Pose estimation is provided by GPS and a fusion of accelerometer / magnetometer from Android. From this library, author has derived a game (BeyondAR Game) where the goal is to kill 2D monsters in AR. BeyondAR has not been updated since 2014.

DroidAR [13] is a framework for location based and marker based Augmented Reality on Android. Rendering is managed by a custom 3D engine on OpenGL 2.0. It is possible to import 3D models or use one of the basic objects: cube, triangle, square, diamond, path, 3D texts..., but from what we have seen, it is not possible to add system views like labels, icons or text...Camera

[8] http://gtjourney.gatech.edu/projects/campus-tour/experience
[9] https://www.w3.org/TR/geolocation-API/
[10] https://www.w3.org/TR/orientation-event/
[11] https://github.com/phishman3579/android-augment-reality-framework
[12] http://beyondar.com/
[13] https://bitstars.github.io/droidar/

	Rendering	AR Format	Authoring Tool	Open-Source	Platforms	Development Years
android-augment-reality-framework	2D Views	Hard coded	No	Yes	Android	2011-2014
Argon	three.js (WebGL)	KARML	Proprietary	Mainly	JS	2009-2017
BeyondAR	OpenGL (images, labels)	Hard coded	No	Yes	Android	2013-2014
DroidAR	OpenGL (3D models, basic objects)	Hard coded	No	Yes (v1)	Android	2010-2013
Mixare	2D Views	Hard coded	No	Yes	Android & iOS	2010-2012
PanicAR	2D Views	Hard coded	No	No	Android & iOS	2011-2014

Table 1.1: List of Geo AR frameworks which allow to customize camera pose estimation.

aspect ratio is not respected either for rendering. In the virtual scene, POIs are placed relatively to user position (center of the frame) and user position came from a custom fusion of GPS and SHS. Finally, attitude estimation is provided by built-in fusion from Android SDK. It is the most complete open source library we tested but DroidAR 1.0 has been discontinued in 2013 and the last version is not public.

Mixare [14] is a free open source augmented reality browser (Android & iOS). It is not officially a framework but as the project is open-source, the core of the application can be re-used for other purposes. Application shows POIs from wikipedia according to user location. POIs are rendered as labels over the camera stream, then when a POI is clicked the Wikipedia page opens. Labels positions are calculated using GPS location and smartphone orientation. Attitude is provided by the Android fusion of accelerometer and magnetometer (no gyroscope). Then labels' positions are projected on the screen. We noticed that the field of view of the device camera was not taken into account in the projection and the rendering is really noisy. Mixare project has not been updated since 2012.

PanicAR is a free but not open-source Geo AR framework for iOS and Android. POIs are shown as labels where the name, description and icon are customizable. Attitude does not seem to be well estimated around the Z-axis (roll) of the smartphone. PanicAR has not been updated since 2014.

[14] http://www.mixare.org/

Unfortunately, all the frameworks we presented are not easily customizable. All of them use the positioning estimation from GNSS and the attitude estimation from the built-in filter. Moreover, the field of view of the camera is partially (or not) taken into account for the rendering. For example, features are moving faster in the scene then in the video stream from the camera.

For these reasons, in Chapter 5 we propose a lightweight implementation of Geo AR framework. The framework can handle several positioning estimation algorithms and attitude estimation filters. Furthermore, we will pay a particular attention to take into account the field of view of the camera.

1.4.3 Evaluation of Geo AR Systems

In the previous section we have reviewed AR frameworks for building AR applications. However, a key activity in developing AR experiences is evaluating the AR application. In [Dünser et al., 2008], Dunser et al. highlight that in the AR papers they collected, only a small percentage (8-10%) contained any formal evaluation. The report identified three main areas for types of user studies: perception, performance/interaction and collaboration. In [Billinghurst et al., 2015], they proposed a methodology that combines different types of evaluation techniques, and show how this was applied in the design of the BARS system [Yohan et al., 2000].

More generally, in [Dünser et al., 2012], authors evaluated the usefulness of an AR browser in an application. They have conducted a user study comparing navigation completion time and distance travel using (1) a classical 2D map, (2) an AR view, (3) a combined map and AR view. They conclude that they found no overall difference in task completion time, but found evidence that AR browsers are less useful for navigation in some environment conditions. Users preferred the combined AR + Map condition, and felt that there were significant problems with using the AR view alone for navigation.

However, to the best of our knowledge, all the papers which evaluate Geo AR systems include the user in the loop. There does not exist any research work which evaluate the precision of a Geo AR system. This is why, in Chapter 4, we propose a method to evaluate precision of Geo AR browsers.

In the next chapter, we propose the first benchmark using a motion lab for the purpose of comparing and evaluating filters from the literature on a common basis.

Chapter 2

Attitude Estimation: Benchmarks and Improvements

Contents

2.1 Experimental Protocol in a Motion Lab 36
2.1.1 Ground Truth 36
2.1.2 Typical Smartphone Motions 37
2.1.3 Introducing Magnetic Perturbations 37
2.1.4 Different Devices 39
2.1.5 Common Basis of Comparison and Reproducibility 40
2.2 Attitude Estimation Algorithms Selected for Comparison 41
2.3 Design of a New Algorithm for Limiting Magnetic Perturbations Impact . 42
2.4 A Deep Analysis of Results 43
2.4.1 Importance of Calibration 44
2.4.2 The Difficulty with Noises for Kalman Filters 45
2.4.3 Bias Consideration 46
2.4.4 Behaviors during Typical Smartphone Motions 47
2.4.5 Impact of Magnetic Perturbations 47
2.4.6 Pitch and Roll in Augmented Reality Scenario 48
2.4.7 Comparison with Device-Embedded Algorithms 50
2.4.8 Empirical Computational Complexity 51
2.4.9 Relevant Sampling Rates 51
2.4.10 Parameter Adjustment for a Balance Between Stability and Precision . 52
2.5 Conclusions 54

Attitude estimation of a solid was first introduced in 1965 by G. Wahba [Wahba, 1965]. At the beginning, estimation was used for satellites and spacecrafts with specific sensors like solar cells, horizon scanners and magnetometers. Today, modern smartphones embed a triad of sensors which make it possible to leverage existing attitude estimation algorithms. One might naturally wonder

whether these algorithms yield estimation that are accurate enough to be used for Augmented Reality (AR) with smartphones? This is what is explored in this chapter.

Existing algorithms have been extensively investigated in various domains such as: robotics [Ojeda and Borenstein, 2002], aerospace [Lefferts et al., 1982], unmanned aerial vehicles [Euston et al., 2008], bio-logging [Fourati et al., 2011], indoor positioning [Renaudin and Combettes, 2014]. However, the particular context of smartphones carried by pedestrians brings new challenges due to singular accelerations and magnetic perturbations, which sometimes invalidate the basic hypotheses that underly state-of-the-art attitude estimation algorithms. In particular, the absence of model describing the smartphone motions (preventing control), and the presence and variations of magnetic perturbations during the estimation phase, both introduce additional difficulties.

We investigate the precision of attitude estimation algorithms in the context of commodity smartphones carried by pedestrians with a specific focus on AR. We consider eight typical motions (such as texting, phoning, running, etc.) with various impacts on external accelerations, as well as the presence/absence of magnetic perturbations typically found in indoor environments. We systematically analyze, compare and evaluate eight state-of-the-art algorithms (and their variants). We precisely quantify the attitude estimation error obtained with each technique, owing to the use of a precise ground truth obtained with a motion capture system. We make our benchmark available[1] and pay attention to the reproducibility of results. We analyze and discuss the obtained results and report on conclusions. We also present a new technique which helps in improving precision by limiting the effect of magnetic perturbations with all considered algorithms.

We first explain our experimental protocol to benchmark attitude filters in Section 2.1. We present the considered algorithms in Section 2.2 and our new technique in Section 2.3. We finally report on obtained results and lessons learned in Section 2.4 before concluding in Section 2.5.

2.1 Experimental Protocol in a Motion Lab

In this section, we explain our experimental methodology. A total of 126 trials have been conducted by 3 peoples with 3 different smartphones, following several typical motions in an environment with low and high magnetic disturbances[2].

2.1.1 Ground Truth

Reference measurements have been made by a Qualisys system. This technology provides quaternions with a precision of $0.5°$ of rotation. Our room is equipped with 20 Oqus cameras connected to a server and a Qualisys Tracker software with a sampling rate at 150Hz. For the purpose of aligning timestamps of our ground truth data with the one of smartphone's sensors, we used a slerp interpolation [Shoemake, 1985]. The motion tracker reference frame has been aligned with EF using room orientation provided by architects. The room is a $10m \times 10m$ square motion lab[3] (see Figure 2.1). In this room, we observed that the magnetic field is almost homogeneous from a sub-place to another (variations are less than $3\mu T$), and with negligible variations over time.

[1] `http://tyrex.inria.fr/mobile/benchmarks-attitude`

[2] Two motions have not be conducted during high magnetic disturbances.

[3] See: http://kinovis.inrialpes.fr

Figure 2.1: Kinovis room at Inria, Grenoble, France.

A smartphone handler with infrared markers has been created with a 3D printer for this study and its markers have been aligned with SF (see Figure 2.2).

2.1.2 Typical Smartphone Motions

We identify 8 typical smartphone motions, inspired from [Renaudin et al., 2012]:

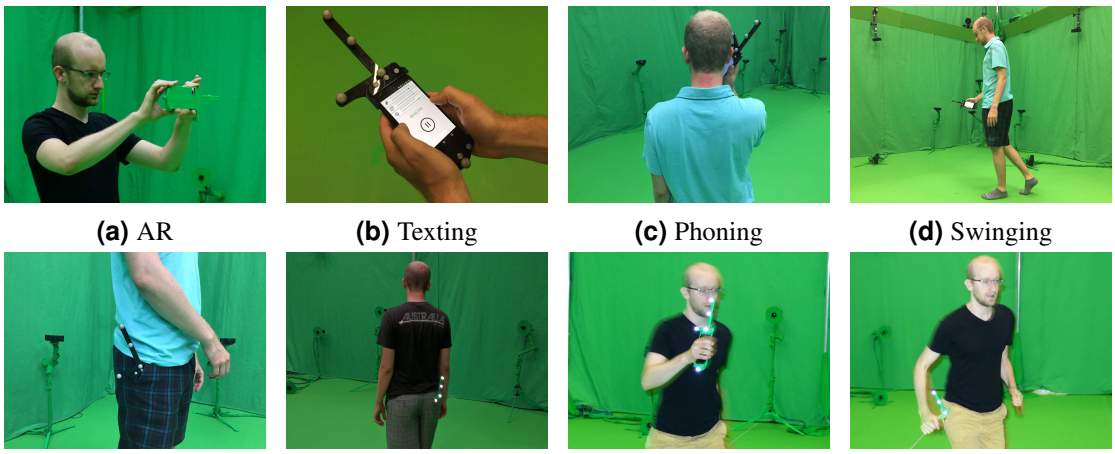
- Querying the context in **augmented reality** (see Figure 2.2a).
- Walking while user is **texting** a message (see Figure 2.2b).
- Walking while the user is **phoning** (see Figure 2.2c).
- Walking with a **swinging** hand (see Figure 2.2d).
- Walking with the device in the **front pocket** (see Figure 2.2e).
- Walking with the device in the **back pocket** (see Figure 2.2f).
- **Running** with the device in the **hand** (see Figure 2.2g).
- **Running** with the device in the **pocket** (see Figure 2.2h).

AR motion is a slow motion typically found during AR experiences. Other motions happen when pedestrians move and are relevant for navigation applications. Each motion is characterized by a particular external accelerations. The Table 2.1 shows some statistics on external acceleration magnitude grouped by motion, for the 126 tests. The second column of Table 2.1 shows the average (AVG) of external acceleration magnitude grouped by motion where the third column shows the estimated one from Eq.1.15.

During tests, we observed that external accelerations almost never reach zero because the device is always moving, and constant speed is very unlikely when the device is held or carried in a pocket. However, we noticed a high variety of external accelerations: some motions involve external accelerations that are 20 times lower than gravity while others (like running hand) are closer to twice the value of gravity. We also noticed that the maximum swing of accelerometer ($\pm 2g$) is often reached during our running experiments.

2.1.3 Introducing Magnetic Perturbations

During tests, we noticed that magnetic disturbances are always present in indoor-environments, and they vary between different buildings. This is mainly due to building structure. We also ob-



(a) AR **(b)** Texting **(c)** Phoning **(d)** Swinging

(e) Front Pocket **(f)** Back Pocket **(g)** Running Hand **(h)** Running Pocket

Figure 2.2: The eight typical motions for a smartphone.

	Ext. Acc. AVG $(m.s^{-2})$	Ext. Acc. AVG est. $(m.s^{-2})$	Ratio	$> 0.5m.s^{-2}$	$> 1.5m.s^{-2}$	$> 5m.s^{-2}$
AR	0.56	0.24	2.39	46.4%	2.4%	0.0%
Texting	1.08	0.61	1.81	83.5%	20.7%	0.1%
Phoning	1.08	0.57	1.96	83.1%	21.0%	0.1%
Front Pocket	2.48	1.40	1.81	97.1%	68.2%	7.5%
Back Pocket	2.53	1.23	2.10	97.5%	72.0%	7.7%
Swinging	5.28	2.30	2.42	99.7%	96.8%	52.5%
Running Pocket	9.56	5.93	1.61	99.6%	98.2%	84.4%
Running Hand	16.34	8.44	2.02	99.9%	99.7%	98.6%

Table 2.1: Statistics on Magnitude of External Accelerations for each motion

served in some cases, if user is close to heaters, electrical cabinets or simply close to a wall, magnitude of magnetic field can grow up to 150 μT during few seconds, that is 3 times more than Earth's magnetic field (see e.g. Figure 2.5).

The motion capture system used is located in a room with low and constant magnetic perturbations. In order to reproduce typical magnetic perturbations of indoor environments inside the motion lab, we used several magnetic boards (see Figure 2.3). This allowed us to introduce magnetic perturbations similar to the ones described above in Figure 2.5. Specifically, during the 2 minutes tests, we brought the device to a few centimeters away from magnetic boards; and we repeated this 3 or 4 times (see Figure 2.4).

The Table 2.2 shows some statistics on External Magnetic Field Magnitude (EMFM). When we do not consider white magnetic boards, magnitude of magnetic field is not totally equal to the magnitude of Earth's magnetic field, so perturbations cannot be entirely omitted. If we add magnetic boards, a difference between the two magnitudes can be clearly observed (column 2). In

Figure 2.3: Magnetic boards for building structure and heaters simulation.

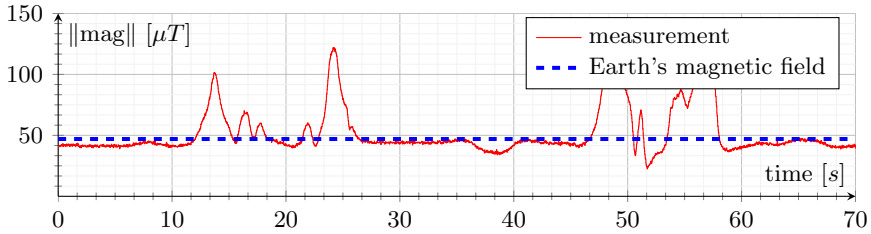


Figure 2.4: Magnitude of magnetic field measurements and Earth's magnetic field during our simulation with magnetic boards.

average, 26.5% of the time, magnetic perturbations have a magnitude higher than $> 5\mu T$ and they not exist if we remove magnetic boards.

	EMFM (μT)	EMFM Estimated (μT)	Ratio	STD (μT)	$> 0.5\mu T$	$> 1.5\mu T$	$> 5\mu T$
High	29.57	18.61	1.65	43.09	46.7%	31.2%	26.5%
Low	7.12	5.18	1.40	1.99	13.0%	0.2%	0.0%

Table 2.2: Statistics on Magnitude of Magnetic Field with low and high magnetic perturbations

2.1.4 Different Devices

Measurements have been recorded with 3 different smartphones from 2 manufacturers. The 3 smartphones used are a LG Nexus 5, an iPhone 5 and an iPhone 4S. We implemented a log application[4] for Android and iOS. Table 2.3 summarizes sensors specifications for the 3 devices.

For the purpose of aligning timestamps of magnetic field and gyroscope data with data obtained from accelerometer, we used a linear extrapolation. In order to keep the focus on a real-time process, interpolation is not allowable here. We choose to align data at 100Hz. Moreover, for each trial, we chose to process 31 algorithms at 4 sampling rates and with 7 different calibrations, that

[4] `https://github.com/tyrex-team/senslogs`

	Accelerometer	Gyroscope	Magnetometer
iPhone 4S	STMicro STM33DH 100Hz	STIMicro AGDI 100Hz	AKM 8975 40Hz
iPhone 5	STMicro LIS331DLH 100Hz	STIMicro L3G4200D 100Hz	AKM 8963 25Hz
Nexus 5	InvenSense MPU6515 200Hz	InvenSense MPU6515 200Hz	AKM 8963 60Hz

Table 2.3: Sensors specifications with the max. sampling rate

is a total of more than 110 000 tests and 804 millions quaternions compared.

2.1.5 Common Basis of Comparison and Reproducibility

To ensure a reasonably fast convergence of algorithms, we initialize the first quaternion (for estimation algorithms) using the first measurement of accelerometer and magnetometer. In addition, we discard the first five seconds from our results, to allow time for filter to converge.

Most smartphone APIs (including Nexus 5 and iPhones) provide both calibrated and uncalibrated data from magnetometer and gyroscope[5], and only uncalibrated data from accelerometer. Calibration phases can be triggered by the Android operating system at anytime. However, we notice that the gyroscope bias is removed during static phases and the magnetometer is calibrated during the drawing of an infinity symbol. For iOS devices, the magnetometer calibration must be explicitly triggered by the user. The exact calibration algorithms embedded in both iOS and Android are not disclosed so we consider them as "black-boxes".

To investigate the impact of calibration, we also developed a custom calibration procedure: every morning, we applied our own implementation of the calibration based on Bartz et al. [Bartz, 1987] to remove soft and hard iron distortions from magnetometer and based on Frosio et al. [Frosio et al., 2013] for the accelerometer. In addition, for all calibrations we applied a scale to adjust magnitude to $9.8m.s^{-2}$ for accelerometer and Earth's magnetic field magnitude for magnetometer. For the gyroscope, we simply remove the bias by subtracting measured values in each axis during static phases.

The precision error is reported using the Mean Absolute Error (MAE) on the Quaternion Angle Difference (QAD) [Huynh, 2009]. It allows to avoid the use of Euler angles with the gimbal-lock problem. The formula of QAD is defined by:

$$\theta = cos^{-1}(2\langle q1, q2\rangle^2 - 1). \tag{2.1}$$

Since the accuracy of the system that provides the ground truth is $\pm 0.5°$, we consider that two algorithms exhibiting differences in precision lower than $0.5°$ rank similarly.

[5] not from iOS API

2.2 Attitude Estimation Algorithms Selected for Comparison

An Allan variance analysis of sensors noises characteristics was carried out in Section 1.2.2. Most of the filters from the literature do not consider such precise models. We will now review the state-of-the-art algorithms that we consider in our study. We have selected 8 filters from the literature which are representative of the different techniques developed for solving the attitude estimation based on IMU sensors. Our selection of algorithms can roughly be divided into two categories: those based on observers, and those based on KFs (with their EKF, UKF, and AKF variants). We summarize the main principles and objectives of each algorithm (see [Michel et al., 2015] for a more formal description of each algorithm using a common notation). For reproducibility purposes, we also indicate parameters that we used with each algorithm – which we set in accordance with authors guidelines found in their papers. We also consider the "black-box algorithms" embedded in Android and iOS. The considered algorithms are the followings:

Madgwick et al. [Madgwick et al., 2011]. This filter is a Gradient Descent (GD) based algorithm designed for pedestrian navigation. The authors propose to consider magnetic field deviations only on North-East plane using the following technique: ${}^{E}\text{mag} = \begin{bmatrix}0 & m_y & m_z\end{bmatrix}^T$, where $m_y = \sqrt{h_x^2 + h_y^2}$, $m_z = h_z$ and $h = \hat{q}^{-1} \otimes {}^{S}\text{mag} \otimes \hat{q}$. *Madgwick* is the common implementation of the filter, and *MadgwickB* is the same with a gyro bias. Parameters: $\beta = 0.08,\ \zeta = 0.016$.

Martin et al. [Martin and Salaün, 2010]. This filter is an observer with a new geometrical structure (invariant observer). The authors introduce new measurements based on the cross product of acceleration and magnetic field. *Martin* is the common implementation of the filter. Parameters: $l_a = 0.7, l_c = 0.1, l_d = 0.01, n = 0.01, o = 0.01, k = 10, \sigma = 0.002$.

Mahony et al. [Mahony et al., 2008]. This filter is a complementary filter designed for aerial vehicles. The main idea is to calculate the error by cross multiplying measured and estimated vectors. *Mahony* is the common implementation of the filter. *MahonyB* is the implementation which takes into account a gyro bias. Parameters: $\beta = 1,\ \zeta = 0.2$.

We provided a new variant of this filter (*MahonyMartin*), the observation vector from magnetometer is replaced by the cross product of accelerometer and magnetometer from Martin et al. Parameters: $\beta = 0.2,\ K_a = 1,\ K_c = 0.5$.

Fourati et al. [Fourati et al., 2011]. This filter is a mix between a complementary filter algorithm and the Marquardt approach designed for bio-logging. *Fourati* is the common implementation of the filter. *FouratiExtAcc* is an extension which takes external accelerations into account using Eq. (1.15)). Parameters: $\beta = 0.3$, $K_a = 2$ and $K_m = 1$. $K_a = 0$ when $\gamma_{\text{acc}} = 0.5m.s^{-2}$.

In the same way we provided *MahonyMartin* variant, we proposed a new filter based on Fourati et al. algorithm, *FouratiMartin* which uses the cross product of accelerometer and magnetometer. Parameters: $\beta = 0.3$, $K_a = 2$, $K_c = 1$.

Choukroun et al. [Choukroun et al., 2006]. This filter provides a linearization of measurement equations. A KF is proposed and guarantees a global convergence. *Choukroun* is the common implementation of the filter. No other parameters than the Kalman noises need to be fixed.

Renaudin et al. [Renaudin and Combettes, 2014]. This filter is an EKF designed for Pedestrian Dead Reckoning (PDR). In addition to Eqs. (1.11) and (1.12), they use two others properties:

$$acc_{t+1} = q_{\omega}^{-1} \otimes acc_t \otimes q_{\omega}, \tag{2.2}$$

$$mag_{t+1} = q_{\omega}^{-1} \otimes mag_t \otimes q_{\omega}, \tag{2.3}$$

where q_{ω} is interpreted as a rotation between two successive epochs. Eqs. (1.11), (1.12), (2.2) and (2.3) are applied only during QSF periods. The detector for QSF works by analyzing variance of acceleration and magnetic field measurements on a small window ($\approx 0.2s$). This filter has to be initialized ($\approx 5s$ at the beginning) without external accelerations and magnetic perturbations (mostly outside). *Renaudin* is the common implementation of the filter. In *RenaudinBG*, the gyro bias estimation is added where, gradients update from Eqs. (2.2) and (2.3) are considered. *RenaudinExtaccExtmag* takes both QSF detectors into account. Parameters: QSF_Window $= 10$, $\gamma_{\text{QSF\_Acc}} = 3.92 m.s^{-2}$, $\gamma_{\text{QSF\_Mag}} = 6\mu T$, $\text{outliers}_{\text{QSF\_Acc}} = 4.90 m.s^{-2}$, $\text{outliers}_{\text{QSF\_Mag}} = 8\mu T$.

Sabatini et al. [Sabatini, 2006]. This filter is an EKF which considers external acceleration and magnetic perturbations as explained in Section 1.2.4. *Sabatini* is the common implementation of the filter. *SabatiniExtacc* and *SabatiniExtmag* takes respectively external accelerations and magnetic perturbations into account. We did not implement the gyro bias part of this filter. Parameters: $\gamma_{\text{acc}} = 0.5\ m.s^{-2}$, $\gamma_{\text{mag}} = 15\ \mu T$, $\gamma_{\theta} = 10°$, $\text{mov\_average}_{\text{mag}} = 0.1s$

Ekf is the common implementation of the Extended KF.

OS The Android API of Nexus 5 and iOS API of iPhones also provides quaternions generated by undisclosed "black-box" algorithms. We include them in our comparisons.

2.3 Design of a New Algorithm for Limiting Magnetic Perturbations Impact

The presence of magnetic perturbations in indoor environments is well-known [Bachmann et al., 2004]. For example, Figure 2.5 illustrates variations of the magnetic field observed inside Inria's research center compared to Earth's magnetic field. To limit the impact of such magnetic perturbations, in addition to selected filters, we propose a new approach that further builds on the idea of detectors à la (1.16). The overall principle is twofold: (1) during periods when we detect magnetic perturbations, we can discard magnetometer measurements for a short period ($\approx 2 - 3s$) so that more importance are given to gyroscope measurements; (2) this period should be reasonably short-enough so that the impact of gyroscope's bias[6] is limited.

We propose an improvement of the magnetic perturbation detector (Eq. (1.16)) adapted to the pedestrian context. When a person is moving with a normal speed (walk) in a building, we have observed huge variations of magnitude of magnetic field $\left\|{}^{S}\text{mag}\right\| > 100\ \mu T$ (see for example Figure 2.5 at $t = 24s$). The main problem with the detector (1.16) is to find a proper γ_{mag} which should be: (i) high enough not to discard magnetometer measurements due to low magnetic perturbations omnipresent in an indoor environment and (ii) small enough to reject high perturbations

[6] We experimentally measured the drift due to gyroscope's bias integration as approximately $5\ °/min$.

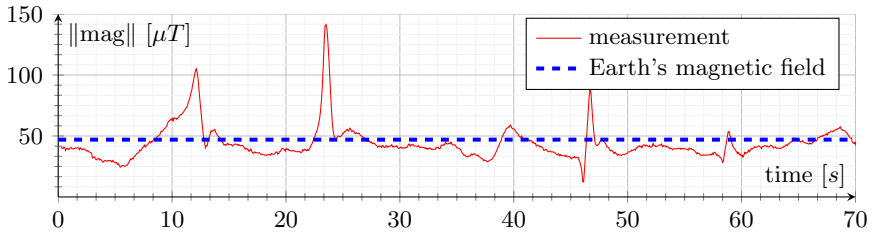


Figure 2.5: Magnitude of magnetic field measurements and Earth's magnetic field in the indoor environment of Inria building in Grenoble.

which affect attitude estimation (such as those coming from the proximity of e.g. heaters, see: Section 2.1.3).

When the threshold of Eq. (1.16) is reached, generally the filter is already diverging. This means that when this detection occurs, and therefore when gyroscope integration starts, magnetometer measurements involving perturbations below the threshold have already impacted attitude estimation.

Figure 2.6 presents our new technique to limit the impact of magnetic perturbations. The principle is that we reprocess the filter for the $t_{\text{mag, rep}}$ last seconds without magnetometer measurements (Eq. (1.12)). When the detection occurs, attitude estimation is immediately replaced by these values. This technique avoids the attitude divergence during the $t_{\text{mag, rep}}$ last seconds before the detection (Eq. (1.16)). This technique can be used for real-time attitude estimation (time for reprocessing being negligible when compared to $t_{\text{mag, rep}}$), in which case a discontinuity of some degrees can be observed when the detection occurs (see Figure 2.7).

During periods of magnetic perturbation, Eq. (1.16) can be true for a small duration. This is because magnitude of magnetometer measurement can be similar to Earth's magnetic field magnitude during a perturbation phase, it depends on the direction of the perturbation. For this purpose a last condition is added: Eq. (1.12) can be used only if there is no detection (Eq. (1.16)) during the last $t_{\text{mag, nopert}}$ seconds.

This technique works with all filters where updates (Eq. (1.11)) from magnetometer can be temporarily removed (which is the case of all algorithms considered here). An important prerequisite is magnetometer calibration. In a context without magnetic perturbations, the magnitude of magnetometer measurements should be equal to the magnitude of Earth's magnetic field.

In addition to the algorithms presented before, we also consider 2 new algorithms based on the aforementioned technique. The first one, *MichelObsF*, is an implementation of the technique where f is the observer function from Fourati et al. [Fourati et al., 2011]. The second algorithm, *MichelEkfF*, is designed with f set to the well known EKF filter from the literature. For both algorithms we use the following common parameters: $\gamma_{\text{mag}} = 15\mu T$, $t_{\text{mag, nopert}} = 2s$ and $t_{\text{mag, rep}} = 3s$.

2.4 A Deep Analysis of Results

We made available the whole benchmark including the 110000+ of 2-minute results and the 126 datasets at: `http://tyrex.inria.fr/mobile/benchmarks-attitude`. Tests can thus be reproduced. This benchmark makes it possible to evaluate new filters over a common

Section etAlgoLined **Data:**
f (gyr, acc, mag, dT, mag_update) is a basic filter (KF or observer) where mag_update is a boolean indicating whether magnetometer measurements have to be used.
vec_states_and_values is a moving vector keeping track of filter state and measurements from sensors over a sliding window.
last_mag_pert is the elapsed time since the last magnetic perturbation detected. Initially it is set to 0.

```
// Detecting magnetic perturbations
```
mag_update$_k$ = abs($\|^S\text{mag}\| - \|^E\text{mag}\|$)) $< \gamma_{\text{mag}}$

```
// Enforcing minimal durations
```
if *mag_update*$_k$ **then**
  last_mag_pert = last_mag_pert + dT
  if *last_mag_pert* $< t_{mag,\ nopert}$ **then**
    mag_update$_k$ = false
  end
else
  last_mag_pert = 0
end

```
// Reprocessing last values without mag data
```
if *!mag_update*$_{k-1}$ ***and*** *mag_update*$_k$ **then**
  f.setState(vec_states_and_values.first)
  foreach *element e of vec_states_and_values* **do**
    f(e.gyr, e.acc, e.mag, e.dT, false)
  end
end

attitude, state = f(gyr, acc, mag, dT, mag_update$_k$)

```
// Store state and measurements
```
vec_states_and_values$_k$ = state, gyr, acc, mag, dT
remove lines of vec_states_and_values where elapsed time $> t_{\text{mag, rep}}$

Figure 2.6: Pseudo-code for limiting the impact of magnetic perturbations.

ground truth, and to compute additional analytics like e.g. precision errors using Euler angles. In this Section we report on a few discussions, backed by aggregated views on a fraction of the obtained results.

2.4.1 Importance of Calibration

We tested attitude estimation algorithms in 6 different situations where magnetometer, gyroscope and accelerometer are either calibrated or not. Table 2.4 presents the results, showing that precision is impacted in the same way with all algorithms.

In a context free from magnetic perturbations, the magnitude of uncalibrated magnetic field is about $350\mu T$. This is why it is impossible to estimate attitude if calibration of hard iron distortions has not be done before. The gyroscope calibration phase is mostly important during periods with no update from accelerometer and magnetometer values. If gyroscope is not calibrated, the integration drift will grow from $5°.\text{min}^{-1}$ to $20°.\text{min}^{-1}$. We observe that the accelerometer calibration does not significantly improve the precision of attitude estimation for the considered datasets. The

	Mag: No Gyr: No Acc: No	Mag: Yes Gyr: No Acc: No	Mag: Yes Gyr: No Acc: Yes	Mag: Yes Gyr: Yes Acc: No	**Mag: Yes Gyr: Yes Acc: Yes**	Mag: OS Gyr: OS* Acc: No
Choukroun	95.1°	16.5°	16.5°	9.9°	**10.0**°	17.3°
Fourati	83.7°	15.6°	15.5°	10.3°	**10.4**°	16.3°
Madgwick	77.5°	18.2°	18.2°	8.1°	**8.1**°	17.7°
Renaudin	82.2°	19.5°	19.5°	8.0°	**8.1**°	18.1°
Ekf	79.8°	19.4°	19.4°	7.9°	**8.0**°	18.2°
MichelEkfF	82.0°	20.1°	20.1°	6.9°	**7.0**°	18.2°
MichelObsF	82.1°	13.6°	13.5°	5.9°	**5.9**°	15.1°

* Not available for iOS devices

Table 2.4: Precision of attitude estimation according to calibration with all motions

way we performed calibration (see Section 2.1.5) provides a significantly better precision in attitude estimation than the calibration performed by device-embedded algorithms.

2.4.2 The Difficulty with Noises for Kalman Filters

KFs are often used in attitude estimation where white noises naturally model physical sensors noises as we described in Section 1.2.2. We know from theory that KFs converge when the object (the smartphone in our case) is static and magnetometer values correspond to the Earth's magnetic field. However, this is not the case in the context that we consider. The magnitude of external accelerations and magnetic perturbations experienced by the smartphone is much higher than its physical sensors noises.

With values for sensors noises experimentally extracted (as commonly found in the literature), filters yield high precision errors and diverge quickly. This is shown in Table 2.5 where *ChoukrounSn*, *RenaudinSn* and *EkfSn* respectively denote the algorithms initialized with values for noises measured from physical sensors.

	AR	Texting	Phoning	Front Pocket	Back Pocket	Swinging	Running Pocket	Running Hand
Choukroun	5.1°	4.3°	4.4°	4.8°	4.6°	6.3°	7.9°	21.1°
ChoukrounSn	15.6°	20.6°	15.9°	17.8°	16.9°	11.5°	17.6°	35.2°
Ekf	4.5°	4.0°	3.7°	4.6°	4.6°	5.9°	8.2°	16.8°
EkfSn	44.0°	57.8°	36.1°	20.6°	30.8°	29.1°	23.3°	54.1°
Renaudin	4.5°	3.8°	3.7°	4.7°	4.6°	6.1°	8.5°	17.9°
RenaudinSn	20.8°	18.5°	17.8°	17.3°	18.4°	11.4°	17.4°	36.5°

Table 2.5: Precision of attitude estimation according to sensor noises without magnetic perturbations.

KFs can still give better results in this context, provided we adapt the "noises values" in a way that does not reflect anymore physical sensors noise, but that instead takes into account the relative importance of sensor measurements in this context. Gyroscope measurements are not impacted by external accelerations nor magnetic perturbations. In our context, we observed that giving more importance to gyroscope measurements (compared to magnetometer and accelerometer measurements) yields better results (despite convergence being a bit longer). Experimentally we obtained the best results (See *Choukroun*, *Renaudin* and *Ekf* in Table 2.5) by using the following "noises values": $\sigma_{\text{acc}} = 0.5,\ \sigma_{\text{mag}} = 0.8,\ \sigma_{\text{gyr}} = 0.3$ for all KFs[7].

Applying KFs remains non trivial, because the notion of noise to model in this context goes much beyond the setting in which initial KFs were designed.

Observers and KFs exhibit similar results for low to moderate external accelerations. For higher accelerations (typically found when swinging and running), observers were found to improve precision. This is especially the case for *MichelObsF* that outperforms *MichelEkfF*, as shown in Table 2.7.

2.4.3 Bias Consideration

Many existing filters propose to estimate sensors bias and in particular gyroscope bias. For example, in observers, typical procedures use residuals between reference and estimated quaternion to estimate bias (e.g. [Mahony et al., 2008, Madgwick et al., 2011]). However, in our context, residuals do not only originate from gyroscope bias but also from magnetic perturbations and external accelerations. Furthermore, a calibration phase is performed in a previous stage.

We can thus wonder how useful classical bias estimation techniques are in our setting. Table 2.6 compares the results obtained with two variants of each filter: one with bias estimation and one without. We observe that bias estimation seems unnecessary in our context of study. This can be explained by the calibration phase of the gyroscope that was carried out before. We remark however that bias estimation can still be useful for situations where the gyroscope is not calibrated. In this particular case, the precision of attitude estimation is improved with bias estimation, provided external accelerations remain small.

	AR	Texting	Phoning	Front Pocket	Back Pocket	Swinging	Running Pocket	Running Hand
Madgwick	4.8°	4.1°	4.6°	4.9°	5.0°	5.8°	7.1°	16.5°
MadgwickB	5.2°	4.8°	5.4°	5.8°	6.2°	11.5°	10.5°	19.8°
Mahony	5.0°	4.6°	4.2°	5.1°	5.2°	7.5°	7.9°	11.2°
MahonyB	5.6°	4.9°	4.7°	6.1°	5.7°	9.9°	13.1°	26.4°
Renaudin	4.5°	3.8°	3.7°	4.7°	4.6°	6.1°	8.5°	17.9°
RenaudinBG	4.5°	3.7°	3.8°	4.5°	4.6°	6.9°	12.8°	19.3°

Table 2.6: Precision of attitude according to bias estimation without magnetic perturbations.

[7] except for the Linear KF from Choukroun where we adapt these values for the linearized model: $\sigma_{\text{acc}} = 0.3,\ \sigma_{\text{mag}} = 0.3,\ \sigma_{\text{gyr}} = 0.5$

2.4.4 Behaviors during Typical Smartphone Motions

Table 2.7 compares the precision of attitude estimation for each motion. We observe a negative correlation between magnitude of external accelerations and precision of attitude estimation. This is verified for all algorithms.

	AR	Texting	Phoning	Front Pocket	Back Pocket	Swinging	Running Pocket	Running Hand
OS	7.1°	5.9°	5.8°	12.7°	13.2°	20.3°	24.4°	62.0°
Choukroun	5.1°	4.3°	4.4°	4.8°	4.6°	6.3°	7.9°	21.1°
Madgwick	4.8°	4.1°	4.6°	4.9°	5.0°	5.8°	7.1°	16.5°
Mahony	5.0°	4.6°	4.2°	5.1°	5.2°	7.5°	7.9°	11.2°
Fourati	4.8°	4.0°	4.4°	4.6°	4.8°	5.3°	6.3°	6.6°
FouratiExtacc	4.9°	5.4°	4.7°	6.0°	5.7°	8.4°	12.2°	29.1°
Sabatini	4.5°	4.0°	3.7°	4.6°	4.6°	5.9°	8.2°	16.8°
SabatiniExtacc	4.5°	4.5°	4.0°	5.5°	5.0°	9.7°	15.0°	33.5°
Renaudin	4.5°	3.8°	3.7°	4.7°	4.6°	6.1°	8.5°	17.9°
RenaudinExtacc	4.5°	3.8°	3.7°	4.8°	4.8°	6.0°	8.0°	30.3°
MichelObsF	**4.8°**	**3.9°**	**4.4°**	**4.6°**	**4.8°**	**5.3°**	**6.3°**	**6.6°**
MichelEkfF	**4.5°**	**4.0°**	**3.7°**	**4.6°**	**4.6°**	**6.0°**	**8.2°**	**16.8°**

Table 2.7: Precision of attitude estimation according to typical motions without magnetic perturbations.

We observe that two algorithms stand out in terms of precision: *Fourati* and *MichelObsF*.

Table 2.1 presents the left term μ of detector (Eq. (1.15)) and the magnitude of external accelerations (extracted from the ground truth). We observe that the two series are highly correlated ($\rho > 99\%$). This suggests that it is possible to reasonably distinguish periods with high external accelerations.

We also observe that filters which take external accelerations into account do not yield better precision than others. This can be explained by long periods of perturbations (external accelerations) without the smartphone becoming completely static. Moreover, filters are very sensitive to false detections which make them quickly diverge. An interesting perspective for the further development of filters in this context would be to investigate how to better leverage the detection of periods with high external accelerations in order to improve the precision of attitude estimation (Table 2.7).

2.4.5 Impact of Magnetic Perturbations

We tested different motions in the presence/absence of magnetic perturbations (Section 2.1.3). Results are shown in Table 2.8.

We observe that filters which implement a magnetic perturbations detector do not systematically exhibit a better behavior when compared to their native variant. However, if we extend them

	AR	Texting	Phoning	Front Pocket	Back Pocket	Swinging
OS	29.0°	24.4°	21.1°	19.8°	37.9°	19.2°
Madgwick	18.2°	7.5°	7.8°	8.1°	9.4°	10.0°
Mahony	31.8°	26.1°	30.0°	19.9°	13.9°	26.6°
Renaudin	17.1°	7.0°	7.6°	8.9°	8.7°	9.5°
RenaudinExtmag	16.8°	6.4°	7.3°	8.4°	8.4°	8.9°
Sabatini	16.6°	7.0°	8.0°	8.9°	8.6°	10.1°
SabatiniExtmag	14.6°	8.7°	8.9°	6.4°	8.4°	9.0°
MichelObs	32.1°	14.0°	16.4°	14.6°	8.8°	19.1°
MichelObsExtmag	18.0°	11.9°	11.4°	7.4°	8.8°	10.3°
MichelObsExtmagWt	15.5°	9.2°	9.7°	7.1°	7.3°	10.1°
MichelObsF	**10.6°**	**5.4°**	**6.0°**	**5.8°**	**7.1°**	**7.7°**
MichelEkf	16.6°	7.0°	8.0°	8.9°	8.6°	10.1°
MichelEkfExtmag	14.2°	8.9°	9.0°	5.5°	8.6°	9.2°
MichelEkfExtmagWt	12.3°	6.3°	7.2°	5.3°	8.5°	8.7°
MichelEkfF	**10.8°**	**5.3°**	**5.5°**	**5.7°**	**10.3°**	**7.5°**

Table 2.8: Precision of attitude estimation according to typical motions with magnetic perturbations.

with our technique for enforcing minimal durations (See Figure 2.6), the precision is systematically improved when compared to their native variant.

RenaudinExtmag implements a different detector for magnetic perturbations based on variances which improves *Renaudin*. However, *RenaudinExtmag* is very sensitive to false detections because the Earth's magnetic field is known only during the initial phase.

We observe that the two variants of our technique (*MichelEkfF* and *MichelObsF*) gives better precisions for all motions except for the back pocket motion in the case of *MichelEkfF*. *MichelObsF* thus stands out: it provides a significantly better precision during periods of magnetic perturbations even with high accelerations. We also notice that precision is improved regardless of the motion.

Figure 2.7 illustrates the relative improvements in precision brought by the respective components of our new technique presented in Section 2.3, in the case of yaw.

2.4.6 Pitch and Roll in Augmented Reality Scenario

In many applications which use attitude, the goal is to provide the most accurate estimation. In Augmented Reality (AR), for example, it is interesting to use algorithms which provide good estimations of only two of the three Euler angles in order to enhance rendering. We defined pitch and roll as the rotations around y-axis and z-axis. As Euler angles suffer from singularity and this singularity is a problem when the smartphone is held in AR mode we apply a rotation of $90°$

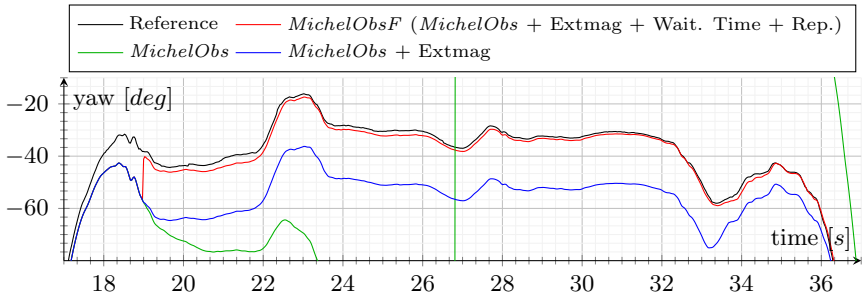
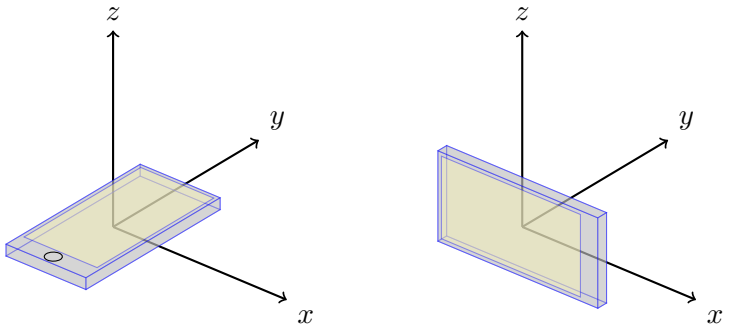


Figure 2.7: Sample run of the reprocessing technique (red) when a magnetic perturbation occurs, in comparison to ground truth (black) and earlier techniques.

around x-axis then another rotation of 90° around z-axis. The smartphone is now considered in "Camera landscape" frame, as shown in Figure 2.8.



(a) Default frame **(b)** Camera landscape frame

Figure 2.8: From default frame to camera landscape frame (rotation of 90°around x-axis then another rotation of 90°around z-axis)

Table 2.9 shows algorithms precision during AR motions in a highly perturbated magnetic environmnent. During motions with low external accelerations, which this is especially the case for AR motions, we can use a specific technique for limiting the impact of magnetic perturbations. We use a cross product between the magnetometer and the accelerometer as yielding our observation vector. This allows algorithms to be more robust to errors from magnetometer mesurements on pitch and roll angles [Martin and Salaün, 2010]. Algorithms using this technique yield better precision than others. For example, *FouratiMartin* is twice accurate than its classical version. The same behavior is observed for *MahonyMartin* algorithm which is 5 times more accurate than the *Mahony* version. It is also possible to equip our filter with this technique in order to enhance overall results (*MahonyMartinF* and *FouratiMartinF*).

It should also be noticed that embedded algorithms have a good behavior in this specific context. It is likely that they use a similar technique.

	QAD	Yaw	**Pitch**	**Roll**
Choukroun	24.8°	22.1°	**6.0**°	**6.6**°
Fourati	32.1°	31.5°	**2.3**°	**3.0**°
FouratiMartin	21.7°	21.3°	**1.4**°	**1.6**°
FouratiMartinF	10.2°	9.8°	**1.4**°	**1.6**°
Madgwick	18.2°	17.1°	**3.2**°	**3.1**°
Mahony	31.8°	28.9°	**6.9**°	**7.9**°
MahonyMartin	14.4°	14.1°	**1.1**°	**1.4**°
MahonyMartinF	10.1°	9.8°	**1.2**°	**1.5**°
Martin	34.4°	34.1°	**0.9**°	**1.2**°
MichelEkfF	10.8°	10.5°	**1.1**°	**1.4**°
MichelObsF	10.7°	10.3°	**1.3**°	**1.6**°
OS	29.0°	28.9°	**1.1**°	**1.2**°
RenaudinExtmag	16.8°	16.0°	**2.6**°	**2.9**°
SabatiniExtmag	14.6°	14.3°	**1.7**°	**1.9**°

Table 2.9: Precision of attitude estimation according to Augmented Reality motions with magnetic perturbations.

2.4.7 Comparison with Device-Embedded Algorithms

Table 2.10 shows algorithms precision depending on the smartphone used. For each algorithm, we observe rather similar results across the different smartphones.

	iPhone 4S	iPhone 5	LG Nexus 5
OS	23.6°	28.6°	12.7°
Choukroun	8.6°	10.4°	10.9°
Mahony	10.8°	15.2°	16.6°
Madgwick	7.1°	8.7°	8.6°
Ekf	6.7°	8.7°	8.5°
MichelObsF	**5.4**°	**6.5**°	**5.9**°
MichelEkfF	**5.6**°	**8.3**°	**7.0**°

Table 2.10: Precision according to device with all motions and with/without magnetic perturbations.

We also observe that all algorithms exhibit a similar or better precision compared to OS-embedded algorithms. We know that this is at least partially due to a bad calibration (especially for iPhones). Finally, we notice that *MichelEkfF* and *MichelObsF* provide much better precision with all smartphones. Specifically, on 126 tests, we noticed that they improve the precision of OS-embedded algorithms on iPhone 4S by 300%, iPhone 5 by 250% and Nexus 5 by 100%.

2.4.8 Empirical Computational Complexity

Because of smartphone's limited resources (e.g. battery), we study to which extent improvements in precision of attitude estimation have an impact in terms of empirical computational complexity. Figure 2.9 summarizes the relative times spent with each algorithm, where unit time corresponds to the running time of *Mahony*. Ratios have been obtained using the offline implementations executed across all 126 datasets.

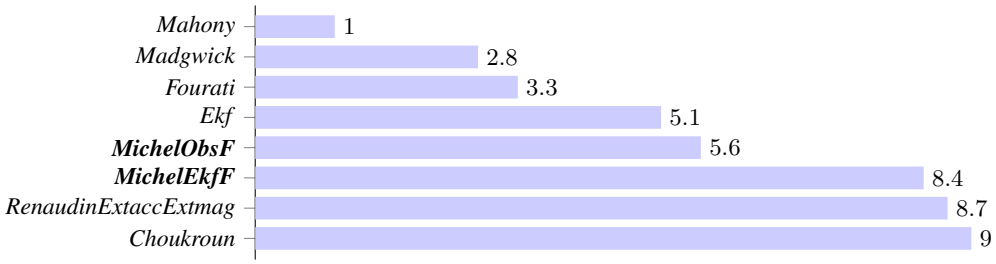


Figure 2.9: Relative performance in terms of CPU cost (lower is better).

We observe that all algorithms can be executed on smartphones even at much higher frequencies than current sensors capabilities (see Table 2.3). For example, our implementation of *Mahony* running on the Nexus 5 can output up to 45000 quaternions per second.

2.4.9 Relevant Sampling Rates

In all aforementioned results, sensors sampling rate was set to 100Hz. We studied the behavior of algorithms whenever the sampling rate varies. Table 2.11 presents precision according to sampling rate. We observe that results with a sampling at 100Hz and 40Hz are relatively similar, and much more precise than with lower frequencies. This suggests to implement filters with a sampling rate of 40Hz to save smartphone's battery life, for a negligible loss in precision.

	100Hz	40Hz	10Hz	2Hz
Choukroun	10.0°	10.1°	15.6°	34.7°
Mahony	14.2°	14.3°	19.7°	48.9°
Madgwick	8.1°	8.1°	17.3°	62.8°
Ekf	8.0°	8.1°	15.3°	49.5°
MichelObsF	**5.9°**	**6.0°**	**14.8°**	**52.5°**
MichelEkfF	**7.0°**	**7.1°**	**14.8°**	**51.3°**

Table 2.11: Precision according to sampling with all motions and with/without magnetic perturbations.

In our specific context, we obtain a mean error of 6° using our best algorithm (*MichelObsF*). When used in an AR application with geolocation and close tracked objects, this might be enough to avoid huge offsets during rendering. This might also be suitable for a navigation application with short trips. For longer trips, the additional use of a map-matching algorithm might be considered.

2.4.10 Parameter Adjustment for a Balance Between Stability and Precision

In the previous section, we evaluated algorithms using parameter values as recommended by their authors (Section 2.2). If authors did not provide instructions on setting parameter values, we chose them empirically. In the present section, we evaluate several sets of parameters for each filter in order to observe their feasibility envelope. For the rest of the study, the precision error of filters is shown in function of the stability.

Stability of a filter

In some specific contexts, eg. AR, the rendering is very important. When the device is static, the augmented point of interests should be static and not moving nor blinking. For this purpose, we added to our benchmark the stability component. The stability is also strongly related to the noise of the sensors and especially the noise of the magnetometer and accelerometer [Michel et al., 2015]. Precision error's STD cannot be used directly to know the stability of the filter. We used a moving STD with a window of $0.1s$ which corresponds to the *moving picture rate* [Card et al., 1986] observable by a user. Obviously, the stability measurement makes sense only when filters assumptions are met (few magnetic perturbations and few external accelerations).

Parameters & Algorithms

Tests have been conducted with different sets of parameter values for each algorithm on a systematic basis. Parameter values have been chosen empirically to cover a spectrum of possibilities and show the trade-off between the stability and the precision error. We recall below the set of parameters of each filter, and for each parameter we give the set of tested values. We consider the cartesian product of all sets of parameter values. We indicate the size of the cartesian product (i.e. the number of configurations tested) next to each filter name.

Choukroun (125)

$$\begin{aligned}\sigma_{acc} &= \begin{bmatrix}0.1 & 0.2 & 0.3 & 0.4 & 0.5\end{bmatrix}\\ \sigma_{mag} &= \begin{bmatrix}0.1 & 0.2 & 0.3 & 0.4 & 0.5\end{bmatrix}\\ \sigma_{gyr} &= \begin{bmatrix}0.3 & 0.4 & 0.5 & 0.6 & 0.7\end{bmatrix}\end{aligned}$$

Ekf (125)

$$\begin{aligned}\sigma_{acc} &= \begin{bmatrix}0.3 & 0.4 & 0.5 & 0.6 & 0.7\end{bmatrix}\\ \sigma_{mag} &= \begin{bmatrix}0.6 & 0.7 & 0.8 & 0.9 & 1.0\end{bmatrix}\\ \sigma_{gyr} &= \begin{bmatrix}0.1 & 0.2 & 0.3 & 0.4 & 0.5\end{bmatrix}\end{aligned}$$

Fourati (125)

$$\begin{aligned}\beta &= \begin{bmatrix}0.1 & 0.2 & 0.3 & 0.4 & 0.5\end{bmatrix}\\ K_a &= \begin{bmatrix}1 & 1.5 & 2 & 2.5 & 3\end{bmatrix}\\ K_m &= \begin{bmatrix}0.5 & 1 & 1.5 & 2 & 2.5\end{bmatrix}\end{aligned}$$

FouratiExtAcc (625)

$$\begin{aligned}\beta &= \begin{bmatrix}0.1 & 0.2 & 0.3 & 0.4 & 0.5\end{bmatrix}\\ K_a &= \begin{bmatrix}1 & 1.5 & 2 & 2.5 & 3\end{bmatrix}\\ K_m &= \begin{bmatrix}0.5 & 1 & 1.5 & 2 & 2.5\end{bmatrix}\\ \gamma_{acc} &= \begin{bmatrix}0.1 & 0.3 & 0.5 & 1 & 3 & 5\end{bmatrix}\end{aligned}$$

$$\beta = \begin{bmatrix}0.1 & 0.2 & 0.3 & 0.4 & 0.5\end{bmatrix}$$

FouratiMartin (125)
$$K_a = \begin{bmatrix}1 & 1.5 & 2 & 2.5 & 3\end{bmatrix}$$
$$K_c = \begin{bmatrix}0.5 & 1 & 1.5 & 2 & 2.5\end{bmatrix}$$

Madgwick (28) $\beta = \begin{bmatrix}0 & 0.05 & 0.01 & 0.02 & ... & 0.24 & 0.25\end{bmatrix}$

$$\beta = \begin{bmatrix}0.5 & 0.75 & 1 & 1.25 & 1.5\end{bmatrix}$$

Mahony (125)
$$K_a = \begin{bmatrix}0.5 & 0.75 & 1 & 1.25 & 1.5\end{bmatrix}$$
$$K_m = \begin{bmatrix}0.5 & 0.75 & 1 & 1.25 & 1.5\end{bmatrix}$$

$$\beta = \begin{bmatrix}0.1 & 0.15 & 0.2 & 0.25 & 0.3\end{bmatrix}$$

MahonyMartin (125)
$$K_a = \begin{bmatrix}0.25 & 0.5 & 0.75 & 1 & 1.25\end{bmatrix}$$
$$K_c = \begin{bmatrix}0.5 & 0.75 & 1 & 1.25 & 1.5\end{bmatrix}$$

MichelObsF (1944)
$$\beta = \begin{bmatrix}0.2 & 0.3 & 0.4\end{bmatrix}$$
$$K_a = \begin{bmatrix}1.5 & 2 & 2.5\end{bmatrix}$$
$$K_m = \begin{bmatrix}0.5 & 1 & 1.5 & 2\end{bmatrix}$$
$$\gamma_{mag} = \begin{bmatrix}12 & 13 & 14 & 15 & 16 & 17\end{bmatrix}$$
$$t_{\text{mag, nopert}} = \begin{bmatrix}1 & 2 & 3\end{bmatrix}$$
$$t_{\text{mag, rep}} = \begin{bmatrix}2 & 3 & 4\end{bmatrix}$$

SabatiniExtMag (486)
$$\sigma_{acc} = \begin{bmatrix}0.4 & 0.5 & 0.6\end{bmatrix}$$
$$\sigma_{mag} = \begin{bmatrix}0.7 & 0.8 & 0.9\end{bmatrix}$$
$$\sigma_{gyr} = \begin{bmatrix}0.2 & 0.3 & 0.4\end{bmatrix}$$
$$\gamma_{mag} = \begin{bmatrix}12 & 13 & 14 & 15 & 16 & 17\end{bmatrix}$$
$$\gamma_{\theta} = \begin{bmatrix}8 & 10 & 12\end{bmatrix}$$

For example, for *MichelObsF* we tested 1944 ways of setting initial parameter values, given by all the possible combinations of the values described above for each parameter.

Augmented Reality: Parameter Adjustment for a Balance Between Stability and Precision

We have set up an online tool[8] to visualize the spectrum of possibilities for each algorithm. Figures 2.10 and 2.11 show the range of possibilities in terms of stability and precision errors for a selection of algorithms during AR trials. Each dot of the graph corresponds to the couple (precision error, stability) for one set of parameter values.

In the case of low magnetic perturbations (Figure 2.10), we observed a common behavior for Kalman filters, whose best results are obtained when $\sigma_{mag} \approx 2\ \sigma_{acc}$ and $\sigma_{acc} \approx 2\ \sigma_{gyr}$. A similar observation holds with the weights of observers (instead of variances – thus with inverted ratios). Ratios found here between the sensors are directly related to sensor noises from the Allan variance [Michel et al., 2015].

[8] http://tyrex.inria.fr/mobile/benchmarks-attitude/#comparison-parameters

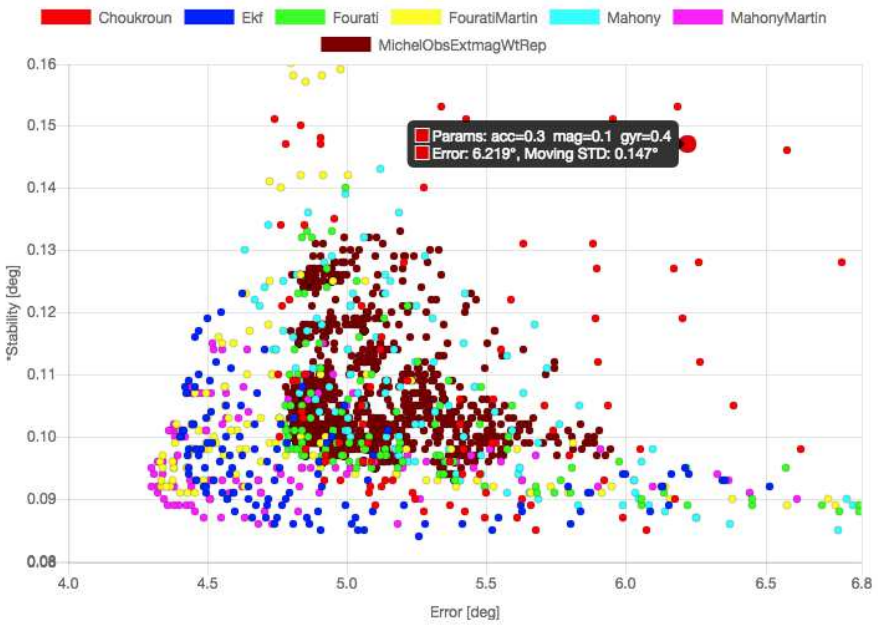


Figure 2.10: Spectrum of possibilities in terms of stability/precision in AR with few magnetic perturbations

In the case of high magnetic perturbations (Figure 2.11), algorithms without detector exhibit a common behavior: their best results are obtained when $\sigma_{acc} < \sigma_{mag}$ and $\sigma_{gyr} \ll \sigma_{mag}$. That behavior shows the impact of magnetic field measurements on the overall results. For algorithms with a magnetic perturbations detector, $\sigma_{gyr} \ll \sigma_{mag}$ is also true, but $\sigma_{acc} \approx 0.75\ \sigma_{mag}$.

To conclude, we observed that some filters provide a better feasibility envelope, especially in the presence of magnetic perturbations (*MichelObsExtmagWtRep*). Also, it is preferable to use a filter which deals with magnetic perturbations, this avoids to create a filter with adaptative parameters in function of the magnetic context.

Moreover, this tool allows us to confirm that parameter values chosen empirically in Section 2.2 are among those that yield the best results in this study.

2.5 Conclusions

In this chapter, we have investigated the use of attitude estimation algorithms in the particular context of pedestrians using commodity smartphones. We have proposed a benchmark for evaluating and comparing the precision of attitude estimations during typical smartphone motions with and without magnetic perturbations. For the first time, our experiments shed light on the relative impacts of calibrations, parameters, noises, bias, motions, magnetic perturbations, and sampling rates when estimating attitude on smartphones. We went further in the study in the particular context of attitude estimation during AR motions. An online tool based on the benchmarks has been released in order to help developers in choosing the right filter and appropriate parameter values in function of the expected motions, device, and magnetic perturbations. In all cases, we recommend developers to use custom calibration and algorithms in replacement of those provided by smart-

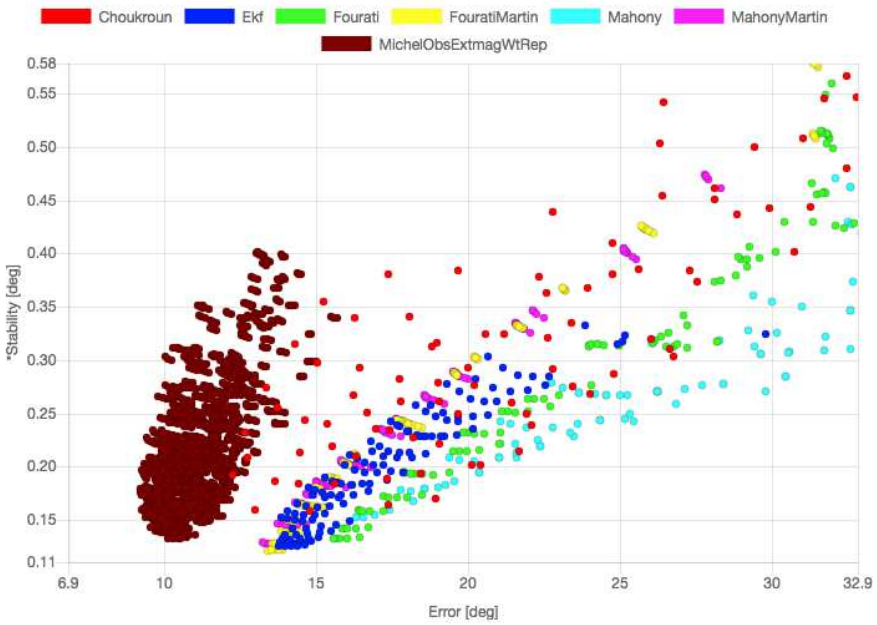


Figure 2.11: Spectrum of possibilities in terms of stability/precision in AR with high magnetic perturbations

phone's OS. Our algorithm "*MichelObsF*" provides significant gains in precision when estimating attitude in the presence of magnetic perturbations. In the absence of magnetic perturbations, it offers the same precision than the most precise algorithms.

Chapter 3

An Experimental Protocol to Evaluate Device Positioning Approches

Contents

3.1 A Set of Applications to Record Sensors Data and Create a Ground Truth . 58
3.1.1 Senslogs - A Core App to Record Data from Built-in Sensors 58
3.1.2 GTR4SL - Ground Truth Recorder for Sensor Localization 61
3.1.3 Fingerprinting Offline App 62
3.2 An Experimental Protocol to Score and Analyze Navigation Algorithms . . 63
3.2.1 A Testbed for Navigation Algorithms with Smartphone 63
3.2.2 Analyze on Vector Maps 64
3.3 Technologies Evaluated 64
3.3.1 WiFi Fingerprinting 65
3.3.2 WiFi Trilateration 65
3.3.3 Step and Heading System 65
3.3.4 SHS + Map-Matching 66
3.3.5 GNSS 66
3.3.6 UWB 66
3.4 Benchmark: A Trade Of Between External Data and Technologies 66
3.4.1 Testbed Context 66
3.4.2 Preprocessing phase 67
3.4.3 Context: Indoor vs Outdoor 68
3.4.4 Smartphone Orientation: Fixed vs Free 69
3.4.5 Starting Position: Known vs Unknown 70
3.4.6 Map: With vs Without 70
3.4.7 WiFi: Sampling Rates and Fingerprints 70
3.4.8 SHS 72
3.5 Conclusions and Perspectives 73

Estimation of device position together with estimation of attitude are the two main components of a Geo AR system. The accuracy of the estimated device position is crucial (see Chapter 4), an error of several meters can produce a poor and unstable AR experience. Users of Geo AR applications can navigate in a wide range of locations. An application which displays names of mountains summits might be assumed to be used in mostly used in a clear space. In 2017, GNSS sensors are present in almost all new manufactured smartphones and are more and more reliable. Horizontal accuracy is less than 1.9 meters 95% of the time [National Coordination Office for Space-Based Positioning, Navigation, and Timing, 2017]. GNSS seems to be an appropriate technology to estimate device location in our Geo AR system. However, some AR applications will not be used in a clear space. For example, an application which helps an engineer to repair a machine located inside a factory cannot use poor signals of GNSS. A more accurate technology needs to be found for this purpose. Some applications even need to manage accurate positions both inside and outside. Unfortunately, there does not exist a "super" technology which works well for all the use cases. But, is the best positioning system accurate enough to provide a good Geo AR experience ?

In order to compare and evaluate technologies from the state of the art, we looked for a benchmark which scores the different systems and ideally find an approach which outperforms others. However, the problem is not simple, there exists lots of benchmarks [Khoury and Kamat, 2009, Curran et al., 2011, Lymberopoulos et al., 2015] but: (i) only few deal with conventional smartphones/tablets, (ii) lots of them are evaluated in unrealistic environments like motion labs or single rooms, (iii) setup is often too much specific, for example, WiFi access points used are more than what we can find in an ordinary building. In addition, these benchmarks do not allow us to use our custom MEMS calibration and to test a novel attitude estimation filter.

The main idea of our work here consists in recording a maximum of sensors data from different devices in several contexts (Section 3.1), then evaluate technologies and approaches on a common dataset (Section 3.2) and finally compare these technologies to conclude on positioning for AR (Section 3.4).

3.1 A Set of Applications to Record Sensors Data and Create a Ground Truth

Most of smartphones and tablets have built-in sensors that measure motion, orientation, and various environmental conditions. Sensors data can be easily accessed via SDKs of main Operating Systems (Android, iOS, Windows) [Google, 2017, Apple, 2017, Windows, 2017]. To benchmark positioning technologies, we designed a set of experimental applications to record data from these sensors and other useful information for navigation algorithms.

3.1.1 Senslogs - A Core App to Record Data from Built-in Sensors

There exists only few applications on Apple Store and Google Play Store which allow us to visualize sensors data freely. Unfortunately, when an application can record all sensors we need for positioning, "store data" action is not free [N Dev Group, 2017, Innoventions, Inc, 2017]. It is for these reasons that we decided to create our own open-sourced library: Senslogs. This

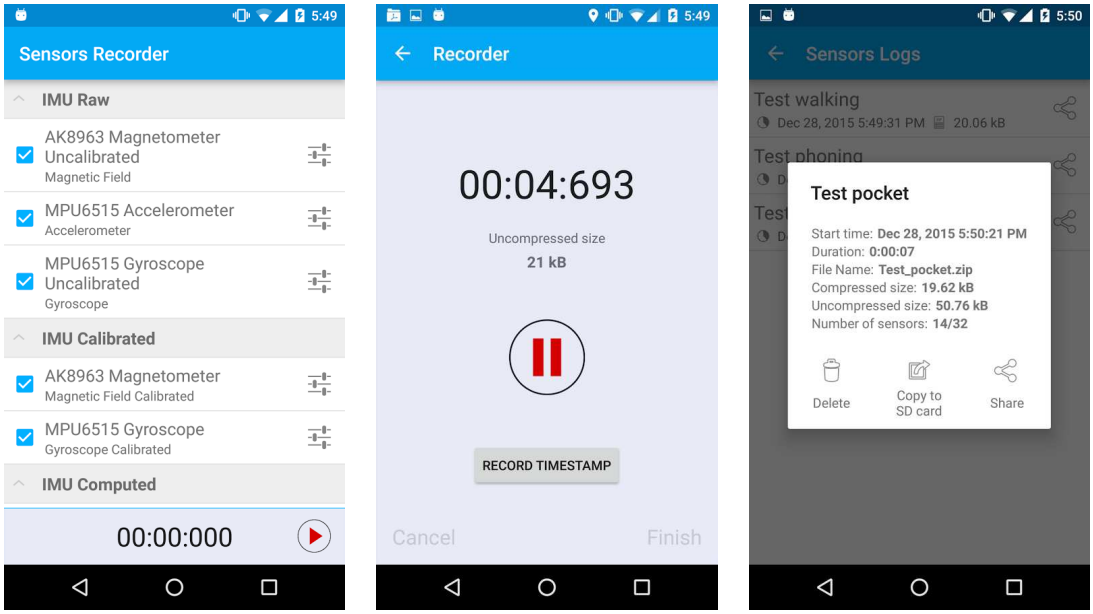


Figure 3.1: Screenshots of Senslogs app

library has been designed to store a maximum of data of a maximum of sensors to be able to score any kind of positioning approach. We focused our work on Android devices[1], but a light application is available for iOS devices[2]. Figure 3.1 shows screenshots at various steps of the recording process within our application. Here is an non-exhaustive list of sensors that we can record with Android SDK: Accelerometer, Gyroscope, Magnetic Field, *Gyroscope Calibrated*, *Magnetic Field Calibrated*, *Game Rotation Vector*, *Geomagnetic Rotation Vector*, *Gravity*, *Linear Acceleration*, *Rotation Vector*, *Significant Motion*, *Step Counter*, *Step Detector*, Ambient Temperature, Light, Pressure, Relative Humidity, Heart Rate, Proximity, *GPS Location*, *Cell and WiFi Location*, *NMEA data*, WiFi signals, Bluetooth signals, NFC, Audio, Video. All sensors in italic text are considered as *computed* sensors because they combine data from raw sensors. In our library, we can record all sensors of the list above but Bluetooth signals, NFC, Audio and Video. We took care to provide all sensors data with a shared clock. Data recorded are stored periodically to the ROM memory to allow huge datasets for long records (like Allan variance). A description of all sensors data handled by Senslogs is provided in Appendix A.2. In Android API, most of sensors can be easily recorded via the SensorManager or the LocationManager, but that is not the case for WiFi signals.

WiFi Recorder

Built-in WiFi sensor and Android SDK are not designed for an accurate pedestrian navigation system. Active scan has been disabled in Android 4.3 and now the minimum sampling rate is relatively high. Implementation of the WiFi sensor API (and more generally for all sensors) is a

[1] `https://github.com/tyrex-team/senslogs`
[2] `https://github.com/tyrex-team/senslogs-ios`

```
timeOfLastScanReceived ⟵ 0; // in seconds
Function main()
    registerBroadcastReceiver(wifiScanReceiver);
    startScan();
    repeat
        if currentTime − timeOfLastScanReceived < 6 then
            startScan();
        end
        sleep(6); // 6 seconds
    until;
end
Function wifiScanReceiver(wifiScan)
    timeOfLastScanReceived ⟵ wifiScan.time;
    storeWifiScan(wifiScan);
    startScan();
end
```

Figure 3.2: Pseudo-code of approach developed to continuously scan WiFi signals.

Device	Avg. sampling rate API	Avg. sampling rate Sensor	Min. sampling rate Sensor	Max. sampling rate Sensor
Nexus 5X	$4.37s$	$4.71s$	$4.28s$	$9.41s$
Nexus 5	$3.03s$	$3.16s$	$2.96s$	$9.07s$
LG G4	$3.79s$	$5.09s$	$3.72s$	$32.86s$
Sony Xperia M2	$0.07s$	$0.07s$	$0.01s$	$3.33s$
Sony Xperia Z Tablet	$2.20s$	$2.50s$	$1.89s$	$28.30s$
Samsung Galaxy E5	$0.03s$	$0.03s$	$0.01s$	$0.44s$
Samsung Galaxy S7	$4.11s$	$4.40s$	$4.07s$	$8.25s$

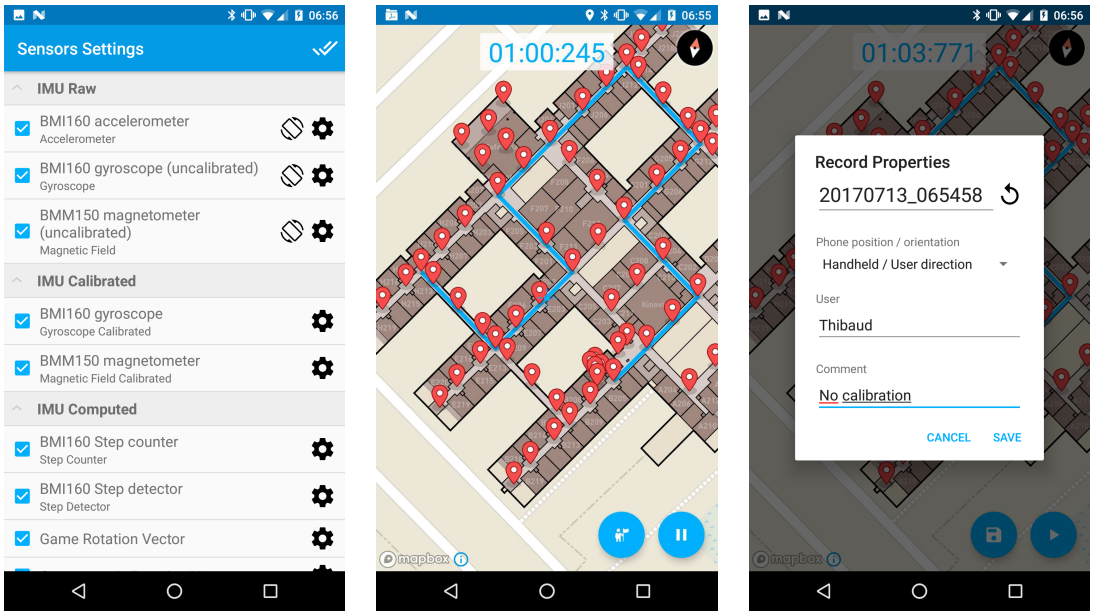
Table 3.1: Statistics on WiFi scan interval

part of manufacturers work, the behavior can differ from one device to another. Sampling rate of WiFi scans is an information that never appears in the documentation provided by manufacturers. We developed and shared an Android application[3] to record and analyze statistics on WiFi sensors embedded in conventional smartphones. For information, in 2017, in iOS SDK, it is still not possible to have access to WiFi scans. We sometimes observed that an incoming WiFi scan from API is equal to the previous one, therefore the API sampling rate is different from the sensor sampling rate. As active WiFi scan has been disabled, the only way to have access to WiFi signals is to register a BroadcastReceiver to be notified when a scan result arrives. Even if the smartphone is connected to a network, Android OS scans WiFi access points continuously (every $30s$-$2min$ depending on the device). At least, a sampling of 30 seconds is not enough for positioning algorithms. This means that if a pedestrian walks straight at 5 $km.h^{-1}$ during 30 seconds, sampling will occur every 42 meters. The method *startScan()* of WiFi API allows developers to force a new sampling procedure asynchronously but without guarantee of result. Figure 3.2 shows the

[3]App: `https://play.google.com/store/apps/details?id=fr.inria.tyrex.wifiscaninterval`
Source code: `https://github.com/ThibaudM/WifiScanInterval`
Results: `http://thibaud-michel.com/mobile/wifi-scan-interval.txt`

approach we used to continuously scan WiFi. Table 3.1 shows some statistics on API sampling rate and WiFi sensor sampling rate for a selection of devices. For each device, average sampling rate of API is close to the average sampling rate of the sensor but the time between two samples varies from $0.03\ s$ to $5.00\ s$ depending on the device. We observed 2 categories: devices with a high sampling rate ($1\ s - 5\ s$) and devices with a low sampling rate ($0.03\ s - 0.1\ s$). In the first category, devices are mostly high-end smartphones and tablets, but the trend seems to move towards an increase of sampling rate, approximatively $5\ s$ for the Galaxy S7 and the Nexus 5X. If sampling rate is at $5\ s$ and user walks at $1.4\ m.s^{-1}$ ($5\ km.h^{-1}$), distance traveled is 7 meters and this corresponds to the error due to the high sampling rate. In the second category, we observed that most of the devices are cheaper smartphones. With these devices, WiFi scan process is almost synchronous, it will be easier to estimate user position.

3.1.2 GTR4SL - Ground Truth Recorder for Sensor Localization



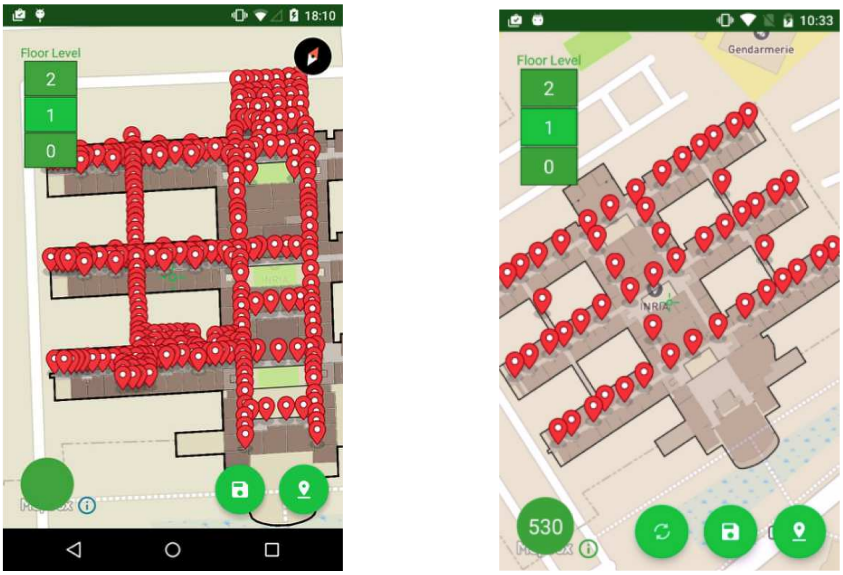
(a) Sensors **(b)** Map **(c)** Properties

Figure 3.3: Screenshots of GTR4SL app

Ground Truth Recorder for Sensor Localization (GTR4SL) is an application we developed with the Senslogs library in order to evaluate navigation algorithms. Setting up a motion capture system in several buildings where some of them are larger than $1000m^2$ is unrealizable. For our approach, we decided to put the user in the loop and ask him to confirm his position when a reference point is crossed. For this purpose, we used our Mapbox-Indoor library (Section 5.3.1) to display outdoor maps and building maps in a same renderer. Thanks to a crosshair at the center of the screen, the user can mark his position. By pushing a button, the user will automatically record a pair $< timestamp, position >$, where $position$ is in WGS84 format and $timestamp$ is on the same clock than the other sensors. As it is tedious to know the exact position of the user when he is moving, we let the possibility to define reference positions with JOSM and import

them as an overlay of the vector map (see Figure 3.3b). During our trials, to help the user in confirming his position at the right place, we marked reference locations with white adhesive tapes. Besides the main functions of Senslogs library, we added shortcuts to quickly record raw data for IMU calibration (see Figure 3.3a). Finally, we added a form at the end of the process to easily categorize datasets (see Figure 3.3c). Categorization is important because some algorithms have specific assumptions: the device might need to be fixed, the user might be required to do a calibration...

3.1.3 Fingerprinting Offline App



(a) A high density fingerprinting

(b) A low density fingerprinting

Figure 3.4: Screenshots of fingerprinting app

A third application has been developed to record sensors data during the offline phase of fingerprinting. This application is based on Senslogs library (see Section 3.1.1) and vector map from Mapbox-Indoor (see Section 5.3.1). To the best of our knowledge, this is the first application which uses a map for fingerprinting and which records data from many sensors in a same process. User selects a location with the crosshair at the middle of the screen then clicks on a button to record sensors data during 15 seconds. Data are stored in a database of pairs $< position, record >$, where $position$ is in WGS84 format and $record$ is the list of sensors data provided by Senslogs during the 15 seconds. With this technique, a $1000m^2$-building with a sampling every 2 meters (250 points) takes about 120 minutes to be recorded.

3.2 An Experimental Protocol to Score and Analyze Navigation Algorithms

In the previous part, we showed how sensors data and reference locations with a smartphone can be recorded. Thanks to the amount of informations we can collect, we are able to compare different navigation algorithms on a same dataset. To the best of our knowledge, this is the first benchmark for indoor and outdoor navigation with conventional smartphones following human typical motions.

3.2.1 A Testbed for Navigation Algorithms with Smartphone

```
Data: sensorsData is a 3 x n ordered queue of data from Senslogs: < timestamp, type, values >
Result: positions is a vector of timestamped positions
algorithm ⟵ new MyNavigationAlgorithm() ;
positions ⟵ new Queue() ;
algorithm.registerPositionListener(PositionListener() {
    Function void newPosition(timestamp, position)
        positions.put(timestamp, position);
    end
});
algorithm.addExternalData(walls, footpaths, accessPoints);
foreach dataRow in sensorsData do
    algorithm.feed(dataRow.timestamp, dataRow.type, dataRow.values);
end
```

Figure 3.5: Pseudo-code of approach developed to continuously scan WiFi signals.

```
void addExternalData(walls, footpaths, accessPoints, fingerprints);
void feed(timestamp, type, values);
void registerPositionListener(PositionListener);
```

where:
walls contains building walls.
footpaths contains footpaths segments.
accessPoints contains positions of WiFi, Bluetooth or other kind of access points.
fingerprints contains information from the Fingerprinting Offline App.
timestamp is the UNIX timestamp of the event.
type is the nature of the sensor (accelerometer, magnetometer, WiFi...).
values are information provided by the sensor.

Figure 3.6: Common interface for navigation algorithms

In order to be as realistic as possible, we chose to simulate the process of a real smartphone by feeding algorithms with data from sensors in the same order than they occurred in the reality. This process is described in Figure 3.5. To formalize the scoring system, all algorithms must share the same interface (see Figure 3.6). Walls and footpaths are segments extracted from OpenStreetMap. Fingerprints come from the Fingerprinting Offline App (see Section 3.1.3). We created our own

ontology based on OSM XML format to store BSSID and positions of WiFi and Bluetooth access points. Finally, algorithms must output an ordered vector of timestamped positions. Now, we have two vectors of positions: one of estimated positions from a custom algorithm and one of the ground truth positions from GTR4SL. Then, vectors are time aligned and we used average (AVG) and standard deviation (STD) of Euclidean distance between both vectors to calculate the score of a dataset. For our implementation, we chose to develop navigation algorithms in pure Java in order to reuse them with both: our MATLAB scoring system and the Android SDK.

3.2.2 Analyze on Vector Maps

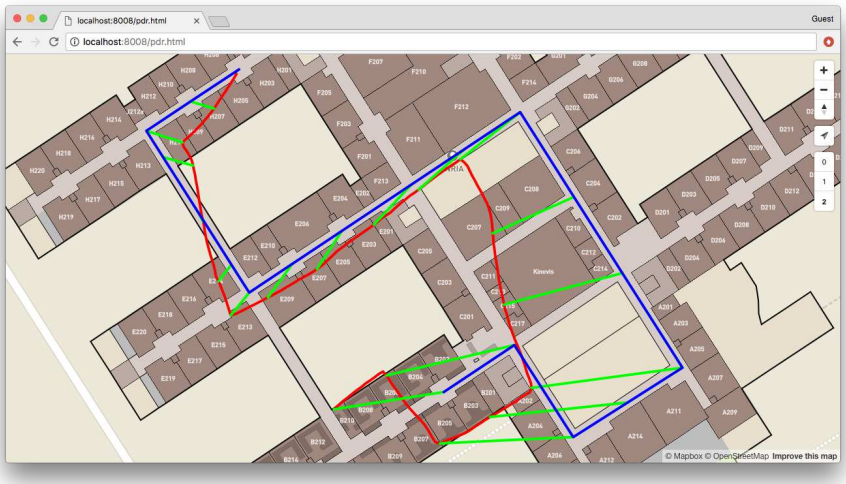


Figure 3.7: Screenshot of our vector map to analyze navigation algorithms. Blue line is the ground truth. Red line is estimated positions. Green lines show distances between estimated and reference positions.

A scoring system is a helpful for classifying navigation algorithms in function of their precision but a score is not enough for explaining all behaviors. With our Mapbox-Indoor library we propose a visualization of estimated positions, ground truth and errors on an indoor and outdoor vector map. An example is given in Figure 3.7. Estimated and ground truth positions are exported in a GeoJSON format, then they are imported thanks to the Mapbox library.

3.3 Technologies Evaluated

There are many different navigation techniques for indoor and outdoor positioning (see Section 1.3.1), and there are even more algorithms which fuse these technologies. To start, we selected an excerpt of techniques that do not require the installation of additional infrastructure, with the notable exception of UWB (which requires installation of anchors). Notice that, WiFi systems rely on WiFi access points but those access points were already and independently deployed (requiring no additional setup for the technique that we consider). We evaluated each technique with

our testbed. The algorithms that we chose to evaluate the various technologies come from the literature and we briefly describe them below.

3.3.1 WiFi Fingerprinting

The first technique we evaluated here is WiFi fingerprinting. The FingerPrints (FP) DataBase (DB) have been filled during an offline phase with our Fingerprinting Offline App (see Section 3.1.3. FP have been recorded at 525 known locations, that is, approximately 4 fingerprints every 100 m^2. Most of them have been recorded in halls and corridors (90%), others have been recorded in small rooms ($<$ 20 m^2). We performed a mean method for each WiFi scan [Bahl and Padmanabhan, 2000, Torres-Sospedra et al., 2014] of a known location to reduce the size of the database. The position of the client is calculated during the online phase. This is done by matching the WiFi Scan (consisting of the received APs and their respective Received Signal Strength Indication - RSSI) of a client with the FP DB. Finally, we used Euclidean distance [He and Chan, 2016] which is the most popular way to calculate the similarity.

3.3.2 WiFi Trilateration

A second technique to determine user position consists in using trilateration with RSSI from WiFi Access Points (AP). Distances between the smartphone and WiFi access points are unknown by the system but thanks to the attenuation of WiFi signal distance can be estimated. There exists specific models for indoor navigation like log-distance path loss model [Rappaport et al., 1996] or ITU model for indoor attenuation [Series, 2012] but all of them need a calibration phase to know the RSSI at a known distance (usually 1 meter). Experiments are under way to take into account wall attenuation and multi-floors thanks to OpenStreeMap information. For our testbed we used the basic FSPL model to avoid the tedious phase of calibration. To apply trilateration algorithm, at least 3 AP signals are required for a 2D localization and at least 4 signals for a 3D localization. To handle signals from a large amount of AP, we used a least square regression to minimize the sum of the distance between user and AP. WGS84 positions of AP cannot be used directly to resolve trilateration algorithm WGS84 is not a Cartesian coordinate system. For this purpose, as WiFi signals affect a small-scale context, a Spherical Mercator projection is applied before the trilateration algorithm.

3.3.3 Step and Heading System

The Step and Heading System (SHS) is a Pedestrian Dead Reckoning (PDR) technique which consists in detecting user's step, estimating step size and user direction. In our specific case, we categorized movements in 2 parts. The first one is *texting* mode, we do not consider misalignment between the smartphone and the navigation frame. The user needs to hold the smartphone with the screen pointing towards the sky and the y-axis pointing forward. In the second category, the position and orientation of the smartphone are free. Most of the people just walked in *swinging* mode. Algorithms used for our benchmark are mainly based on the approach from Ladetto et al. [Ladetto, 2000]. Step detection uses a threshold (2 $m.s^{-2}$) on acceleration of z-axis and a minimum delay between two consecutive steps (0.5 s). The threshold has been lowered to (1 $m.s^{-2}$) for tablets because users naturally hold them with both hands and therefore vertical accelerations are usually lower than those observed with smartphones. The step length is computed using step frequency.

Finally, heading is provided by yaw angle from an attitude filter (see Section 2.4). We compared SHS with 3 different attitude filters: *built-in*, *Fourati* and *MichelObsExtMagWtRep*. Because SHS is a PDR technique, the first user location need to be known (unlike WiFi approaches).

3.3.4 SHS + Map-Matching

Because it is a relative positioning system, the precision error of SHS increases linearly over time. In order to enhance accuracy, we used map information from OpenStreetMap to apply map-matching. The map-matching approach we used in our implementation is *point-to-curve* with condition to match closest segments only if user's heading is close enough to the direction of the segment (see Section 1.3.2). In our setup we used $\theta = 15°$ and $\gamma = 5\ m$.

3.3.5 GNSS

In addition to indoor technologies described above, GNSS has been added to our testbed. Most of our GNSS data recorded come from the GPS. We defined the sampling rate of Android's LocationManager at $1\ s$.

3.3.6 UWB

The last technology introduced here is ultra-wideband (UWB). Ultra-wideband characteristics are well-suited to short-distance applications like indoor positioning. The setup we used for trials came from the BeSpoon company. We used 6 anchors and 1 server. Signal strength information are sent from the 6 anchors to the server which calculates the position of the tag. Smartphones and tablets do not embed UWB tags, but as the processing is made on the server, we fixed a tag on the smartphone and we retrieved the tag position in real time via a web-service. BeSpoon claims a centimetric precision of their navigation algorithms and our testbed is not able to benchmark a so precise system. Therefore, we used Bespoon system only in a subset of our building, in a room of $50\ m^2$. We measured the difference between 20 reference points and estimated positions from the system.

3.4 Benchmark: A Trade Of Between External Data and Technologies

The behavior of above mentioned techniques can be completely different from a context to an other. AR applications can be designed to be used indoor, outdoor or both. Some of them will be used world-wide while others are specific to a building. Therefore, assigning a single global score to each technique is not relevant. In this work, we chose to present precision of navigation algorithms in function of the context, assumptions and knowledge.

3.4.1 Testbed Context

Trials have been conducted in 2 places: a $15\ 000\ m^2$-building with 3 floors and an outside $5\ 000\ m^2$-clear space area (see Fig 3.8). The building is composed of hundreds rooms, dozens of corridors and 1 hall per floor. 30 datasets have been recorded by 5 people. Every person walked

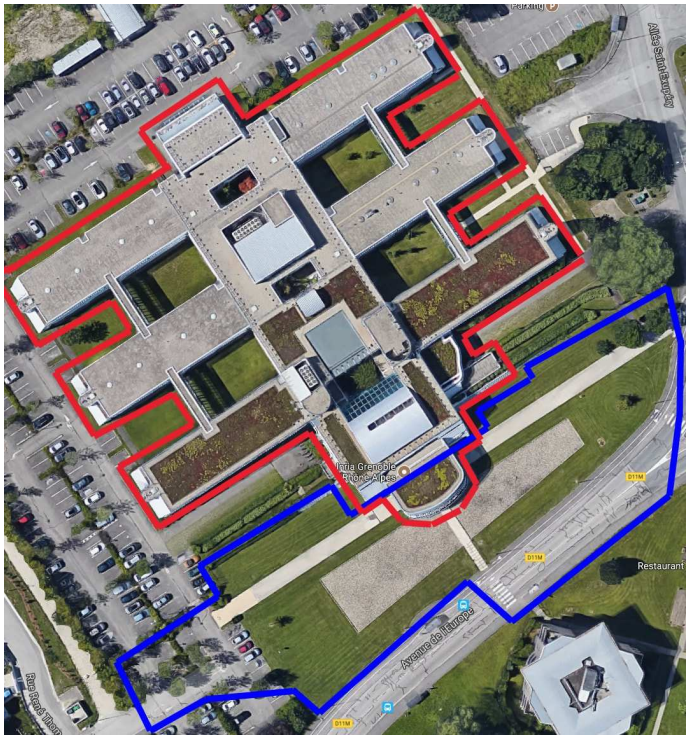


Figure 3.8: Satellite View (Google Maps) of the two places where benchmarks took place. In red the 15 000 m^2-building and in blue the 5 000 m^2-clear space area.

between 3 and 5 minutes. Each person was always accompanied by a second person to verify if reference positions were confirmed at the right place. Foot paths for map-matching were drawn at the center of corridors. 3 smartphones (LG Nexus 4, LG Nexus 5, LG Nexus 5X) and 1 Tablet (Sony Xperia Z Tablet) have been used. The precision errors reported in this section are average errors in meters from our scoring library described in Section 3.2.1.

3.4.2 Preprocessing phase

Unfortunately, except GNSS, all navigations systems need an offline phase. It consists in recording specific data or calibrate sensors. According to the technology used, this offline phase can take time. In Table 3.2, we compared the time we spent for each technique from our own experience and more details on the process are given below.

Map A vector map of the building has been designed for the purpose. The conversion from an AutoCAD map from architects to OSM format is not fully automatic. To create a map of our 15 000 m^2-building with all corridors and rooms it took us approximately 12 hours. A light version just with corridors, took us $1hour$. This map have been reused for most of the technologies evaluated and for rendering (see Section 5.3.1).

SHS As all PDR systems, SHS needs to start with a known position. It is tedious to know the absolute position of a point in an indoor environment. One possibility is to use specific equipments like a laser rangefinder coupled with a DGPS but it takes time. The second

Technology	Time Spent
SHS	Detailed map ($\sim$ 12 hours)
SHS + Map-Matching	Few minutes + Detailed map ($\sim$ 12 hours)
WiFi-Fingerprinting	5$\sim$6 hours + Detailed map ($\sim$ 12 hours)
WiFi-Trilateration	1 hour + Light map (1 hour)
UWB	2$\sim$3 hours (but for a room of 50 m^2)
GNSS	0

Table 3.2: Average time spent for the offline phase with each navigation technology in our 15 000 m^2-building.

possibility, the one we chose in this work, consists in selecting the starting position on the detailed map of the building.

SHS + Map Matching The point-to-curve map-matching technique for SHS use building corridors and outdoor footpaths. Building corridors are represented by segments in the center of corridors. It took us only few minutes to add these segments to our map. Footpaths were extracted from OpenStreetMap data.

WiFi - Fingerprinting Our fingerprinting offline map application (see Section 3.1.3) uses the building map to help user to record positions. Without this map user is not able to provide a WiFi database. Then, on the fingerprinting offline map application, we spent approximatively 5 hours to record fingerprints at the 525 known locations. Most of the people uses robots to do the job but the setup is often more longer than our process.

WiFi - Trilateration The goal here is to find absolute positions of the WiFi access points. A non-detailed map of the building ($\sim$ 1 hour) is enough to place access points at the right position.

UWB In the 50 m^2-room, we tried to find a quasi-optimal setup to cover as much as possible the volume of the room. Auto-calibration is not adapted when the number of anchor points is low (we had 6), consequently we measured their exact positions with a laser rangefinder, this took us approximately 2 hours.

GNSS The position provided by the trilateration from GNSS does not need any extra information. The first position fix is not considered here because it is automatic.

3.4.3 Context: Indoor vs Outdoor

We tested navigation algorithms in two different contexts: inside a large-building and outside in clear space area. Table 3.3 presents the results when the smartphone is held in the same orientation than the navigation frame. Clearly, indoor, UWB outperforms others technologies. Unfortunately, this is the only one technique we compared which is not natively implemented in smartphones. Among others, SHS + Map-Matching exhibits a good behavior with an average of 2.28 m, the gain compared to SHS without Map-Matching is 240%. Outdoor, GNSS has a mean

	Indoor		Outdoor	
	AVG	STD	AVG	STD
SHS	8.18 m	4.96 m	16.68 m	14.44 m
SHS + Map-Matching	2.26 m	1.55 m	11.93 m	9.60 m
WiFi-Fingerprinting	8.12 m	8.56 m	x*	x
WiFi-Trilateration	7.72 m	8.32 m	x*	x
UWB	0.49 m	0.26 m	x*	x
GNSS	25.44 m	14.76 m	3.54 m	2.58 m

* Technologies based on WiFi and UWB have not been deployed outside.

Table 3.3: Average (AVG) and Standard Deviation (STD) of precision error of navigation algorithms inside and outside.

error of 3.29 m and outperforms SHS techniques. Moreover, GNSS does not need any preprocessing phase nor the knowledge of the starting position. In the rest of the benchmark we now consider only datasets recorded inside the building.

3.4.4 Smartphone Orientation: Fixed vs Free

	Fixed		Free	
	AVG	STD	AVG	STD
SHS	8.18 m	4.96 m	17.20 m	12.77 m
SHS + Map-Matching	2.26 m	1.55 m	15.16 m	14.33 m
WiFi-Fingerprinting	8.12 m	8.56 m	9.61 m	11.89 m
WiFi-Trilateration	7.72 m	8.32 m	10.18 m	12.41 m
UWB	0.49 m	0.26 m	0.49 m	0.26 m
GNSS	25.44 m	14.76 m	25.93 m	16.22 m

Table 3.4: Comparison of precision error of navigation algorithms in an indoor environment when device is fixed or free.

In the previous comparison, algorithms have been evaluated in the specific case where the device is held in the same direction than the user direction. We called it "fixed". In reality, when the user needs to be localized, he can hold the smartphone in any orientation. We called it "free". All algorithms based on a SHS suffer from this, because navigation frame is not aligned with the smartphone frame. Approaches to overcome this problem are explained in [Combettes and Renaudin, 2015] but have not been implemented yet. Table 3.4 shows results for fixed and free devices. Therefore, if we do not consider UWB due to the heavy equipment, the best technique which is not impacted by the misalignment is WiFi-Fingerprinting with an average error of 9.61 m.

	Unknown Starting Position	
	AVG	STD
SHS	x	x
SHS + Map-Matching	x	x
WiFi-Fingerprinting	$8.12\ m$	$8.56\ m$
WiFi-Trilateration	$7.72\ m$	$8.32\ m$
UWB	$0.49\ m$	$0.26\ m$
GNSS	$25.44\ m$	$14.76\ m$

Table 3.5: Precision error of navigation algorithms without the knowledge of starting position with a fixed smartphone in an indoor context.

3.4.5 Starting Position: Known vs Unknown

We can categorize navigation techniques in two groups: radio signal based (WiFi, UWB, GNSS, Bluetooth) and dead reckoning (SHS, INS). Radio signal based algorithms work with fingerprinting or trilateration, the starting position of the user is not required. That is not the case for dead reckoning approaches. Without this information, SHS and other dead reckoning systems cannot provide a position. Table 3.5 presents the results when the smartphone is held in the same orientation than the navigation frame. In this case, when a huge area needs to be covered, WiFi-Trilateration is the most precise technique with an average error of $7.72\ m$.

3.4.6 Map: With vs Without

Among the algorithms we benchmarked here, only the approach SHS + Map-Matching uses information from the map during navigation. When the device is fixed, the point-to-curve technique reduces the precision error of 240% from $7.76\ m$ to $2.28\ m$. We showed in Section 3.4.2 that all approaches (except GNSS) need to use a building map during the pre-processing phase. As the time spent to draw corridors is negligible compared to the time spent to provide a detailed building map, it seems interesting to always use an approach with map-matching.

3.4.7 WiFi: Sampling Rates and Fingerprints

In the previous sections, we compared some navigation techniques. Each technique can be implemented in different ways and consequently the precision error is impacted. This is the case for WiFi technologies where we want to highlight some points.

First we studied the WiFi-Fingerprinting approach and especially the impact of the offline phase. Our offline phase was recorded with a LG Nexus 4 because the sampling rate was fast (~ 1 sample every $0.7\ s$). Nevertheless, our online phase was benchmarked with the Nexus 4 and 3 others devices. Precision error of WiFi-Fingerprinting and WiFi-Trilateration in function of the device are shown in Table 3.6. Data from this subset have been recorded in the way to benchmark only the positioning accuracy and not the navigation accuracy: we waited $10\ s$ at every reference position to be sure the smartphone scanned WiFi signals from this exact position. We observed that the smartphone we used during the offline phase (Nexus 4) shows a better precision than

	WiFi - Fingerprinting		WiFi - Trilateration	
	AVG	STD	AVG	STD
Nexus 4	1.29 m	1.53 m	3.91 m	1.77 m
Nexus 5	4.97 m	5.33 m	4.57 m	1.82 m
Nexus 5X	2.24 m	2.60 m	2.99 m	1.60 m
Xperia Z Tablet	2.02 m	2.62 m	3.38 m	2.15 m

Table 3.6: Precision error of WiFi approaches with different static devices in an indoor context.

others during fingerprinting with an accuracy close to 1 meter. This observation is not true with trilateration technique where all devices exhibit a similar precision of $3 \sim 4$ meters. That is a normal behavior because trilateration do not rely on a specific device.

	Sampling Rate	WiFi - Fingerprinting		WiFi - Trilateration	
		AVG	STD	AVG	STD
Nexus 4	0.72 s	2.72 m	2.43 m	5.90 m	4.76 m
Nexus 5	3.18 s	14.53 m	16.63 m	17.21 m	17.04 m
Nexus 5X	4.96 s	7.87 m	6.23 m	6.14 m	4.63 m
Xperia Z Tablet	2.50 s	7.80 m	5.82 m	6.34 m	4.49 m

Table 3.7: Precision error of WiFi approaches with different moving devices in an indoor context.

In a second study, we benchmarked the impact of WiFi sampling rate (introduced in 3.1.1) on final precision. During these trials, we walked continuously, so there was no guarantee to have a sampling on the exact position of the reference point due to the low sampling rate. Results of this benchmark are shown in Table 3.7. Nexus 4, with a "low" sampling rate of 0.72 s exhibits the best accuracy. It is even more striking when looking at fingerprinting because the same smartphone has been used during offline phase.

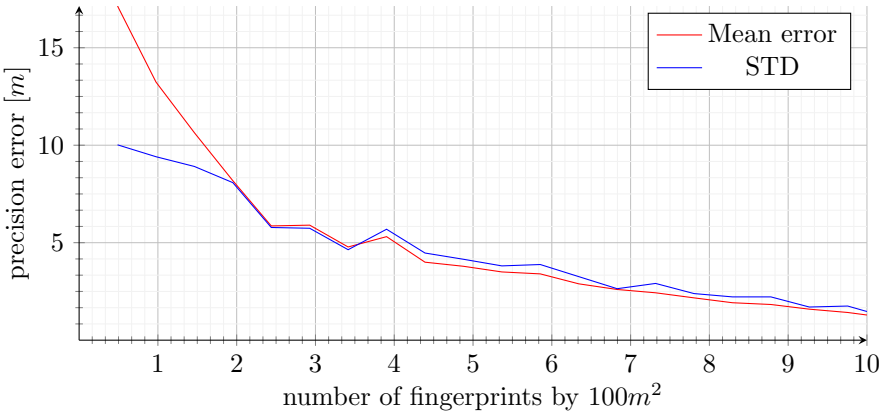


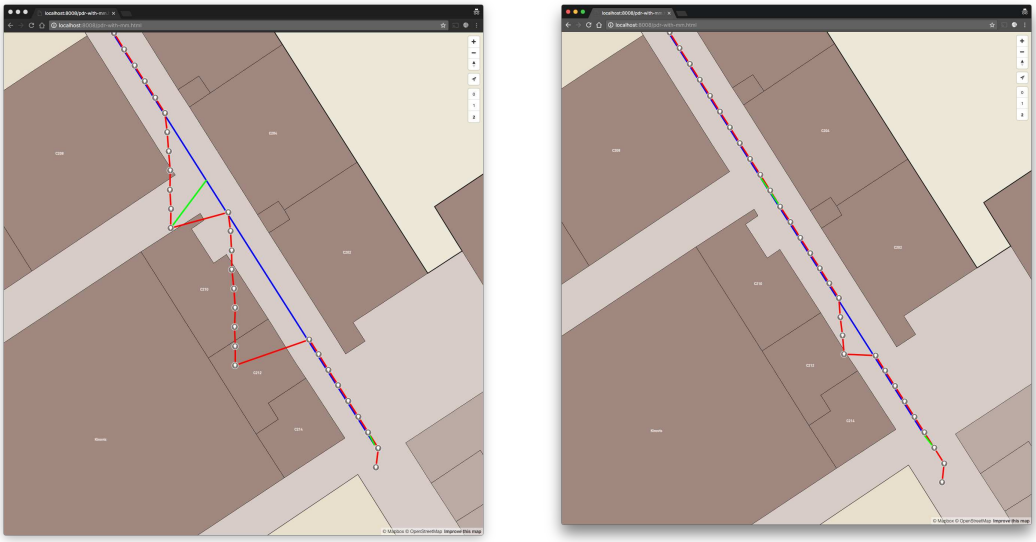
Figure 3.9: Precision error of WiFi-Fingerprinting approach in function of the number of fingerprints recorded.

Finally, we went further to understand the impact of the number of considered WiFi fingerprints on the final precision. For this purpose, we reused same datasets where we waited 10 s at each reference position with the Nexus 4. In these datasets, we used about 10 fingerprints by 100 m^2. We deliberately removed fingerprints randomly and we displayed results on Figure 3.9. We observed a high correlation between the number of fingerprints recorded and the precision error. The greater the number of fingerprints, the more accurate are the estimations of the algorithm. We did not observe a convergence value in our dataset before 10 fingerprints by $100m^2$. Therefore, it is questionable to spend a lot of time during the offline phase to be more accurate in the online phase.

3.4.8 SHS

	Without map-matching		With map-matching	
	AVG	STD	AVG	STD
Built-in	55.21 m	68.19 m	49.41 m	70.58 m
Fourati	8.62 m	4.70 m	2.33 m	1.55 m
MichelObsExtMagWtRep	8.18 m	4.96 m	2.26 m	1.55 m

Table 3.8: Precision error of SHS approaches with different attitude algorithms when smartphone is fixed in an indoor context.



(a) *Fourati* **(b)** *MichelObsExtMagWtRep*

Figure 3.10: The eight typical motions for a smartphone.

In the same way than for WiFi technologies, we evaluated different implementations of SHS technologies and essentially with different attitude estimation filters. In Section 2.4, we presented an evaluation of attitude estimation algorithms when a person is walking. Among existing filters in

the literature, we observed a better behavior of *Fourati* algorithm during *texting* motion and when external accelerations are higher (see Table 2.7). As a reminder, *MichelObsExtMagWtRep* (Section 2.3) is an attitude filter built over *Fourati* which deals with magnetic perturbations typically found in buildings. In this study, we compared these 2 filters and the built-in filter from Android API with SHS approaches. Results are shown in Table 3.8. As we described in Section 2.4.7, the built-in filter from Android is not reliable. When we went in details on each dataset, we observed that some of them exhibit a similar error compared to *Fourati* and *MichelObsExtMagWtRep*, but for others the behavior is totally different and precision is highly impacted. This behavior seems to be related to a problem from a bad magnetometer calibration, which is automatic and not controllable in the Android system. *MichelObsExtMagWtRep* exhibits a better precision than its native variant (*Fourati*) when map-matching is not used. The difference between both is less noticeable with map-matching due to the threshold on the angle compensation (see Eq 1.22). An example of this behavior of *MichelObsExtMagWtRep* against *Fourati* is shown in Figure 3.10.

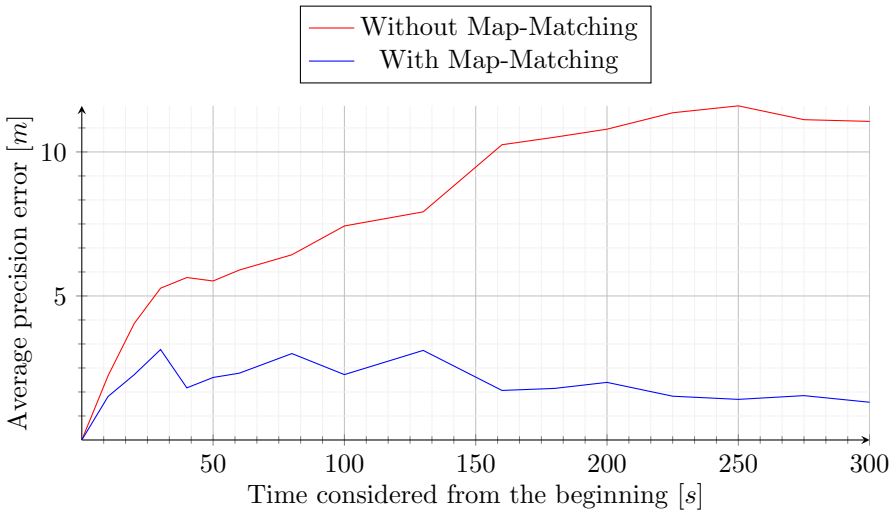


Figure 3.11: Precision error of SHS approaches in function of the time considered from the beginning, when smartphone is fixed in an indoor context.

Finally, as SHS is a dead reckoning approach and suffers from divergence, we evaluated the precision error over time. Results are shown in Figure 3.11. Error from SHS without Map-Matching is constantly increasing. From $0s$ to $30s$, we observed a linear progression of 15 $cm.s^{-1}$. This behavior is mainly due to users who did trials because they tend to go far than the starting position during the first seconds. The progression is less noticeable with map-matching due to the threshold on angle compensation (see Eq 1.22). About SHS with map-matching, the precision error we obtained through our benchmark tend to converge to 2 m. It shows that our point-to-curve algorithm can be lost very locally (only during few seconds) but is never totally lost (diverging).

3.5 Conclusions and Perspectives

We have investigated the use of positioning estimation algorithms in the particular context of pedestrians using commodity smartphones. We have proposed a tool to create experimental pro-

tocols for evaluating and comparing the precision of navigation algorithms during typical smartphone motions. We have used this tool to benchmark several techniques: WiFi Fingerprinting, WiFi Trilateration, SHS, SHS + Map-matching, GNSS and UWB. The results of our evaluation have shown that there is no single technique which outperforms all the others for all use cases. However, some techniques are particularly appropriate with a specific context (outdoor, not fixed, with map...).

For the moment, our benchmark has been setup in one building only. As our application GTR4SL is not specific to our building, it would be interesting to enrich our benchmark with other buildings as well. On these new setups, we plan to add Bluetooth tags to study the differences with WiFi technologies. Moreover, we would like to evaluate more techniques and especially algorithms which merge several approaches. A list of interesting perspectives for further work is given below:

- Use algorithms with particle filters as in our testbed we already have access to the vector map of walls from OpenStreetMap.
- Take into account wall attenuation for WiFi-Trilateration for the same reason than the previous point.
- Use misalignment algorithms to avoid user to hold the smartphone in the same direction than the navigation frame.

Chapter 4

An Evaluation Method for Augmented Reality

Contents

4.1 Introduction 76
4.2 An Evaluation Method to Calculate Distance and Angle Errors 77
4.2.1 Attitude Estimation Model 77
4.2.2 Position Estimation Model 78
4.2.3 Attitude + Position Estimation Model 79
4.3 Projected Distance on the Screen 80
4.4 Scores from our Benchmarks: Examples on Different Use Cases 81
4.4.1 Use Case 1: An Application to Identify Mountains and Cities 82
4.4.2 Use Case 2: An Application to Discover the History of a City 83
4.4.3 Use Case 3: Make 3D Models Appear and Turn Around in an Indoor Environment 84
4.4.4 Use Case 4: Identify and Interact with Objects in a Building using UWB 84
4.5 Conclusion 85

In Chapters 2 and 3, we have compared several algorithms which provide estimations for attitude and position of the device. We did it because both represent the core of the Geo AR approach. But, what is the impact of a poor attitude estimation or a poor positioning estimation on the rendering of an AR browser? Is it possible to quantify them? Errors from positioning estimation and errors from attitude estimation are not of the same nature, it is not possible to directly combine them and provide an average error. Moreover, the notion of distance between a feature and the device is very important, because impact can be totally different depending on whether a feature is close or far. For example, we consider a positioning system with an accuracy of 10 m and an attitude estimation algorithm which exhibits an error of 2°. If the user aims a mountain at 10 km in front of him; on the smartphone screen, distance between mountain from the video feed and mountain from the virtual scene is very small. Whereas, if the user is aiming a bus stop at 20 m ahead of him, it is very likely that this virtual bus stop will not be displayed on the screen. This is mainly due to the huge inaccuracy of the positioning system.

4.1 Introduction

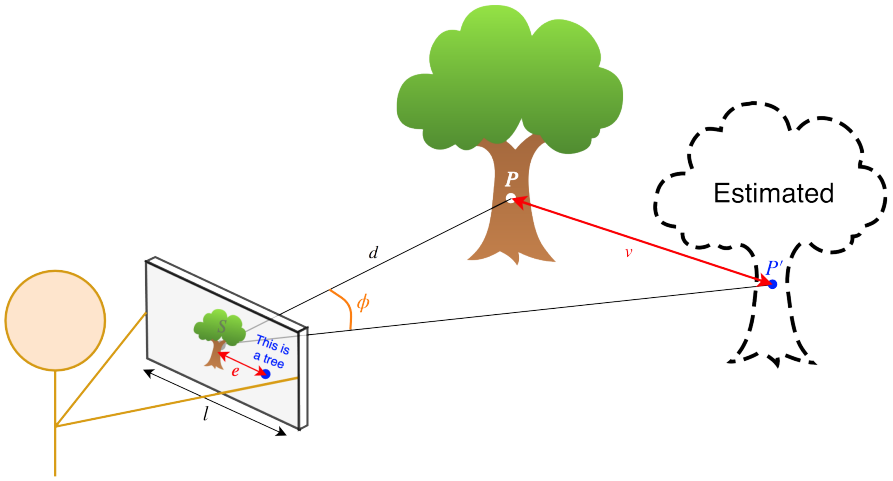


Figure 4.1: Representation of errors due to a poor estimation of a virtual feature position. v is the distance between the estimated feature (P') and the real position of the feature (P). The projection of this distance on the screen is named e.

We propose an evaluation model to calculate the average distance between a real and a virtual point represented on the screen given the 2 vectors of errors (F_{pos} and F_{att}):

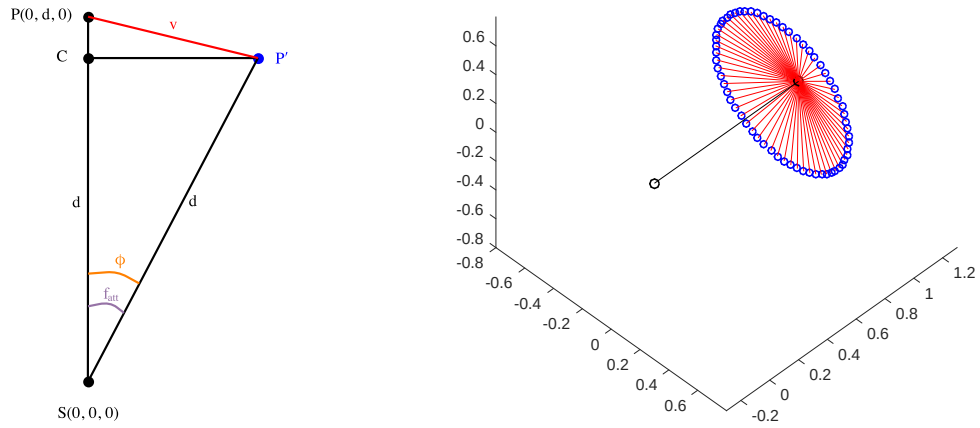
- f_{pos} which denotes positioning estimation errors in meters given by a navigation algorithm (see Chapter 3). F_{pos} is the vector of f_{pos} values for a specific use case (e.g: outdoor using GNSS).
- f_{att} which denotes the angle describing the magnitude of the rotation due to a poor attitude estimation. This angle corresponds to the angle from *axis-angle* representation of a rotation. It is directly related to Quaternion Angle Difference (QAD) obtained in Chapter 2 for benchmarks on the attitude filters. F_{att} is the vector of f_{att} values for a specific use case (e.g: *MichelObs* filter with high magnetic perturbations).

We consider a feature point (P) at a fixed distance (d) (see Figure 4.1). The feature (here: a tree) is displayed in the middle of the smartphone (S) screen. The system estimates the position of the feature (P') from attitude and position of the device. v is the distance between the estimated feature (P') and the real position of the feature (P). The projection of this distance on the screen is named e. This distance depends on screen size l and the field of view (fov) of the rendering (for the sake of clarity fov is not rendered on the figure). And finally, ϕ is the positive angle between $\vec{SP}$ and $\vec{SP'}$. The process explained here is the same as OpenGL rendering with our Rajawali scene in Section 5.2.1. In [Panel, 2014, Ichikari et al., 2017, ISO 18520, 2017], to evaluate visual SLAM techinques for AR, some metrics are equivalent to ours. v is equivalent to 3DEVO (3D Error of Virtual Object), and e is equivalent to PEVO (Projection Error of Virtual Object).

4.2 An Evaluation Method to Calculate Distance and Angle Errors

We start by reducing the problem to look for parameters v and ϕ because they are independent to the screen size and the field of view.

4.2.1 Attitude Estimation Model



(a) α fixed **(b)** α varying

Figure 4.2: Evaluation model and simulation where $f_{\text{pos}} = 0$ and f_{att} is fixed

To begin with, we assume the estimated position is perfect ($f_{\text{pos}} = 0$). We consider f_{att}, the angle between $\vec{SP}$ and $\vec{SP'}$ due to a poor attitude estimation. To determine $v_{f_{\text{pos}}=0}$ and $\phi_{f_{\text{pos}}=0}$, we propose the following model:

$$\begin{cases} S = (0,0,0) \\ P = (0,d,0) \\ \widehat{PSP'} = f_{\text{att}} \\ \left\|SP'\right\| = d \end{cases} \tag{4.1}$$

We consider α, the angle between $P'_{f_{\text{pos}}=0}$, $C_{f_{\text{pos}}=0}$ and the x-y plan where $C_{f_{\text{pos}}=0}$ is the projection of P' on $[SP]$. With the modeling introduced above we determine the position of $C_{f_{\text{pos}}=0}$ and $P'_{f_{\text{pos}}=0}$:

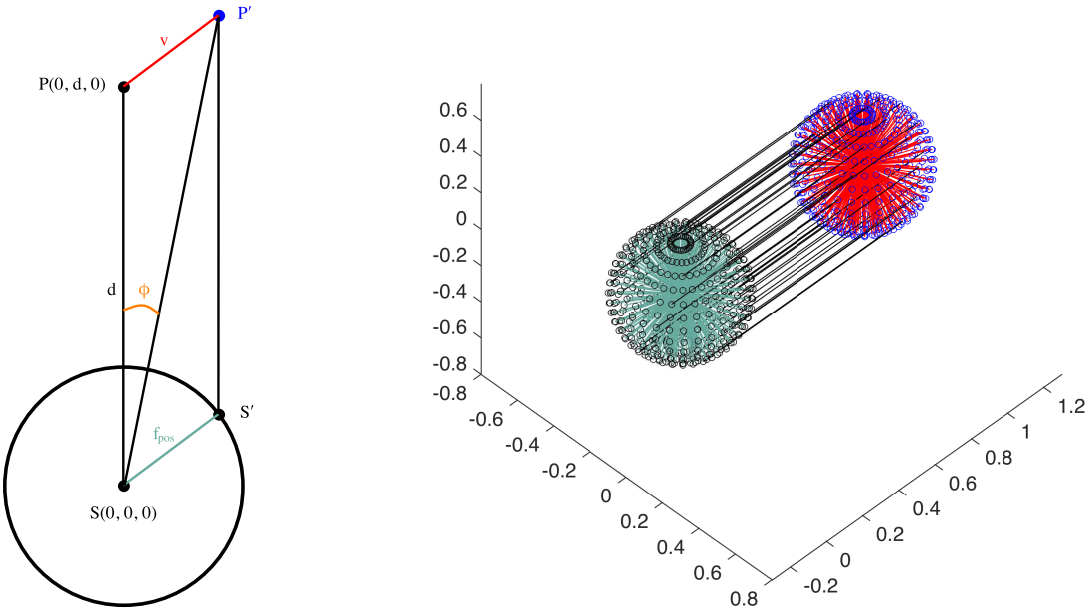
$$\begin{aligned} C_{f_{\text{pos}}=0}(d, f_{\text{att}}) &= (0, d * cos(f_{\text{att}}), 0) \\ P'_{f_{\text{pos}}=0}(d, f_{\text{att}}, \alpha) &= C_{f_{\text{pos}}=0} + (d * sin(f_{\text{att}}) * cos(\alpha), 0, d * sin(f_{\text{att}}) * sin(\alpha)) \end{aligned} \tag{4.2}$$

Then we determine the distance $v_{f_{\text{pos}}=0}$ and the angle $\phi_{f_{\text{pos}}=0}$:

$$\begin{aligned} \phi_{f_{\text{pos}}=0}(d, f_{\text{att}}) &= f_{\text{att}} \\ v_{f_{\text{pos}}=0}(d, f_{\text{att}}, \alpha) &= \sqrt{(d * sin(f_{\text{att}}) * cos(\alpha))^2 + (d * cos(f_{\text{att}}) - d)^2 + (d * sin(f_{\text{att}}) * sin(\alpha))^2} \\ &= \sqrt{2 * d^2(1 - cos(f_{\text{att}}))} \end{aligned} \tag{4.3}$$

The representation of the model and a simulation are given in Figure 4.2.

4.2.2 Position Estimation Model



(a) β and γ fixed **(b)** β and γ varying

Figure 4.3: Evaluation model and simulation where $f_{\text{att}} = 0$ and f_{pos} is fixed

In a second step, we assume the estimated attitude is perfect ($f_{\text{att}} = 0$). We consider f_{pos}, the distance between the estimated position of the device ($S'_{f_{\text{att}}=0}$) and its real position (S).

To determine $v_{f_{\text{att}}=0}$ and $\phi_{f_{\text{att}}=0}$, we propose the following model:

$$\begin{cases} S = (0,0,0) \\ P = (0,d,0) \\ \left\|SS'\right\| = f_{\text{pos}} \\ \vec{S'P'} = \vec{SP} \end{cases} \tag{4.4}$$

$S'_{f_{\text{att}}=0}$ belongs to the sphere centered in S with a radius of f_{pos}. We can characterize S' with two parameters: γ and β. Let $S''_{f_{\text{att}}=0}$ be the projection of $S'_{f_{\text{att}}=0}$ in the x-y plan, γ is the angle between SS'' and the x-axis and β is the angle between $S''_{f_{\text{att}}=0}$, S and $S'_{f_{\text{att}}=0}$. With this modeling we have:

$$\begin{aligned} S'_{f_{\text{att}}=0}(d, f_{\text{pos}}, \beta, \gamma) &= (f_{\text{pos}} * cos(\beta) * cos(\gamma), f_{\text{pos}} * cos(\beta) * sin(\gamma), f_{\text{pos}} * sin(\beta)) \\ P'_{f_{\text{att}}=0}(d, f_{\text{pos}}, \beta, \gamma) &= S' + (0,d,0) \end{aligned} \tag{4.5}$$

Then we determine the distance $v_{f_{\text{att}}=0}$ and the angle $\phi_{f_{\text{att}}=0}$:

$$\begin{aligned} (\vec{P} - \vec{S}) \cdot (\vec{P'} - \vec{S}) &= \left\|\vec{P} - \vec{S}\right\| * \left\|\vec{P'} - \vec{S}\right\| * cos(\phi) \\ \vec{P} \cdot \vec{P'} &= d * \left\|\vec{P'}\right\| * cos(\phi) \end{aligned} \tag{4.6}$$

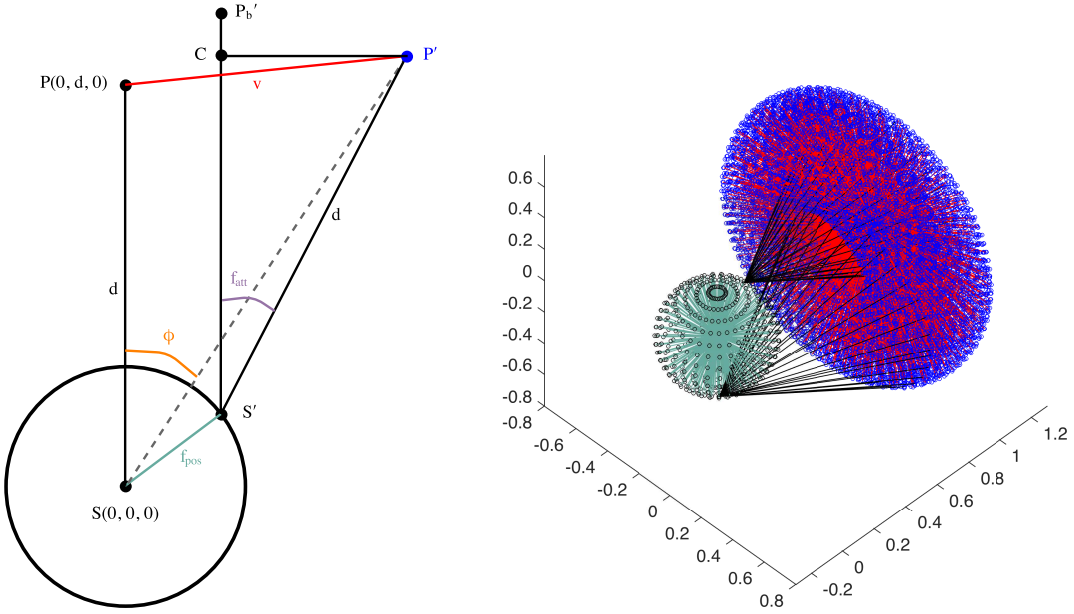
$$\begin{aligned}\phi_{f_{\text{att}}=0}(d, f_{\text{pos}}, \beta, \gamma) &= acos(\frac{\vec{P} \cdot \vec{P'}}{d * \left\|\vec{P'}\right\|}) = acos(\frac{P'_y}{\left\|\vec{P'}\right\|}) \\ v_{f_{\text{att}}=0}(d, f_{\text{pos}}, \beta, \gamma) &= f_{\text{pos}}\end{aligned} \tag{4.7}$$

The representation of the model and a simulation are given in Figure 4.3.

Here, we considered f_{pos} as an error on the 3 dimensions. But, the model can be modified to handle distributions on 2D or 2.5D if vertical (f^v_{pos}) and horizontal (f^h_{pos}) errors are known:

$$\begin{aligned}S'_{2D}(f^h_{\text{pos}}) &= (f^h_{\text{pos}} * cos(\gamma), f^h_{\text{pos}} * sin(\gamma), 0) \\ S'_{2.5D}(f^h_{\text{pos}}, f^v_{\text{pos}}) &= (f^h_{\text{pos}} * cos(\gamma), f^h_{\text{pos}} * sin(\gamma), f^v_{\text{pos}} * z) \qquad \text{where: } z \in \{-1, 1\}\end{aligned} \tag{4.8}$$

4.2.3 Attitude + Position Estimation Model



(a) α, β and γ fixed

(b) α, β and γ varying

Figure 4.4: Evaluation model where f_{att} and f_{pos} are fixed

Now, we consider the estimated attitude and position not perfect, we combine both models. To determine v and ϕ, we propose the following model:

$$\begin{cases} S = (0, 0, 0) \\ P = (0, d, 0) \\ \left\|SS'\right\| = f_{\text{pos}} \\ \widehat{PSP'} = f_{\text{att}} \\ \left\|\vec{S'P'}\right\| = d \end{cases} \tag{4.9}$$

Firstly, we determine positions of S', C and P':

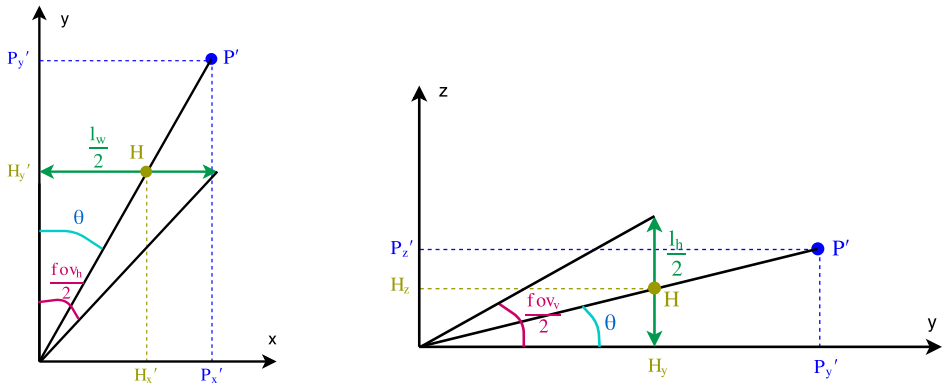
$$\begin{aligned} S'(d, f_{\text{pos}}, \beta, \gamma) &= (f_{\text{pos}} * cos(\beta) * cos(\gamma), f_{\text{pos}} * cos(\beta) * sin(\gamma), f_{\text{pos}} * sin(\beta)) \\ C(d, f_{\text{pos}}, f_{\text{att}}, \beta, \gamma) &= S' + (0, d * cos(f_{\text{att}}), 0) \\ P'(d, f_{\text{pos}}, f_{\text{att}}, \alpha, \beta, \gamma) &= C + (d * sin(f_{\text{att}}) * cos(\alpha), 0, d * sin(f_{\text{att}}) * sin(\alpha)) \end{aligned} \tag{4.10}$$

where α, β and γ are determined in the previous models. Then we determine the distance v and the angle ϕ:

$$\begin{aligned} \phi(d, f_{\text{pos}}, f_{\text{att}}, \alpha, \beta, \gamma) &= acos(\frac{P'_y}{\left\|\vec{P'}\right\|}) \\ v(d, f_{\text{pos}}, f_{\text{att}}, \alpha, \beta, \gamma) &= \left\|P'P\right\| \\ &= \sqrt{(f_{\text{pos}} * cos(\beta) * cos(\gamma) + d * sin(f_{\text{att}}) * cos(\alpha))^2 + \\ &\quad \overline{(f_{\text{pos}} * cos(\beta) * sin(\gamma) + d * cos(f_{\text{att}}) - d)^2 + \\ &\quad \overline{(f_{\text{pos}} * sin(\beta) + d * sin(f_{\text{att}}) * sin(\alpha))^2} \end{aligned} \tag{4.11}$$

The representation of the model and a simulation are given in Figure 4.4.

4.3 Projected Distance on the Screen



(a) Projection on x-axis **(b)** Projection on z-axis

Figure 4.5: Projection of distance error on the screen

In the previous section, we proposed a model to represent distance error (v) and angle error (ϕ) between the estimated feature and the real feature. So far this does not allow us to know how far will the virtual feature be displayed compared to the real one on the screen of the device. This distance (e) depends on two additional parameters: screen size (l) and the field of view (fov). We consider $H(H_x, H_z)$, the projection of $P'(P'_x, P'_y, P'_z)$ on the screen of the device (see Figure 4.5) and consequently: $e = \|H\|$. For the sake of clarity, in this section we will consider $P' = P'(d, f_{\text{pos}}, f_{\text{att}}, \alpha, \beta, \gamma)$.

In the following equations, we consider l_w and l_h, respectively the width and the height of the device screen. The relation between l_w and l_h is expressed with the aspect ratio ar:

$$l_h = ar * l_w. \tag{4.12}$$

fov_H and fov_V are respectively the horizontal and the vertical field of view of the camera. The relation between fov_H and fov_V is given by:

$$fov_V = 2 * atan(tan(\frac{fov_H}{2}) * ar) \tag{4.13}$$

Now, we are looking for H given P', l and fov:

$$\begin{aligned} H_x(P', fov_H, l_w) &= tan(\theta) * u \qquad \text{where: } u = \frac{l_w}{2 * tan(\frac{fov_H}{2})} \text{ and } \theta = atan(\frac{P'_x}{P'_y}) \\ &= \frac{P'_x}{P'_y} * \frac{l_w}{2 * tan(\frac{fov_H}{2})} \\ H_z(P', fov_V, l_h) &= \frac{P'_z}{P'_y} * \frac{l_h}{2 * tan(\frac{fov_V}{2})} \end{aligned} \tag{4.14}$$

We will show that $H_x(P', fov_H, l_w) = H_x(P', fov_V, l_h)$:

$$\begin{aligned} H_x(P', fov_H, l_w) &= \frac{P'_x}{P'_y} * \frac{l_w}{2 * tan(\frac{fov_H}{2})} \\ H_x(P', fov_V, l_h) &= \frac{P'_x}{P'_y} * \frac{l_h}{ar} * \frac{1}{2 * tan(\frac{1}{2} * 2 * atan(tan(\frac{fov_H}{2}) * \frac{1}{ar}))} \\ &= \frac{P'_x}{P'_y} * \frac{l_w}{2 * tan(\frac{fov_H}{2})} \\ &= H_x(P', fov_H, l_w) \end{aligned} \tag{4.15}$$

Consequently, H_x and H_z can be expressed with the same pair of <field of view, screen size> (horizontal or vertical). Finally:

$$\begin{aligned} e(P', fov, l) &= \|H\| = \sqrt{H_x^2 + H_z^2} \\ &= \sqrt{(\frac{P'_x}{P'_y} * \frac{l}{2 * tan(\frac{fov}{2})})^2 + (\frac{P'_z}{P'_y} * \frac{l}{2 * tan(\frac{fov}{2})})^2} \\ &= \frac{\sqrt{P'^2_x + P'^2_z}}{P'_y} * \frac{l}{2 * tan(\frac{fov}{2})} \end{aligned} \tag{4.16}$$

4.4 Scores from our Benchmarks: Examples on Different Use Cases

The distance v between the estimated feature and the real feature, as well as its projection e will allow us to verify if an AR application is valuable for a specific use case. For example,

is a Geo AR system accurate enough to turn around a 3D virtual object in a mall? If not, what precision is needed?

To answer these questions, we used errors from attitude estimation and positioning estimation from our benchmarks in Chapters 2 and 3. We have a little adapted the evaluation method proposed previously to correspond to data from our benchmarks:

- In Section 3.4, f_{pos} was calculated on 2 dimensions. Our benchmark does not allow us to know positioning error vertically. For this reason, we used the 2D model proposed in Eq 4.8, therefore, $e_{\text{2D}}(d, f_{\text{pos}}, f_{\text{att}}, \alpha, \gamma, fov, l) = e(d, f_{\text{pos}}, f_{\text{att}}, \alpha, 0, \gamma, fov, l)$.
- In Section 2.4, the estimated attitude error was given in QAD. Theoretically, f_{att} is not equal to the attitude error, because QAD does not only represent angle between both vectors but also the rotation of the feature around $\vec{SP}$-axis. But, as external accelerations are very low during AR motions, the rotation around $\vec{SP}$-axis will be little affected, so we will consider $f_{\text{att}} \simeq$ QAD error.

Then, we propose to see how accurate the Geo AR applications are on different use cases. Results for each use case will be detailed with the average (AVG) and standard deviation (STD) of distances e and v as well as angle the θ (see Fig 4.1). As e, is dependent to the size of a device and camera FOV, we fixed them to $l = 11.4\ cm$ and $fov = 60°$ which correspond to the metrics of a Nexus 5X. The problem is that in our benchmark we do not know in which direction the bad estimated attitude is pointing (α), we only know its magnitude (f_{att}). We also do not know in which direction the bad estimated position is (γ), we only know its magnitude (f_{pos}). From this information, we consider E_{2D} be the average value of e_{2D}:

$$E_{\text{2D}}(d, f_{\text{pos}}, f_{\text{att}}, fov, l) = \frac{\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} e_{\text{2D}}(d, f_{\text{pos}}, f_{\text{att}}, \alpha, \gamma, fov, l)\ d\alpha\ d\gamma}{\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} 1\ d\alpha\ d\gamma} \tag{4.17}$$

For its implementation, we used `integral2` function of MATLAB.

Average (μ) and standard deviation (σ) of E from a vector of position errors (F_{pos}) and a vector of attitude errors (F_{att}) are defined by:

$$\begin{aligned}
\mu_E(d, F_{\text{pos}}, F_{\text{att}}, fov, l) &= \frac{\sum_{f_{\text{pos}} \in F_{\text{pos}}} (\sum_{f_{\text{att}} \in F_{\text{att}}} (E_{\text{2D}}(d, f_{\text{pos}}, f_{\text{att}}, fov, l)))}{\sum_{F_{\text{pos}}} \sum_{F_{\text{att}}}} \\
\sigma_E(d, F_{\text{pos}}, F_{\text{att}}, fov, l) &= \sqrt{(\frac{\sum_{f_{\text{pos}} \in F_{\text{pos}}} (\sum_{f_{\text{att}} \in F_{\text{att}}} ((E_{\text{2D}}(d, f_{\text{pos}}, f_{\text{att}}, fov, l) - \mu_e)^2))}{\sum_{F_{\text{pos}}} \sum_{F_{\text{att}}}})}
\end{aligned} \tag{4.18}$$

In the same manner, we defined $\mu_{V_{\text{2D}}}$, $\sigma_{V_{\text{2D}}}$, $\mu_{\Theta_{\text{2D}}}$, $\sigma_{\Theta_{\text{2D}}}$, the average and standard deviation of v and θ from a vector of position errors (F_{pos}) and a vector of attitude errors (F_{att}) In the rest of the chapter, for the sake of clarity, we consider $e = E_{\text{2D}}$, $v = V_{\text{2D}}$, $\theta = \Theta_{\text{2D}}$. We then evaluated 4 typical use cases of Geo AR.

4.4.1 Use Case 1: An Application to Identify Mountains and Cities

From a high position, a person wants to identify mountains and cities around him. The person is on a hiking trail or in a ski resort and the space around him is clear.

- **Position:** From GNSS outside ($avg \simeq 3.54\ m$)
- **Attitude:** From AR with a low magnetic perturbations ($avg \simeq 4.5°$). In reality, it should be less, because our benchmark was setup in a building and we noticed some magnetic perturbations.
- **Feature Distance:** From $1\ km$ to $50\ km$.

	Screen distance (e)		Real to virtual distance (v)		Real to virtual angle (θ)	
	AVG	STD	AVG	STD	AVG	STD
Feature at $1\ km$	$0.77\ cm$	$0.02\ cm$	$78.6\ m$	$1.8\ m$	$4.5°$	$0.10°$
Feature at $10\ km$	$0.77\ cm$	$0.00\ cm$	$785.2\ m$	$1.8\ m$	$4.5°$	$0.01°$
Feature at $50\ km$	$0.77\ cm$	$0.00\ cm$	$3926\ m$	$1.8\ m$	$4.5°$	$0.00°$

Table 4.1: Scores of use case 1: an application to identify mountains and cities

Table 4.1 exhibits results for 3 distances of features ($1\ km$, $10\ km$ and $50\ km$). The average distance error on the screen is not high, even more if an attitude error of $2°$ is considered (e reduces to $0.34\ cm$). Conversely, if the positioning error grows to $50\ m$; e, v and θ remain almost unchanged. One last important thing, is the linear growing of v in function of the feature distance. It means that, if the user is aiming a $50\ km$-far summit, he has a good chance to point another mountain $4\ km$ on the side than the real one. Finally, if we consider the user in a building instead of a clear space, then we place a feature at $10\ km$: $e = 1.9\ cm$ and $v = 1880\ m$. This shows the limits of aiming a far-feature from a place impacted by magnetic perturbations.

4.4.2 Use Case 2: An Application to Discover the History of a City

A person is touring a city. He wants to learn more about the history of this city. He uses his smartphone to read stories by pointing it to old buildings.

- **Position:** From GNSS in downtown (we did not benchmark for this use case, but from literature [Ben-Moshe et al., 2011], $avg \simeq 15\ m$)
- **Attitude:** From AR with a low magnetic perturbations ($avg \simeq 4.5°$).
- **Feature Distance:** From $5\ m$ to $100\ m$.

Table 4.2 shows results for 4 distances of features ($5\ m$, $20\ m$, $30\ m$ and $100\ m$). In this use case, it is not possible to provide a good AR experience if the feature is too close from the user. But, if building is big enough ($>\ 30\ m$-wide), from $d = 30\ m$, AR starts to be more reliable because, the feature information will be shown as an overlay of the building. This is true only if the building size is greater than $2 * v$. We also tried the $30 - m$ far use case with a better estimation of attitude $f_{att} = 1°$ and results obtained are similar. However, if we decrease f_{pos} to 5 meters, when $d = 30\ m$, the projected error e on the screen is equal to $1.28\ cm$. This shows that to enhance reliability of this use case, more efforts needs to be made on the positioning estimation than the attitude estimation.

	Screen distance (e)		Real to virtual distance (v)		Real to virtual angle (θ)	
	AVG	STD	AVG	STD	AVG	STD
Feature at 5 m	$+\infty^*$	$+\infty^*$	15.0 m	0.2 m	77.7°	50.44°
Feature at 20 m	6.29 cm	3.62 cm	15.1 m	0.8 m	30.2°	15.00°
Feature at 30 m	3.59 cm	1.81 cm	15.1 m	1.2 m	19.5°	9.25°
Feature at 100 m	1.20 cm	0.56 cm	16.5 m	3.7 m	6.9°	3.17°

* e is not provided when $f_{pos} > d$ because P' is not projected on the screen, sometimes it is behind the user.

Table 4.2: Scores of use case 2: an application to discover the history of a city

4.4.3 Use Case 3: Make 3D Models Appear and Turn Around in an Indoor Environment

In a building, a user makes a 3D model appear (e.g.: a cat). Then, he turns around to look the 3D model from other angles.

- **Position:** From SHS + Map-Matching ($avg \simeq 2.26\ m$)
- **Attitude:** From AR with a high magnetic perturbations ($avg \simeq 10.8°$).
- **Feature Distance:** From $0.5\ m$ to $2\ m$.

	Screen distance (e)		Real to virtual distance (v)		Real to virtual angle (θ)	
	AVG	STD	AVG	STD	AVG	STD
Feature at 0.5 m	$+\infty^*$	$+\infty^*$	2.3 m	0.0 m	82.0°	51.24°
Feature at 1 m	$+\infty^*$	$+\infty^*$	2.3 m	0.1 m	73.7°	48.93°
Feature at 2 m	$+\infty^*$	$+\infty^*$	2.3 m	0.2 m	53.5°	35.09°

* e is not provided when $f_{pos} > d$ because P' is not projected on the screen, sometimes it is behind the user.

Table 4.3: Scores of use case 3: make 3D models appear and turn around in an indoor environment

Table 4.3 shows the results for 3 distances of features (0.5 m, 1 m and 2 m). It is clearly shown that using the Geo AR for this kind of application is not relevant. The virtual feature will almost never be displayed at the right place on the screen because $f_{pos} > d$. As long as f_{pos} is not significantly higher than the distance to the feature (d), this AR use case remains not reliable.

4.4.4 Use Case 4: Identify and Interact with Objects in a Building using UWB

A user points objects in a house to monitor the energy consumption (e.g. radiators, fridge) or to interact with them (e.g. lights, blinds).

- **Position:** From UWB ($avg \simeq 0.49\ m$)
- **Attitude:** From AR with a high magnetic perturbations ($avg \simeq 10.8°$).

	Screen distance (e)		Real to virtual distance (v)		Real to virtual angle (θ)	
	AVG	STD	AVG	STD	AVG	STD
Feature at $0.5\ m$	$37.76\ cm$	$105.29\ cm$	$0.5\ m$	$0.0\ m$	$45.0°$	$25.52°$
Feature at $1\ m$	$4.02\ cm$	$2.07\ cm$	$0.5\ m$	$0.1\ m$	$21.4°$	$10.05°$
Feature at $2\ m$	$2.41\ cm$	$1.13\ cm$	$0.6\ m$	$0.2\ m$	$13.6°$	$6.14°$
Feature at $5\ m$	$1.96\ cm$	$0.50\ cm$	$1.0\ m$	$0.2\ m$	$11.2°$	$2.82°$

Table 4.4: Scores of use case 4: identify and interact with objects in a building using UWB

- **Feature Distance:** From $0.5\ m$ to $5\ m$.

Table 4.4 exhibits results for 4 distances of features ($0.5\ m$, $1\ m$, $2\ m$ and $5\ m$). The results are averaged. This use case mostly works when features are far. We also tried the $1-m$ far use case with a better estimated attitude $f_{att} = 2°$ (divided by 5) and we obtained $e = 3.4\ cm$. The difference with the previous e is not very noticeable. However, if we decrease f_{pos} to 0.25 meters (divided by 2), we obtain $e = 2.2\ cm$. Once again, to improve the accuracy of AR for this use case, efforts must be made on the positioning estimation.

4.5 Conclusion

We proposed an evaluation method to calculate the average distance between a real and a virtual feature represented on the screen of the device. This system takes as input vectors of errors from the attitude estimation and positioning estimation approaches. We have shown that the distance between the feature and user has a huge impact on rendering. Finally, we gave some examples of use cases of AR applications, where we applied our evaluation method to know if the Geo AR is relevant. It showed us that for the moment, some AR applications cannot be developed with theGeo AR (e.g. turn around a 3D model without a significant progress on the positioning estimation), while others are promising (e.g names of mountains or monitoring object indoor with UWB).

Chapter 5

On a Geo AR Browser Framework

Contents

5.1 OSM for AR Documents 88
5.1.1 A Format over OSM Specifications 88
5.1.2 JOSM: A Fast Authoring Tool for Geo AR 91
5.2 Geo AR Viewer: From Reality to Virtual World 91
5.2.1 Positioning Features in the OpenGL Scene 92
5.2.2 Orientation: from Sensors to OpenGL 93
5.2.3 Camera stream: Field Of View and Aspect Ratio 95
5.2.4 Scheduling and Updating the OpenGL Scene 96
5.3 AR Browser Framework 97
5.3.1 Mapbox to handle Indoor Vector Maps 98
5.3.2 Core of the Framework 99
5.3.3 AR Browser Applications 101
5.4 Conclusions and Perspectives 105

In Chapters 2 and 3, we evaluated some pose estimation approaches using the geolocalisation and attitude. Then in Chapter 4, we studied the feasibility of such a system. In the continuity, we wanted to test our algorithms in a real AR browser.

As there is no project which proposes an open-source stack (from authoring to rendering) for the Geo AR, we decided to develop and describe our own approach of a Geo AR browser. Creating the full stack (format, authoring, rendering) of an AR browser may take a long time. In recent years, the OpenStreetMap project grew a lot [1] and several tools developed by the community become very useful for our developements. Moreover, thanks to APIs like overpass [2], it is possible to fill AR experiences with millions of Points of Interest (POI)[3].

In Section 5.1, we first show how we reused OSM XML specifications to propose our own format for Geo AR. Then, in Section 5.2, we will describe how the virtual features are displayed

[1] `http://wiki.openstreetmap.org/wiki/Stats`

[2] The Overpass API is a read-only API that serves custom selected parts of the OSM map data. `https://overpass-turbo.eu/`

[3] `https://taginfo.openstreetmap.org/reports/database_statistics`

over the smartphone camera stream. Finally, in Section 5.3, we introduce our Geo AR browser and give an example of applications we developed with.

In Section 5.2 and 5.3, we mainly focus our work on Android SDK but a similar approach can be developed with another operating system.

5.1 OSM for AR Documents

In Section 1.4.1, we introduced ARML [Lechner, 2015], a rich markup language for AR. The language is well-defined and covers a lot of use cases: vision, geo-based, 3D models, labels... but not non-viewable objects like audio. The main problem we encountered with ARML is the impossibility to easily generate a document. Not even one of the professionnal frameworks which deals with ARML (Junaio, Layar and Wikitude) provide an interface to place virtual features on a map. That is why we chose to create our own format. Our format needs to be as extensible as possible to build a multitude of applications and handle many concepts: indoor, outdoor, labels, audio, polygons, 3D models... Especially, it is necessary to find a format that could be easily generated.

5.1.1 A Format over OSM Specifications

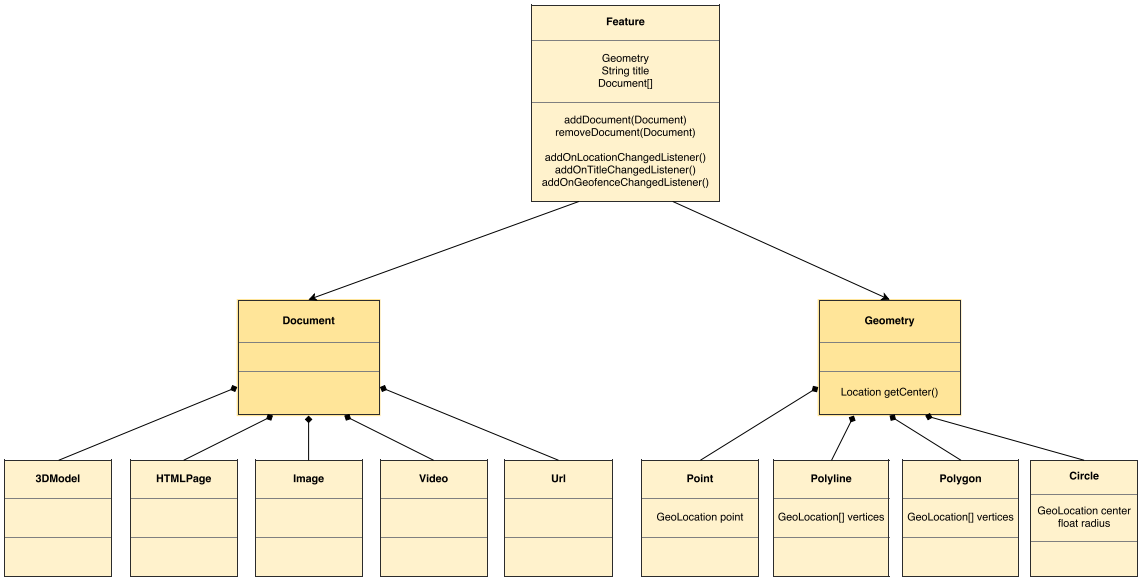


Figure 5.1: Hierarchy of features and documents handled by our format.

The idea here is to reuse OSM XML specifications to propose our own format and take advantage of the power and the multitude of tools from the OSM community. OSM database contains billions of data which can be directly reused for city bounds, buildings positions, summits... Furthermore, we already used OSM data in our ecosystem: for map-matching (see Section 3.3) and for rendering (see Section 5.3.1). Unlike in [Scioscia et al., 2014] where authors proposed an ontology, we propose to describe features with pairs of $< key, value >$ tags as well

as OSM XML specifications. Below is a list of tags we propose for our AR browser (tags can be used on nodes, ways or area).

$<$ ***feature***, ***yes*** $>$ (mandatory) This tag defines if an OSM element is a feature or not.

$<$ ***name***, ***[name]*** $>$ This is the primary tag used for naming an element. *[name]* is a string defined by user. Tag was already provided by OSM specifications but not exclusively for features.

$<$ ***category***, ***[category]*** $>$ Define the category of a feature. *[category]* is a string defined by user.

Each feature can be described by one or more documents (image, video, html...). Figure 5.1 shows the hierarchy of features and documents.

$<$ ***web-local-page***, ***[file-path]*** $>$ A local html page.

$<$ ***image***, ***[file-path]*** $>$ An image.

$<$ ***video***, ***[file-path]*** $>$ A video.

$<$ ***audio***, ***[file-path]*** $>$ An audio soundtrack.

$<$ ***3dmodel***, ***[file-path]*** $>$ A 3D model.

$<$ ***3dmodel-heading***, ***[heading]*** $>$ Horizontal orientation in degrees from north of the 3D model.

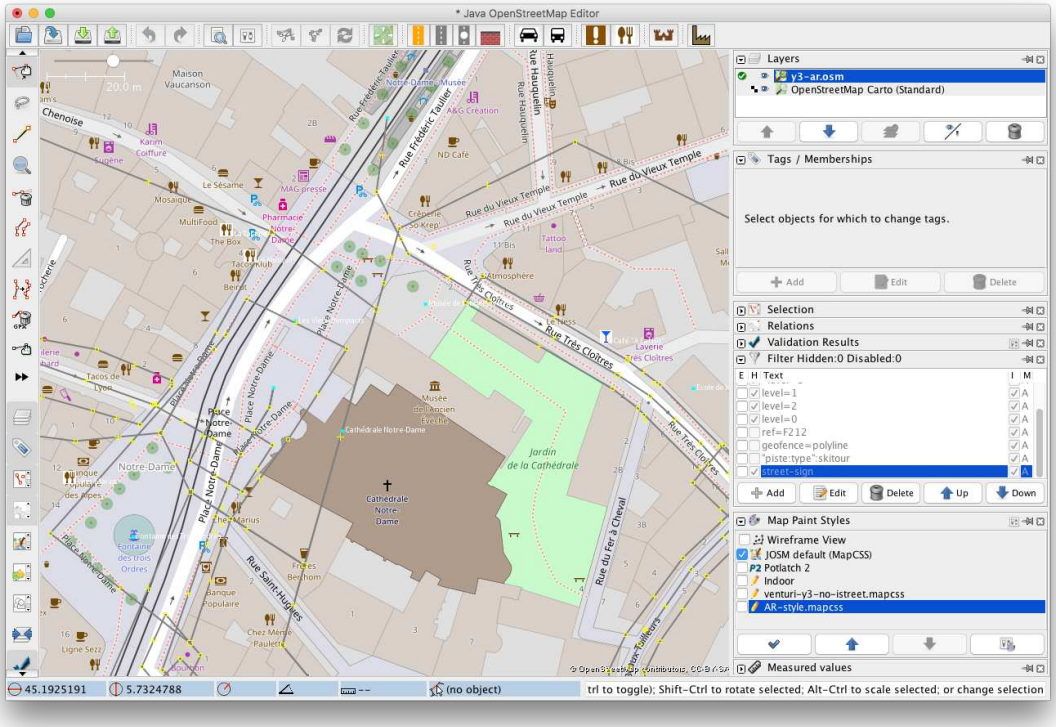
Finally features can be triggered using a geofence.

$<$ ***geofence***, ***[geofence-type]*** $>$ Triggering area type. *[geofence-type]* can be: *circle*, *polyline*, *polygon*.

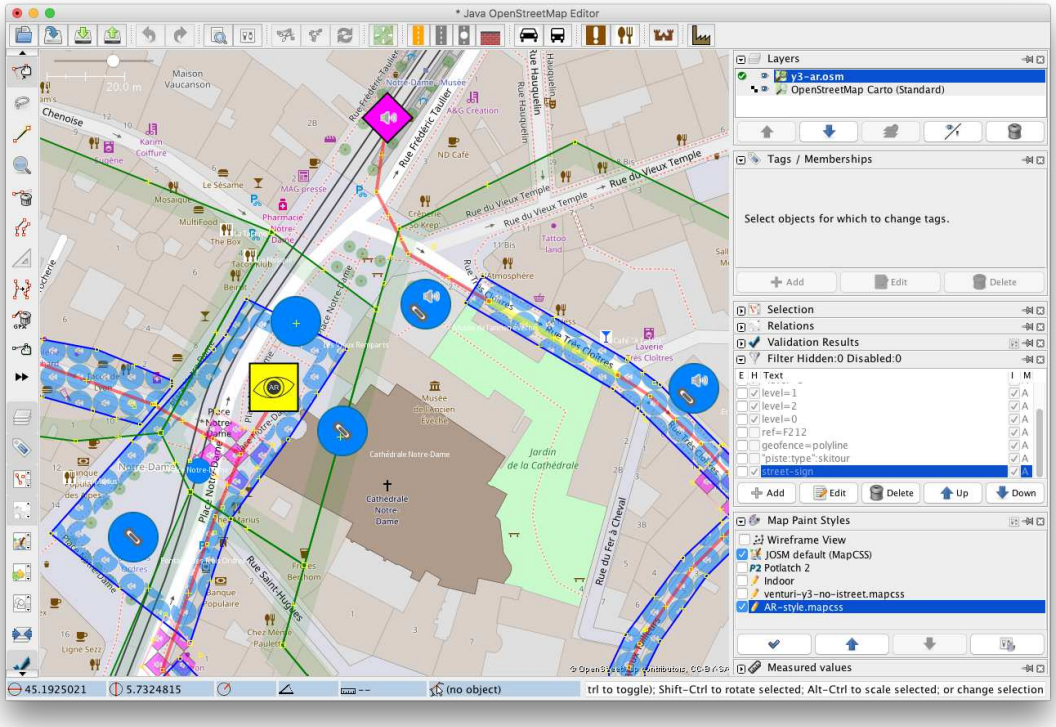
$<$ ***geofence-radius***, ***[radius]*** $>$ If the geofence is *circle* or *polyline*, *[radius]* corresponds to the radius in meters in which the geofence will be triggered around the element.

Tags who are already present on a node or way can be reused (e.g: *wikipedia*, *amenity*...). New tags can be added for the necessity of an application (e.g: *audio-background*, *tour-number*...). An example of a node feature with 2 documents (audio and web page) and a circle geofence with a radius of 20 meters is given below:

```
<node id='-33126' lat='45.19302533994' lon='5.73215301873'>
        <tag k='feature' v='yes' />
        <tag k='audio' v='musee-ancien-eveche.mp3' />
        <tag k='geofence' v='circle' />
        <tag k='geofence-radius' v='20' />
        <tag k='category' v='museum-archaeological' />
        <tag k='tour-number' v='2' />
        <tag k='web-local-page' v='musee-eveche.html' />
        <tag k='name' v='Musée de l'ancien évêché' />
        <tag k='name:en' v='Museum of the Former Bishopric' />
        <tag k='wikipedia' v='fr:Musée de l'Ancien Évêché' />
</node>
```



(a) Screenshot of JOSM without map paint style for AR



(b) Screenshot of JOSM with map paint style for AR

Figure 5.2: Comparison of JOSM with and without map paint style for AR

5.1.2 JOSM: A Fast Authoring Tool for Geo AR

Creating manually an OSM XML document for AR is really tedious. Hopefully, the OSM community developed lots of tools to enhance generation of OSM documents: iD[4] and Potlatch[5] which are online editors and JOSM[6] and Merkaartor[7] which are desktop softwares. The idea here is to divert one of these tools to generate our AR format. The tool we have focused on is Java OpenStreetMap Editor (JOSM) which is a free and open-source editing tool for OpenStreetMap geodata. JOSM allows us to reuse or create features easily using nodes, lines and polygons over orthophotos or classical maps. Then, create an AR document consists only of adding new tags on elements and generate the OSM file.

The rendering of JOSM map can be easily customized with different styles. Many styles come with JOSM by default, for example *JOSM standard* or *Potlatch 2* which are base styles that cover a wide range of features each. They can be combined with add-on styles that cover more special topics (e.g indoor mapping). That is why, we defined an add-on style specific to our AR format we proposed in the previous section. The style highlights features and documents attached to it. The source code of the style in MapCSS format is given in Appendix A.3 and an example of the style applied on a OSM AR document is shown in Figure 5.2. This style helps a lot the authors to quickly create AR documents. Furthermore, in case of specific applications, style sheet is really easy to customize.

5.2 Geo AR Viewer: From Reality to Virtual World

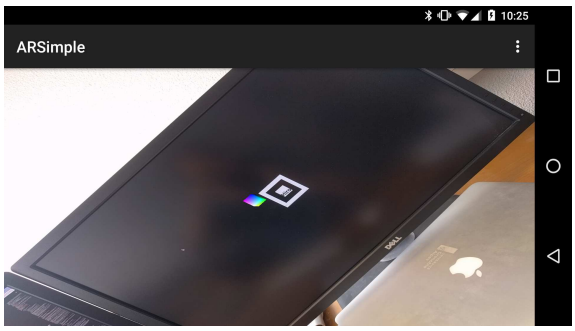


Figure 5.3: An example of AR application which does not respect camera aspect ratio. Camera stream should render the screen with right angles but here we observe a distortion.

As there does not exist any project which proposes an open-source stack (from authoring to rendering) for the Geo AR, in previous section we started by developing a new AR format based on OSM XML. Now, we will build our own AR viewer to exhibit features over the video stream from the camera of the smartphone. It is difficult to find information on how AR viewers works: (i) known projects (Layar, Wikitude...) do not provide any implementation information, (ii) to the best of our knowledge there exists only one scientific paper [Pryss et al., 2016] which deals

[4] http://ideditor.com/
[5] http://www.openstreetmap.org/edit?editor=potlatch2
[6] https://josm.openstreetmap.de
[7] http://merkaartor.be/

with an AR viewer implementation, but it is for 2D rendering and finally (iii) it is possible to find some code snippets on GitHub but they are often undocumented and abandoned for years. Among open-source projects, some of them [Bhide, 2014, Heinen, 2014, Nguyen, 2017] display geo-located texts and labels over the camera but they do not handle complex rendering like polygons, 3D models... and sometimes rendering seems not reliable. For example, we noticed for some of them than features move faster than the camera stream or the camera aspect ratio is not respected. Moreover, controlling the whole process of AR rendering allows us to understand problems experienced by others applications. In this section we will describe the process and technical choices we followed to build our own AR viewer.

The idea proposed here is to represent a virtual world in the same manner than a user who is moving in the real world. In other words, if user has moved one meter forward in the real world, in the virtual world his position will also move of one meter in the same direction. In order to set up such a system, we based our viewer on an OpenGL scene where axes denote the real world coordinates. In contrast to [Heinen, 2014, Puig Sanz, 2017] where axes represent an East-North-Up (ENU) frame centered on user position, we consider OpenGL axes as an Earth-Centered and Earth-Fixed (ECEF) frame. The camera in the OpenGL scene moves in function of the position and the orientation of user. This approach avoids to recalculate manually the position of each object relatively to the frame center at any new user location.

Contrary to iOS SDK with SceneKit, Android SDK does not provide any support library to place 3D objects in an OpenGL scene. In [Heinen, 2014, Puig Sanz, 2017], authors recreated their own 3D engine which supports some basic rendering: cubes, diamonds, triangles, labels... Instead of rewrinting a whole library to handle 3D models, transparency, occlusions..., why not use an existing 3D engine which is maintained by a huge community? This is the way we followed with the open-sourced Rajawali library. Rajawali is a 3D engine for Android based on OpenGL ES 2.0/3.0 maintained by more than 50 contributors (August 2017). All the features we introduced before and many others are supported by this library.

5.2.1 Positioning Features in the OpenGL Scene

To begin, we filled the OpenGL scene with features. Features are placed given a position and for some of them (images, labels, 3D models) an orientation. In our OSM AR format and more generally in all databases of features, the position of a feature is referenced with a latitude and a longitude in WGS84 format + altitude (geodetic coordinates). The formula to convert geodetic coordinates to ECEF coordinates is given in Section 1.1.3. The orientation of a feature differs in function of the use case, we separated them in two categories:

Fixed features They are features which have a fixed size and orientation. For example a 3D model of a vase on a table. User can turn around the object, if he moves backwards, vase will become smaller and if he moves forward vase will be bigger. Fixed features are mostly 3D models, textured surfaces and blocks. The orientation of features which are blocks are automatically calculated from their positions. For example, to draw a wall we will use a 3D plane which will be placed from one side of the wall to the other one. This is not the case for 3D models, images, videos or labels. We set up-vector of the 3D model in direction to the sky, towards the z-axis of ENU frame and rotation around this axis is fixed by the author (see Figure 5.4). This last rotation is defined by the tag *3dmodel-heading* in our OSM AR format.

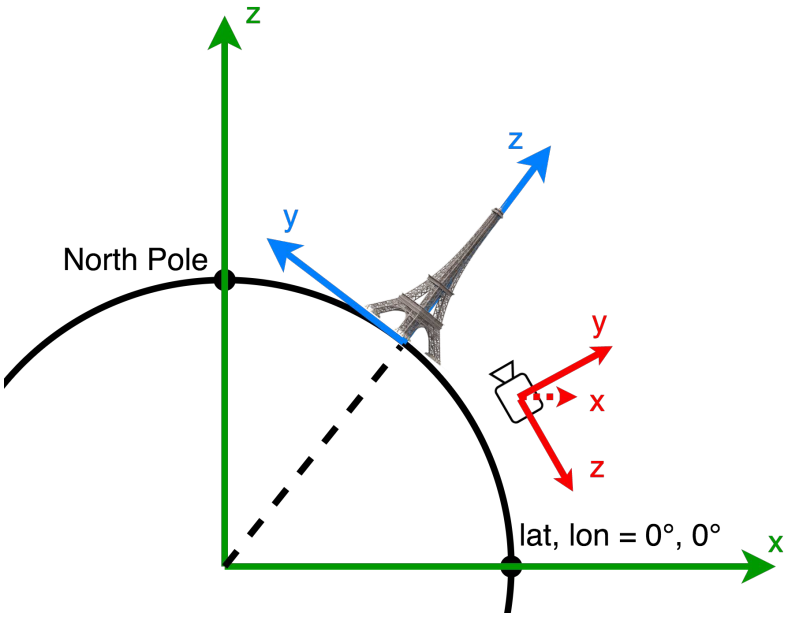


Figure 5.4: Representation of the 3 frames which deal with Rajawali: ECEF (in green), OpenGL Camera (in red) and a 3D Model (in blue).

Informational features They are features which have always the same size on screen and they are facing the camera. For example, the name and description of a building during an AR tour. They can be considered as Points of Interest (POI). It is not necessary to be at closer than 1 m to the building to read the description. Informational features are mostly labels, images or videos. Their positions are the same than fixed features to keep the right order of overlays but their orientations differ. Their orientations are not fixed because we want images or texts always facing the camera. Consequently, at each new user position, camera moves and features orientations need to be updated. In addition, we also have to maintain a scale factor to avoid small text/picture size due to the remoteness of the camera. Scale factor is defined by: `scale = 1/distanceBetween(camera, feature)`.

5.2.2 Orientation: from Sensors to OpenGL

In this second part, we will place and orient the OpenGL camera from estimated location and orientation of the device. The camera position is provided in the same manner than features using the geodetic to ECEF conversion (Eq 1.1). Rotation applied to the OpenGL camera consists on a succession of rotations from OpenGL frame to ECEF frame (see Figure 5.5).

The OpenGL frame and camera frame both use a right-handed coordinate system similar to the right hand Android system: the x-axis is horizontal with the positive direction to the right, the y-axis is vertical with the positive direction pointing up, and the z-axis is depth (forward and back), with the positive direction pointing out of the screen and towards the user. If an Android device in its default orientation is being used to control a camera, the device and OpenGL camera are using the same coordinate system. Otherwise, a rotation around the z-axis is applied (e.g. in

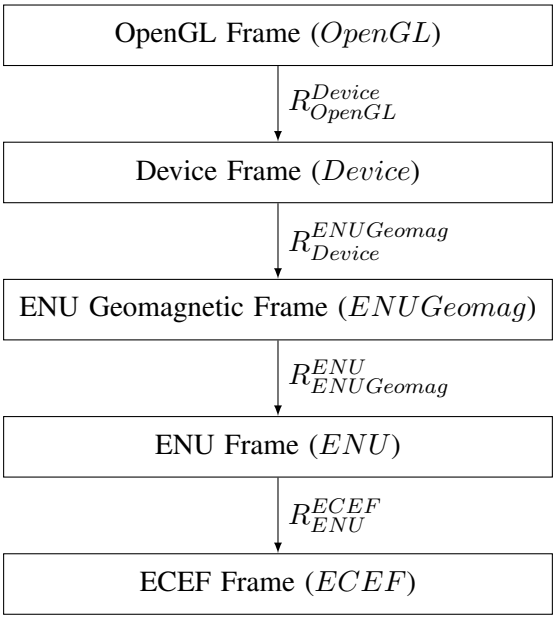


Figure 5.5: OpenGL camera: a succession of rotations from OpenGL frame to ECEF frame.

landscape mode, a rotation of $90°$ is applied. Global formula is given below:

$$R^{Device}_{OpenGL} = R_z(-\alpha), \tag{5.1}$$

where α is the screen orientation (portrait $0°$, landscape $90°$, reverse portrait $180°$, reverse landscape $270°$).

In Chapter 2, we studied the attitude estimation of a smartphone using different filters. We have defined the attitude as the rotation from the device frame to the ENU geomagnetic frame:

$$R^{ENUGeomag}_{Device} = R_{attitude}, \tag{5.2}$$

The ENU geomagnetic frame is a local tangent plane frame like ENU where the y-axis is pointing toward the magnetic north instead of geographic north.

Therefore, the next rotation we consider is from ENU geomagnetic frame to ENU frame and is defined by:

$$R^{ENU}_{ENUGeomag} = R_z(dec), \tag{5.3}$$

where dec is declination angle defined by WMM [(NGA) and the U.K.'s Defence Geographic Centre (DGC), 2015].

Then, the last rotation is from ENU frame to ECEF frame:

$$R^{ECEF}_{ENU} = R_z(-\frac{\pi}{2} + \lambda)\ R_x(-\frac{\pi}{2} - \phi), \tag{5.4}$$

where ϕ is the latitude and λ the longitude.

Finally, the whole rotation applied to the OpenGL camera is the succession of all these rotations:

$$\begin{aligned} R^{ECEF}_{OpenGL} &= R^{Device}_{OpenGL}\, R^{ENUGeomag}_{Device}\, R^{ENU}_{ENUGeomag}\, R^{ECEF}_{ENU} \\ &= R_z(-\alpha)\, R_{attitude}\, R_z(dec)\, R_z(-\frac{\pi}{2} + \lambda)\, R_x(-\frac{\pi}{2} - \phi) \end{aligned} \tag{5.5}$$

In Figure 5.4, the last 3 rotations correspond to the rotation from the red frame (Device) to the green frame (ECEF).

The OpenGL camera is now placed and oriented in function of device location and device attitude.

5.2.3 Camera stream: Field Of View and Aspect Ratio

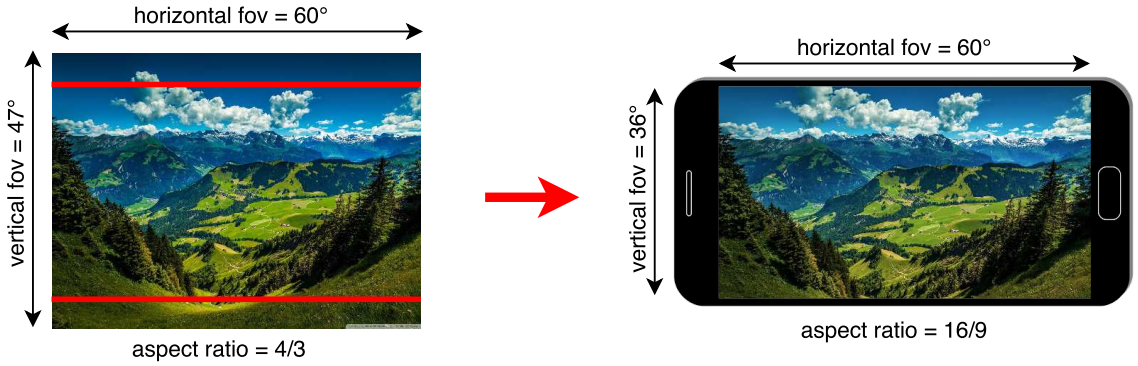


Figure 5.6: Camera feed scaled to fill the size of the view and keep aspect ratio.

The background of our OpenGL view has been set to transparent, this allows us to display the scene as an overlay of the video stream. Android provides libraries to retrieve camera information and display the video feed. It is rare that aspect ratio from the screen of the device is the same than the aspect ratio from the focal length of the camera. Therefore, to adapt the stream to the screen we have 3 choices:

1. Scale the content to fit the screen size by changing the aspect ratio of the content if necessary.
2. Scale the content to fit the size of the view by maintaining the aspect ratio. Any remaining area of the view's bounds is transparent.
3. Scale the content to fill the size of the view. Some portion of the content may be clipped to fill the view's bounds.

The first option, changing the aspect ratio of the content does not work for AR, OpenGL scene must deal with orthonormal coordinates. The second option will let the AR experience less immersive than the third one due to the empty areas. This is for these reasons that we based our browser on the third approach (see Figure 5.6). As the video feed will be clipped, the field of view (FOV) displayed on the screen will be different than the field of view provided by the camera. For example, in Figure 5.6, the top and bottom are clipped, and consequently the vertical FOV

is modified due to a different aspect ratio. The relation between horizontal FOV (fov_h), vertical FOV (fov_v) and aspect ratio (ar) is given below:

$$\begin{aligned} fov_h &= 2 * atan(tan(\frac{fov_v}{2}) * \text{ar}) \\ fov_v &= 2 * atan(tan(\frac{fov_h}{2})/\text{ar}) \end{aligned} \tag{5.6}$$

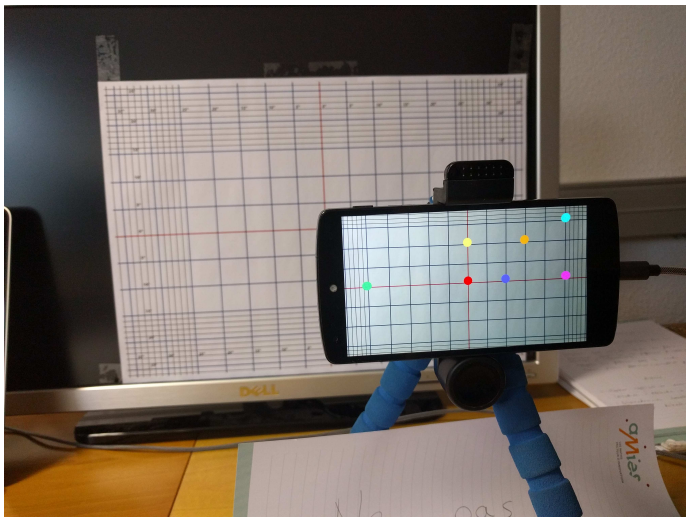


Figure 5.7: Virtual spheres placed at a predefined position on a custom target to verify FOV.

Horizontal FOV and aspect ratio are then passed in parameters of the OpenGL camera to build the perspective matrix. We built an experimental setup to verify if the horizontal and vertical FOV provided by the camera correspond to the reality (see Figure 5.7). A target has been drawn to show FOV angles at a fixed distance, we chose $0.3\ m$. The last line displayed at the top (or the bottom) of the screen corresponds to the vertical FOV. In a same manner, the last line displayed at the left (or right) of the screen corresponds to the horizontal FOV. In addition, we virtually placed spheres into the 3D scene at known distances to verify if they match the reality. For example, on Figure 5.7, a red sphere was placed at $< 0, 0, -0.3 >$, a yellow sphere at $< 0, tan(10°) * 0.3, -0.3 >$... Experiments have been conducted with different smartphones and different view size. As a result, we observed that the virtual points are displayed at the right positions (on intersections). We shown that our 3D viewer is working if the position and orientation of the device are known. Now, in the next part, we will introduce the device position and orientation as an input of our OpenGL scene.

5.2.4 Scheduling and Updating the OpenGL Scene

Many information and events need to be taken into account in the calculation of the 3D scene: device position, screen rotation, device orientation... These events have a direct impact on rotations, scales and FOV we defined in previous paragraphs. To avoid a too heavy process, some of them do not need to be calculated at each frame. For example, $R^{ENU}_{ENUGeomag}$, R^{ECEF}_{ENU} are preprocessed at each new position and R^{Device}_{OpenGL} is modified only when the screen is rotated. In

```
enuToEcefMatrix = Matrix3;
declinationMatrix = Matrix3;
screenOrientationMatrix = Matrix3;

informationalFeaturesList = List<Feature>;

Event NewPosition(latitude, longitude, altitude)
    cameraPosition = ECEF.fromGeodetic(latitude, longitude, altitude);

    // Update camera position
    camera.setPosition(cameraPosition);

    // Update ENU to ECEF Matrix
    enuToEcefMatrix = Rz(-pi/2 + longitude) * Rx(-pi/2 - latitude);

    // Update declination from World Magnetic Model
    declinationMatrix = Rz(WMM.getDeclination(latitude, longitude, altitude, now()));

    // Update all informational features
    upVector = cameraPosition.normalize();
    foreach feature in informationalFeaturesList do
        featureDirectionVec = camera.getPosition() - feature.getPosition();

        // Update feature model scale and orientation
        feature.setScale(1 / featureDirectionVec.norm());
        feature.lookAt(featureDirectionVec, upVector);
    end
end

Event ScreenOrientationChanged(orientation, fieldOfView)
    screenOrientationMatrix = Rz(orientation);
    camera.setFieldOfView(fieldOfView);
end

Event NewAttitude(attitude)
    camera.setRotation(screenOrientationMatrix * attitude * declinationMatrix * enuToEcefMatrix);
end
```

Figure 5.8: Pseudo-code of features update triggered by ”device location changed”, ”device attitude changed” or ”screen is rotated” events.

Figure 5.8, we described in pseudo-code how OpenGL camera and informational features are modified when events: *device location changed*, *device attitude changed* or *screen is rotated* are triggered.

Our AR viewer based on Rajawali is now ready to be used with our AR format and other algorithms we developed in previous chapters.

5.3 AR Browser Framework

In this work we introduced a multitude of concepts which are related to Augmented Reality: device attitude, device positioning, AR format, AR viewer.... Then, we chose to gather these concepts into a generic framework to build AR applications effortless. For this purpose, we developed a core library to link and schedule all the modules. But before that, a 2D map library which supports indoor data is missing for our framework.

5.3.1 Mapbox to handle Indoor Vector Maps

We often reduce an AR application to a camera stream with augmented objects, but, in reality, users need to be guided from an AR experience to another [DüNser et al., 2012]. For years, we have used maps to navigate from a position to a specific point. Many navigation systems (Google Maps, Bing Maps, MapsForge, ...) exist and can be embedded in a custom application to show user position and a direction to follow. These libraries handle outdoor data and use tiles with orthophotos or road networks. Some projects work also with indoor maps (WRLD 3D, IndoorAtlas, Google Maps, ...), but all of them deal with proprietary formats and are not free. Unfortunately, among mapping projects for Android, only few deal with OpenStreetMap data. The goal here is not to recreate a library from scratch but to use an existing library and add support of indoor information.

We retained two open-source projects: Mapsforge and Mapbox. The two share the same approach: the elements from OSM are converted to an intermediate format then a style sheet is applied. These two libraries give the possibility to add icons, polylines or polygons as an overlay of the map. These functionalities are primordial to show features on a 2D map. However, two main differences remain. First, Mapbox manages data using vector tiles whereas Mapsforge uses raster tiles. Second, Mapbox is supported on multiple platforms whereas Mapsforge is developed only for Android. Application will also be used for debugging indoor navigation algorithms in Matlab (see Section 3.2.2), therefore the desktop version of Mapbox is suitable. It is for these reasons that we chose to use Mapbox.

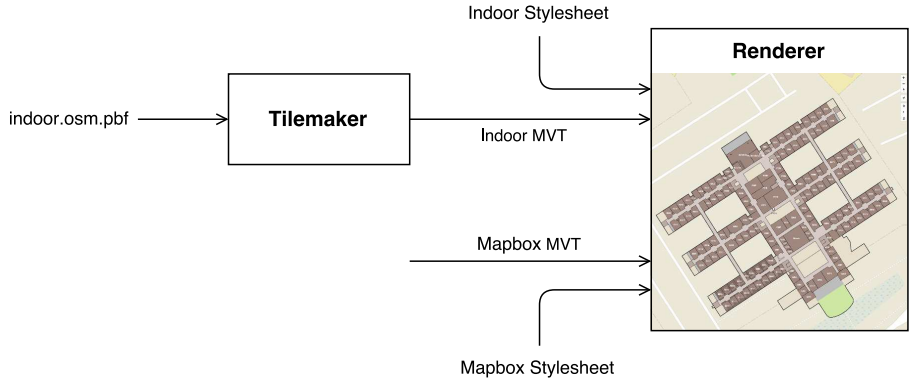


Figure 5.9: Mapbox indoor process.

Data source provided to Mapbox renderer is encoded in Mapbox Vector Tiles (MVT) format and can be accessed via this url:

```
http://a.tiles.mapbox.com/v4/mapbox.layer/z/y/x.mvt
```

where z, y and x are respectively zoom level and tile coordinates. Unfortunately, indoor information (like rooms, corridors, ...) from OSM specifications are not encoded in MVT and that is a problem because in addition to the style sheet we have to manage data conversion. In our approach, we kept the Mapbox style sheet with MVT as a base layer, then we added a custom data source and a custom style sheet to display indoor informations. We generated our indoor data source by converting OSM data to MVT using the open-source Tilemaker project [Fairhurst,

2015]. Figure 5.9 sums up the overall process. The input format ”OSM PBF” (Protocolbuffer Binary Format) is a more compact and fastest alternative of OSM XML.

Now, we obtained a specific renderer for indoor maps which have been designed for OSM. One specificity of indoor maps browsers remains that of managing multiple floors rendering and this is not supported by the Mapbox library natively. That is why we modified the source code of Mapbox GL to handle this important feature. We first worked on Mapbox GL JS library and an Android version is in progress. Our approach is detailed below.

Figure 5.10: Screenshots of Mapbox GL JS with our indoor plugin

First, we created a new module we called *Indoor*. The new module subscribes to events `tile.load` and `tile.remove` of the indoor MVT source. When a new tile is loaded or unloaded by Mapbox GL, we go through each feature to find tags `building:levels` and `building:min_level` of buildings. From these information, we create (or update) a *Mapbox Control* to navigate between floors (see Figure 5.10). Then, when a floor button is clicked, a Mapbox style filter is applied to display features at the selected level:

```
mapLayers.forEach(function(layer) {
  map.setFilter(layer, ["all", currentFilter, ["==", "level", selectedLevel]]);
});
```

where currentFilter is the raw style filter (without floor information). The key word `all` is equivalent to an *and gate*, it signifies: apply ”`currentFilter`” and ”filter where level is equal to `selectedLevel`”. An example of the rendering of our modified Mapbox GL JS is given in Figure 5.10.

5.3.2 Core of the Framework

The whole framework and links between modules are illustrated in Figure 5.11 and described below. As an input of our framework, the OSM AR format defined in Section 5.1 is parsed to generate the hierarchy of features and documents (see Figure 5.1). We designed the parser in a way that is generic enough to allow for the creation of new OSM tags for specific applications. Then features and documents are stored in a shared model (*ARModel*). OSM network (*OSMGraph*)

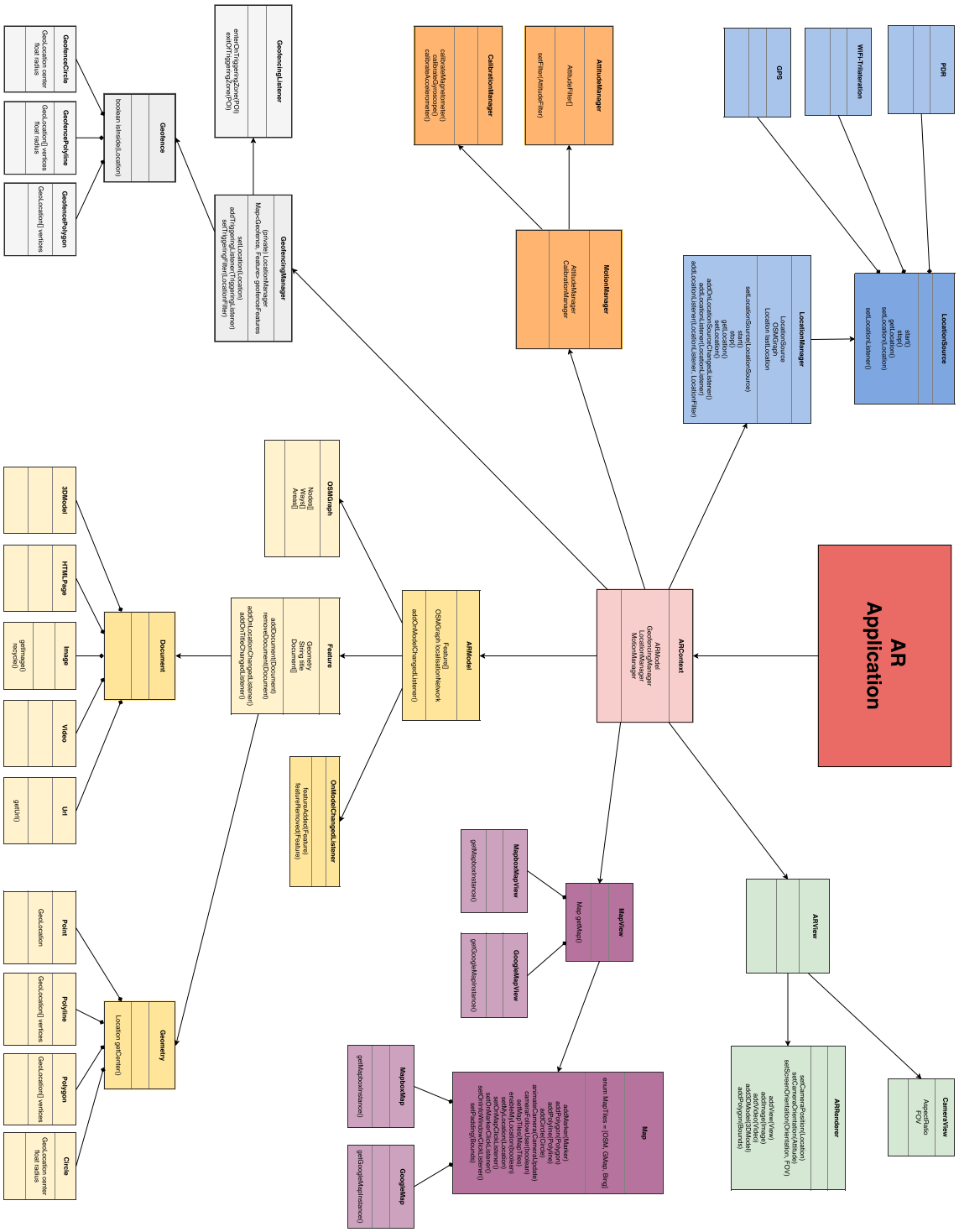


Figure 5.11: Proposed Geo AR Browser Overview.

is parsed in parallel of the AR document and it is also stored in the shared model. *ARModel* is an observable, so each modification (add a feature, remove a document, change location...) triggers a notification to observers. A super-entity (*ARContext*) knows *ARModel* and all others submodules (e.g. *LocationManager*, *MotionManager* and *ARView*). *ARContext* coupled with *GeofencingManager* serves as a scheduler to send location, attitude or features from model to the viewers. Actions triggered by the *GeofencingManager* are not pre-defined by the library, developer can choose the behavior of the application when user enters or exits an area (e.g. open an AR experience, play a sound...). *LocationManager* provides estimated position of the device for viewers and *GeofencingManager*. Some algorithms from approaches compared in Section 3.3 have been implemented. Approaches which use map-matching are dependent to map data, footpaths and walls. These information are directly provided by *OSMGraph* through the *ARContext*. *MotionManager* via *AttitudeManager* and *CalibrationManager* provides estimated attitude of the device. We implemented some filters among those we compared in Section 2.2. Finally, we propose two different renderings (2.5D or AR): *MapView* or *ARView* for a same dataset of features. Both viewers are also registered to modifications updates from the *ARModel*, this allows to visualize moving features, documents changes, etc. The device position and attitude are provided to viewer by *LocationManager* and *MotionManager*, respectively.

5.3.3 AR Browser Applications

This framework has been built to be as generic as possible and to facilitate the development of a wide range of AR applications. Now, we will present three applications we developed with this framework. The feasibility of these applications has been studied with our evaluation method before (see Section 4.4).

Tyr-AR

The first application is an AR viewer to name the mountains, cities and historical buildings over the camera feed of the smartphone. The user can turn on himself with his device to discover names and information about Points of Interest (POIs). POIs are directly extracted from the OSM database thanks to the Overpass Turbo API. POIs are displayed on the screen with their name, an icon and an extra information. City POIs exhibit the number of inhabitants, mountains are associated to their altitude and historical buildings display their date of construction. Screenshots of the application are given in Figure 5.12. It is also possible to add more POIs by clicking on the 2D map and by transitivity they are also available on the AR viewer. Provide altitude of a feature is mandatory to make our AR Browser work, and especially for this kind of application where there are huge differences in terms of elevation. POIs which are added manually and some extracted POIs do not contain the elevation information. For this purpose, we setup a web-service based on Shuttle Radar Topography Mission 1 (SRTM1) to retrieve ground elevation of a WGS84 position. The positioning system used by the AR Browser for rendering is only based on GNSS signals because accuracy is good enough for this type of outdoor application. The attitude filter provided by default with this application is *MichelObsExtMagWtRep* but others can be used via the application settings. Finally, we provided a video [8] which compares *MichelObsExtMagWtRep* filter and the built-in algorithm on this application.

[8] `https://www.youtube.com/watch?v=pKCOfHVTwIU` (TyrAr - Geo Augmented Reality on a Smartphone)

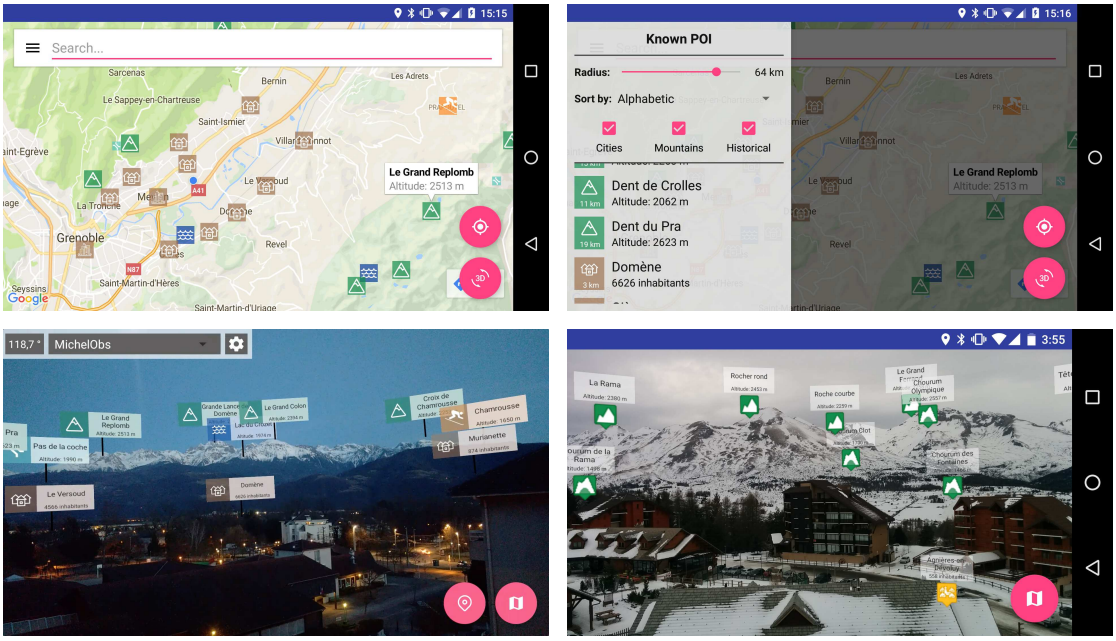


Figure 5.12: Screenshots of Tyr-AR application

Smart Home AR

The second application is a proof of concept of a Geo AR system in a smart apartment. This setup has been conducted in EquipEx Amiqual4Home[9]. The goal here is to control objects in the apartment using widgets over the video feed from the camera. For example, a user points a lamp with his smartphone, a widget appears, then he uses a slider in this widget to modify the light intensity. As the previous application, the user can choose to interact through a 2.5D map or an AR view. The 2.5D map is our mapbox-indoor viewer introduced in Section 5.10. We used JOSM (see Section 5.1.2) to create the features layer. Features we defined are lights, blinds or TV. A new tag have been added to the OSM AR format to associate an id to a feature. This id allows the application to send and receive requests to/from a JSON/REST service which deals with all interactive objects in the apartment (e.g "close the kitchen blind" or "turn off bedroom lights"). The position of the device is provided by a UWB tag fixed to the back of the smartphone. The UWB server processes and sends tag positions at $4Hz$ to the smartphone via MQTT (Message Queue Telemetry Transport)[10]. Using the same approach with UWB, we also provide a movable feature: a remote control. Its position is not defined in AR document, the updates came also from the MQTT service.

In Section 4.4.4, we obtained average results with this setup, this proved to be true in practice. During our calibration phase, we noticed a huge magnetic perturbations, north was indicated with more than 30° of error at some places in the room. This is the case for lots of buildings and this is the reason why we adapted our attitude filter to this specific use case as we now describe. When the application starts, we provide the attitude thanks to the *MichelObsExtMagWtRep* filter coupled

[9] `https://amiqual4home.inria.fr/`

[10] MQTT is an ISO standard publish-subscribe-based "lightweight" messaging protocol for use on top of the TCP/IP protocol.

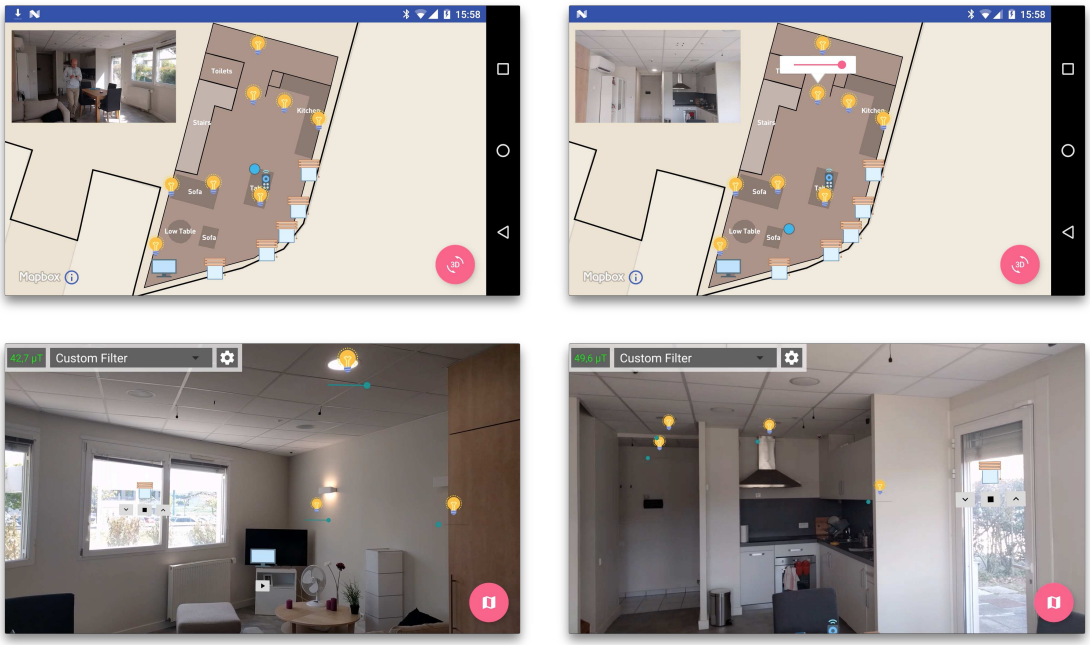


Figure 5.13: Screenshots of Smart Home AR application

with cross product of accelerometer and magnetometer from Martin et al. (see Section 2.2). Then, we suggest user to use two finger swipe to fix the attitude offset around z-axis due all magnetic perturbations[11]. As shown in Section 2.4.6, when a smartphone is static (this is mostly the case for AR) and magnetic perturbations are huge, errors come mainly from yaw angle. This is why, when the two finger swipe is used, we remove magnetometer measurements and we add the offset calculated by the two finger swipe movement. As we use cross product from Martin et al. and yaw is corrected by the two finger swipe, attitude estimation error is close to $0°$. The drawback of this technique is the drift of $5°.min^{-1}$ on yaw angle due to the gyroscope integration. However, the user can re-use the two finger swipe at anytime to re-adjust the system.

	Screen distance (e)		Real to virtual distance (v)		Real to virtual angle (θ)	
	AVG	STD	AVG	STD	AVG	STD
Feature at $0.5\ m$	$14.44\ cm$	$13.56\ cm$	$0.5\ m$	$0.0\ m$	$42.9°$	$24.01°$
Feature at $1\ m$	$3.37\ cm$	$1.73\ cm$	$0.5\ m$	$0.0\ m$	$18.4°$	$9.07°$
Feature at $2\ m$	$1.57\ cm$	$0.77\ cm$	$0.5\ m$	$0.0\ m$	$9.0°$	$4.37°$
Feature at $5\ m$	$0.62\ cm$	$0.30\ cm$	$0.5\ m$	$0.0\ m$	$3.6°$	$1.73°$

Table 5.1: AR scores for Smart Home AR application after adjusting z-axis with the two finger swipe

[11] `https://www.youtube.com/watch?v=_un8dfGPpNA` (AmiAr - Smart Home Augmented Reality on a Smartphone)

	Screen distance (e)		Real to virtual distance (v)		Real to virtual angle (θ)	
	AVG	STD	AVG	STD	AVG	STD
Feature at 0.5 m	16.28 cm	17.58 cm	0.5 m	0.0 m	43.8°	24.49°
Feature at 1 m	3.53 cm	1.77 cm	0.5 m	0.0 m	19.2°	9.08°
Feature at 2 m	1.79 cm	0.84 cm	0.5 m	0.1 m	10.2°	4.69°
Feature at 5 m	1.02 cm	0.44 cm	0.6 m	0.2 m	5.9°	2.64°

Table 5.2: AR scores for Smart Home AR application after 1 minute-drift on z-axis

We updated Table 4.4 from Section 4.4 to match with the setup proposed here. Table 5.1 exhibits results when user just used the two finger swipe ($f_{att} = 0°$) and Table 5.2 shows results after 1 minute without swipe ($f_{att} = 5°$). Compared to Table 4.4, we noticed a good improvement in rendering (e) when distance of a feature is greater than 1 m. This show us how the two finger swipe enhances the overall system. A video of the system is available [11].

Venturi - Y3 Demo repackaged

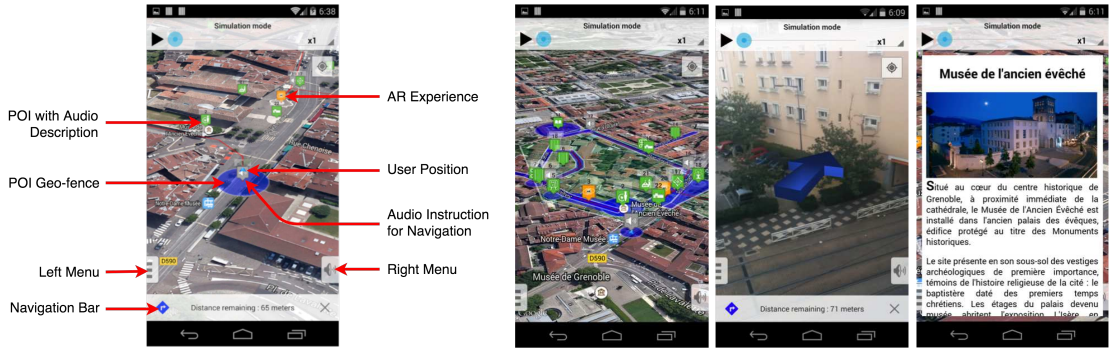


Figure 5.14: Screenshots of Venturi Y3 application

The last application is an AR navigator specifically designed for pedestrians. This application has been developed during the Venturi FP7 (2011-2015) project but we chose to present it because it has been repackaged with our AR framework since then. Between two visually driven AR experiences (at the time, developed by partners), the navigator provides the user with an audio and visual guidance through a pre-defined touristic path in Grenoble. The position of the user is obtained through a fusion of GPS signal (when available), pedometer estimates and a map-matching algorithm exploiting OpenStreetMap. As the GPS signal is poor in several parts of the old city the integration of the pedometer enables the navigator to obtain a sufficiently reliable position estimate, crucial for AR applications and geofencing. Within the application, there are several options given to the user to view the navigation path through the city, ranging from a satellite image of the streets to a vector map. In the navigation pane, the geofences relating to the AR experiences and other points of interest can be seen. Screenshots of the application are given in Figure 5.14 and authoring has been carried out thanks to our authoring tool (Figure 5.2). A

video of the system is available[12].

5.4 Conclusions and Perspectives

In this chapter, we introduced a group of tools for the Geo AR. We developed and commented our approach from authoring to rendering. We proposed a lightweight format for Geo AR documents which is based on OSM XML specifications. Content creation is manageable by JOSM (Java editor for OSM) in the same manner than classical geographic information thanks to the map paint style dedicated to AR we developed. This allows authors to reuse entities from OpenStreetMap (potentially more than 122 millions[13]) and to create new ones. Then, we detailed the process of how features and medias in the AR scene are rendered in function of: the device attitude, the device position and the field of view of the device camera. Finally, we presented our AR framework based on geolocation. We exhibited three applications we built with our AR framework: TyrAR (outdoor, naming mountains and cities), Smart Home AR (indoor, control features in an apartment), Venturi - Y3 Demo repackaged (outdoor, city tour).

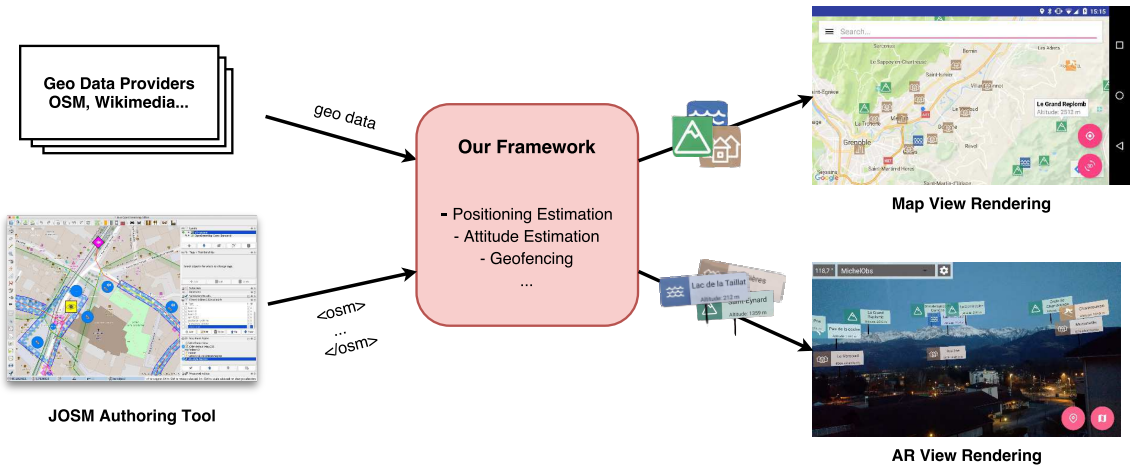


Figure 5.15: Our AR framework at the center of the process for building Geo AR applications.

In the AR framework classification provided by Billinghurst et al. in [Billinghurst et al., 2015], our work could be placed both: at *low-level software library* because rendering is customizable with Rajawali (although we provide helpers) and *stand-alone AR authoring tools* thanks to the documents generated by JOSM. To summarize: the developer needs to define how features in our AR document have to be rendered on the map view and the AR view (see Figure 5.15). In Table 5.3, we also provide an updated list of AR frameworks from the state of the art including ours.

The most important perspective we would like to explore is to handle occlusions in our virtual scene. Handling occlusion consists in hiding virtual surfaces which should not be visible in the reality (e.g hide a feature behind a wall). This could be possible indoor thanks to the Rajawali

[12] `https://www.youtube.com/watch?v=iqoIKA_1MgE` (VENTURI: Third year AR technology demonstrator)

[13] `https://taginfo.openstreetmap.org/reports/database_statistics`

	Rendering	AR Format	Authoring Tool	Open-Source	Platforms	Development Years
android-augment-reality-framework	2D Views	Hard coded	No	Yes	Android	2011-2014
Argon	three.js (WebGL)	KARML	Proprietary	Mainly	JS	2009-2017
BeyondAR	OpenGL (images, labels)	Hard coded	No	Yes	Android	2013-2014
DroidAR	OpenGL (3D models, basic objects)	Hard coded	No	Yes (v1)	Android	2010-2013
Mixare	2D Views	Hard coded	No	Yes	Android & iOS	2010-2012
PanicAR	2D Views	Hard coded	No	No	Android & iOS	2011-2014
Contribution	**2D Views & 3D Views (Rajawali)**	**OSM based**	**JOSM**	**Not yet**	**Android**	**2014-2017**

Table 5.3: Updated list of Geo AR frameworks with our contribution

library coupled with building walls (that we already used) and outdoor thanks to a digital elevation model like SRTM.

General Conclusions & Perspectives

We now summarize the advances we realized. In a first part, we conclude on our contributions. In a second part, we present a list of perspectives and future works that our thesis has led to. In a third part, we recap the various projects that we made available online. Finally, we present the summary of our publications.

Contributions

In this manuscript, we focused our work on the study, the evaluation and the improvement of augmented reality based on geolocation. For this purpose, we started by studying the two components that make a Geo AR system accurate: attitude estimation and positioning estimation.

In Chapter 2, we have investigated the use of attitude estimation algorithms in the particular context of pedestrians using commodity smartphones. We have proposed a benchmark for evaluating and comparing the precision of attitude estimations during typical smartphone motions with and without magnetic perturbations. For the first time, our experiments shed light on the relative impacts of calibrations, parameters, noises, bias, motions, magnetic perturbations, and sampling rates when estimating attitude on smartphones. We went further in the study in the particular context of attitude estimation during AR motions. An online tool based on the benchmarks has been released in order to help developers in choosing the right filter and appropriate parameter values in function of the expected motions, device, and magnetic perturbations. In all cases, we recommend developers to use custom calibration and algorithms in replacement of those provided by smartphone's OS. We proposed a new parallel filtering technique for limiting the impact of magnetic perturbations with any attitude estimation algorithm used in this context. The principle is that we reprocess the filter for a few last seconds without magnetometer measurements. When the detection occurs, attitude estimation is immediately replaced by these values. Our algorithm "*MichelObsF*" which is based on this new technique provides significant gains in precision when estimating attitude in the presence of magnetic perturbations. In the absence of magnetic perturbations, it offers the same precision than the most precise algorithms.

In Chapter 3, we have investigated the use of positioning estimation algorithms in the particular context of pedestrians using commodity smartphones. We have proposed a tool GTR4SL (Ground Truth Recorder for Sensor Localization) to create experimental protocols for evaluating and comparing the precision of navigation algorithms during typical smartphone motions. We have used this tool to benchmark several techniques we presented in Chapter 1: WiFi Fingerprinting, WiFi Trilateration, SHS, SHS + Map-matching, GNSS and UWB. The results of our evaluation have shown that there is no single technique which outperforms all the others for all use cases. However, some techniques are particularly appropriate with a specific context (outdoor, not fixed,

with map...).

These two studies showed that a multitude of attitude estimation filters and positioning estimation techniques can be used for Geo AR experiences. To demonstrate if these filters and algorithms are relevant for Geo AR, we proposed an evaluation method in Chapter 4. This evaluation method calculates the average distance between a real and a virtual feature represented on the screen of the device. This system takes as input the vectors of errors from attitude estimation and positioning estimation. We have shown that the distance between feature and user has a huge impact on rendering. Finally, we applied our evaluation method to four use cases to know if Geo AR is relevant. It showed us that for the moment, some AR applications cannot be developed with Geo AR (e.g. turn around a 3D model without significant progress on positioning estimation), while others are promising (e.g names of mountains or monitoring object indoor with UWB). From our evaluation method, we noticed that such Geo AR applications are possible to develop.

In Chapter 5, we introduced a group of tools for Geo AR. Then, we developed and commented our approach from authoring to rendering. We proposed a lightweight format for Geo AR documents which is based on OSM XML specifications. Content creation is manageable by JOSM (Java editor for OSM) in the same manner than classical geographic information thanks to the map paint style dedicated to AR we developed. This allows authors to reuse entities from OpenStreetMap (potentially more than 122 millions[14]) and to create new ones. Then, we detailed the process of how features and medias in the AR scene are rendered in function of: the device attitude, the device position and the field of view of the device camera. Finally, we presented our AR framework based on geolocation.

Perspectives

We now present some perspectives for further developments.

Fine-grained AR experiences by fusing vision-based and geolocation-based approaches

In Chapters 4 and 5, we show that our work on attitude estimation and navigation positioning improved the quality of Geo AR. Yet, we also observed the limits of such a system, particularly when features are close to the user in the AR environment 4.4. Furthermore, a technique like SHS does not estimate the position of the smartphone between two steps, consequently, AR experience remains less immersive. AR based on vision, and specifically the SLAM (Simultaneous Localization And Mapping) technique might help in this exercise. The SLAM creates and places features in a geographic frame, which is initialized after few seconds of image processing. However, the transformation (translation and rotation) from known geographic coordinates (WGS84, ECEF...) to the geographic coordinates generated by SLAM is unknown. Consequently, the millions of POIs from geo data providers became useless. The idea of further researches would be to find this transformation by getting the best of the lessons learned from our Geo AR study, and especially with the estimation of positioning and the estimation of attitude.

Some systems deal with a fusion of vision tracking and geolocation sensors [Zhou et al., 2009, Shepard, 2013, Menozzi et al., 2014], but none of them are designed for smartphone use. During the last years, the SLAM technique has improved a lot in terms of precision and computational

[14] `https://taginfo.openstreetmap.org/reports/database_statistics`

cost [Cadena et al., 2016]. Very recently, in June 2017, Apple has shown the robustness[15] of the SLAM technique on smartphones with the release of ARKit[16]. This information enables good perspectives for the fusion of both approaches.

Improving the Rendering Capabilities

Another perspective we would like to explore is to improve the AR viewer with more complex rendering.

For instance, we planed to handle occlusions in our virtual scene. Handling occlusion consists in hiding virtual surfaces which should not be visible in the reality (e.g hiding a feature behind a wall). This could be possible indoor thanks to the Rajawali library coupled with building walls (that we already used) and outdoor thanks to a Digital Elevation Model (DEM) like SRTM.

For the moment, the geographical surfaces (i.e cities boundaries) we display in our AR browser are "flat", they do not follow the elevation of the ground and this can be a problem when surfaces exhibit a high variation in elevation (i.e. a mountain range). Thanks to a DEM like SRTM, it becomes possible to sample the polygon with ground elevations and enhance the rendering of geographical surfaces.

Geo Augmented Reality for UAV

The study we conducted in this manuscript refers to smartphone held by pedestrians. However, the Geo AR approach can be extended to other supports, for instance UAVs (Unmanned Aerials Vehicles). UAVs are equipped by a camera and inertial sensors, this makes Geo AR possible. Moreover, the RapidImaging company[17] experimented it recently (2016) by displaying Google maps streets and POIs as virtual features over the camera of the UAV (see Figure 6.1).

Figure 6.1: Geo AR with a UAV (image from RapidImaging company[17]).

The displacement model of a UAV is well known because the machine is controlled by motors. Moreover, as a UAV flights far enough to buildings, the magnetic perturbations are low. A similar approach of Chapter 2, 3 and 4 adapted to UAVs can exhibits the possibilities of such a system. Another perspective is to adapt our framework to take into account remote sensors and remote video streams to handle Geo AR with UAVs.

[15] https://medium.com/super-ventures-blog/why-is-arkit-better-than-the-alternatives-af8871889d6a

[16] https://developer.apple.com/arkit/

[17] http://www.rapidimaging.net/augmented-reality

External Links

- The detailed results of our attitude estimation experiments are displayed on `http://tyrex.inria.fr/mobile/benchmarks-attitude/`.

- The source code of this benchmark and filters are available online from `https://github.com/tyrex-team/benchmarks-attitude-smartphones`.

- SENSLOGS for Android and iOS are also on the team's *github* at `https://github.com/tyrex-team/senslogs` `https://github.com/tyrex-team/senslogs-ios`.
  Senslogs is also available on the Play Store at `https://play.google.com/store/apps/details?id=fr.inria.tyrex.senslogs`

- The source code of WIFI SCAN INTERVAL for Android is available at `https://github.com/ThibaudM/WifiScanInterval`
  and the application on the Play Store at `https://play.google.com/store/apps/details?id=fr.inria.tyrex.wifiscaninterval`. Statistics are shown at `http://thibaud-michel.com/mobile/wifi-scan-interval.txt`

Related Publications

- **A comparative analysis of attitude estimation for pedestrian navigation with smartphones.**
  Thibaud Michel, Hassen Fourati, Pierre Genevès and Nabil Layaïda.
  International Conference on Indoor Positioning and Indoor Navigation, Oct 2015, Banff, Canada.

- **On attitude estimation with smartphones.**
  Thibaud Michel, Pierre Genevès, Hassen Fourati and Nabil Layaïda.
  IEEE International Conference on Pervasive Computing and Communications, Mar 2017, Kona, U.S.

- **[Submitted] Attitude estimation with smartphones.**
  Thibaud Michel, Pierre Genevès, Hassen Fourati and Nabil Layaïda.
  (Extended version of the above Percom paper.)

Bibliography

[Aboodi and Wan, 2012] Aboodi, A. and Wan, T.-C. (2012). Evaluation of wifi-based indoor (wbi) positioning algorithm. In *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, pages 260–264. IEEE.

[Afzal et al., 2011] Afzal, M. H., Renaudin, V., and Lachapelle, G. (2011). Magnetic field based heading estimation for pedestrian navigation environments. In *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*, pages 1–10. IEEE.

[Al Nuaimi and Kamel, 2011] Al Nuaimi, K. and Kamel, H. (2011). A survey of indoor positioning systems and algorithms. *Innovations in information technology (IIT), 2011 international conference on*, pages 185–190.

[Apple, 2017] Apple (2017). Core motion. `https://developer.apple.com/documentation/coremotion`. [Online; accessed 31-August-2017].

[Bachmann et al., 2004] Bachmann, E. R., Yun, X., and Peterson, C. W. (2004). An investigation of the effects of magnetic variations on inertial/magnetic orientation sensors. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1115–1122. IEEE.

[Bahl and Padmanabhan, 2000] Bahl, P. and Padmanabhan, V. N. (2000). RADAR: An in-building RF-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784. Ieee.

[Bartz, 1987] Bartz, P. (1987). Magnetic compass calibration. US Patent 4,698,912.

[Beder and Klepal, 2012] Beder, C. and Klepal, M. (2012). Fingerprinting based localisation revisited: A rigorous approach for comparing RSSI measurements coping with missed access points and differing antenna attenuations (PDF) - Semantic Scholar. *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on*.

[Belimpasakis et al., 2010] Belimpasakis, P., You, Y., and Selonen, P. (2010). Enabling Rapid Creation of Content for Consumption in Mobile Augmented Reality. In *2010 Fourth International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 1–6.

[Ben-Moshe et al., 2011] Ben-Moshe, B., Elkin, E., Levi, H., and Weissman, A. (2011). Improving accuracy of gnss devices in urban canyons. In *CCCG*, pages 511–515.

[Bernardos Barbolla et al., 2011] Bernardos Barbolla, A. M., Corredera, C., Ramon, J., and Cano García, J. (2011). Mobile Indoor Augmented Reality. Exploring applications in hospitality environments. *International Conference on Pervasive and Embedded Computing and Communication Systems*.

[Bernstein and Kornhauser, 1998] Bernstein, D. and Kornhauser, A. (1998). An introduction to map matching for personal navigation assistants. Technical report, New Jersey Institute of Technology.

[Bhide, 2014] Bhide, S. (2014). Augmented reality view for android. `https://github.com/raweng/augmented-reality-view`.

[Billinghurst et al., 2015] Billinghurst, M., Clark, A., and Lee, G. (2015). A Survey of Augmented Reality. *Foundations and Trends® in Human–Computer Interaction*, 8(2-3):73–272.

[Black, 1964] Black, H. D. (1964). A passive system for determining the attitude of a satellite. *AIAA journal*, 2(7):1350–1351.

[Brajdic and Harle, 2013] Brajdic, A. and Harle, R. (2013). Walk detection and step counting on unconstrained smartphones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and ubiquitous computing*, pages 225–234. ACM.

[Brondi et al., 2012] Brondi, R., Carrozzino, M., Tecchia, F., and Bergamasco, M. (2012). Mobile augmented reality for cultural dissemination. In *Proceedings of 1st International Conference on Information Technologies for Performing Arts, Media Access and Entertainment, Firenze, Italy*, pages 113–117.

[Cadena et al., 2016] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332.

[Card et al., 1986] Card, S. K., Moran, T. P., and Newell, A. (1986). An engineering model of human performance. *Ergonomics: Psychological mechanisms and models in ergonomics*, 1:35.

[Chen et al., 2015] Chen, Z., Zou, H., Jiang, H., Zhu, Q., Soh, Y. C., and Xie, L. (2015). Fusion of wifi, smartphone sensors and landmarks using the kalman filter for indoor localization. *Sensors*, 15(1):715–732.

[Choukroun et al., 2006] Choukroun, D., Bar-Itzhack, I. Y., and Oshman, Y. (2006). Novel quaternion kalman filter. *Aerospace and Electronic Systems, IEEE Transactions on*, 42(1):174–190.

[Combettes and Renaudin, 2015] Combettes, C. and Renaudin, V. (2015). Comparison of Misalignment Estimation Techniques Between Handheld Device and Walking Directions. *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*.

[Cox et al., 2002] Cox, S., Cuthbert, A., Daisey, P., Davidson, J., Johnson, S., Keighan, E., Lake, R., Mabrouk, M., Margoulies, S., Martell, R., et al. (2002). Opengis® geography markup language (gml) implementation specification, version.

[Crassidis and Markley, 2003] Crassidis, J. L. and Markley, F. L. (2003). Unscented filtering for spacecraft attitude estimation. *Journal of guidance, control, and dynamics*, 26(4):536–542.

[Crassidis et al., 2007] Crassidis, J. L., Markley, F. L., and Cheng, Y. (2007). Survey of nonlinear attitude estimation methods. *Journal of Guidance, Control, and Dynamics*, 30(1):12–28.

[Curran et al., 2011] Curran, K., Furey, E., Lunney, T., Santos, J., Woods, D., and McCaughey, A. (2011). An evaluation of indoor location determination technologies. *Journal of Location Based Services*, 5(2):61–78.

[Diebel, 2006] Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58:15–16.

[DüNser et al., 2012] DüNser, A., Billinghurst, M., Wen, J., Lehtinen, V., and Nurminen, A. (2012). Exploring the use of handheld ar for outdoor navigation. *Computers & Graphics*, 36(8):1084–1095.

[Dünser et al., 2008] Dünser, A., Grasset, R., and Billinghurst, M. (2008). *A survey of evaluation techniques used in augmented reality studies*. Human Interface Technology Laboratory New Zealand.

[Dünser et al., 2012] Dünser, A., Billinghurst, M., Wen, J., Lehtinen, V., and Nurminen, A. (2012). Exploring the use of handheld AR for outdoor navigation. *Computers & Graphics*, 36(8):1084–1095.

[Ebner et al., 2015] Ebner, F., Fetzer, T., Deinzer, F., Köping, L., and Grzegorzek, M. (2015). Multi sensor 3d indoor localisation. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*, pages 1–11. IEEE.

[Euston et al., 2008] Euston, M., Coote, P., Mahony, R., Kim, J., and Hamel, T. (2008). A complementary filter for attitude estimation of a fixed-wing uav. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 340–345. IEEE.

[Fairhurst, 2015] Fairhurst, R. (2015). Tilemaker. `https://github.com/systemed/tilemaker`.

[Faragher and Harle, 2014] Faragher, R. and Harle, R. (2014). An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+'14)*, pages 201–210.

[Feiner et al., 1997] Feiner, S., MacIntyre, B., Hollerer, T., and Webster, A. (1997). A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. In *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 74–81. IEEE.

[Fourati, 2015] Fourati, H. (2015). Heterogeneous data fusion algorithm for pedestrian navigation via foot-mounted inertial measurement unit and complementary filter. *Instrumentation and Measurement, IEEE Transactions on*, 64(1):221–229.

[Fourati et al., 2011] Fourati, H., Manamanni, N., Afilal, L., and Handrich, Y. (2011). A nonlinear filtering approach for the attitude and dynamic body acceleration estimation based on inertial and magnetic sensors: Bio-logging application. *Sensors Journal, IEEE*, 11(1):233–244.

[Frosio et al., 2013] Frosio, I., Pedersini, F., and Borghese, N. A. (2013). Autocalibration of MEMS accelerometers. In *Advanced Mechatronics and MEMS Devices*, pages 53–88. Springer.

[Geiger et al., 2014] Geiger, P., Schickler, M., Pryss, R., Schobel, J., and Reichert, M. (2014). Location-based mobile augmented reality applications: Challenges, examples, lessons learned. *Int'l Conference on Web Information Systems and Technologies, Special Session on Business Apps*.

[Google, 2017] Google (2017). Sensors overview. `https://developer.android.com/guide/topics/sensors/sensors_overview.html`. [Online; accessed 31-August-2017].

[Guimarães et al., 2016] Guimarães, V., Castro, L., Carneiro, S., Monteiro, M., Rocha, T., Barandas, M., Machado, J., Vasconcelos, M., Gamboa, H., and Elias, D. (2016). A motion tracking solution for indoor localization using smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on*, pages 1–8. IEEE.

[Hansson and Tufvesson, 2011] Hansson, A. and Tufvesson, L. (2011). *Using Sensor Equipped Smartphones to Localize WiFi Access Points*. Department of Automatic Control, Lund University, 2011.

[Harada et al., 2004] Harada, T., Uchino, H., Mori, T., and Sato, T. (2004). Portable absolute orientation estimation device with wireless network under accelerated situation. In *International Conference on Robotics and Automation*, volume 2, pages 1412–1417. IEEE.

[Harle, 2013] Harle, R. (2013). A survey of indoor inertial positioning systems for pedestrians. *IEEE Communications Surveys and Tutorials*, 15(3):1281–1293.

[He and Chan, 2016] He, S. and Chan, S.-H. G. (2016). Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons. *IEEE Communications Surveys & Tutorials*, 18(1):466–490.

[Heinen, 2014] Heinen, S. (2014). Droidar. `https://github.com/simon-heinen/droidar`.

[Hou, 2004] Hou, H. (2004). *Modeling inertial sensors errors using Allan variance*. University of Calgary, Department of Geomatics Engineering.

[Huynh, 2009] Huynh, D. Q. (2009). Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164.

[Ichikari et al., 2017] Ichikari, R., Kurata, T., Makita, K., Taketomi, T., Uchiyama, H., Kondo, T., Mori, S., and Shibata, F. (2017). Reference Framework on vSRT-method Benchmarking for MAR. In Lindeman, R. W., Bruder, G., and Iwai, D., editors, *ICAT-EGVE 2017 - International Conference on Artificial Reality and Telexistence and Eurographics Symposium on Virtual Environments*. The Eurographics Association.

[Innoventions, Inc, 2017] Innoventions, Inc (2017). Sensor Kinetics.

[ISO 18305, 2016] ISO 18305 (2016). Information technology – Real time locating systems – Test and evaluation of localization and tracking systems. Standard, International Organization for Standardization.

[ISO 18520, 2017] ISO 18520 (2017). Information technology — Computer graphics, image processing and environmental data representation — Benchmarking of vison-based spatial registration and tracking methods for Mixed and Augmented Reality (MAR). Standard, International Organization for Standardization.

[Jahn et al., 2010] Jahn, J., Batzer, U., Seitz, J., Patino-Studencka, L., and Gutiérrez Boronat, J. (2010). Comparison and evaluation of acceleration based step length estimators for handheld devices. In *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, pages 1–6. IEEE.

[Karimi, 2013] Karimi, H. A. (2013). *Advanced Location-Based Technologies and Services*. CRC Press.

[Khoury and Kamat, 2009] Khoury, H. M. and Kamat, V. R. (2009). Evaluation of position tracking technologies for user localization in indoor construction environments. *Automation in Construction*, 18(4):444–457.

[Kim and Kim, 2001] Kim, S. and Kim, J.-H. (2001). Adaptive fuzzy-network-based c-measure map-matching algorithm for car navigation system. *IEEE Transactions on industrial electronics*, 48(2):432–441.

[Kounavis et al., 2012] Kounavis, C. D., Kasimati, A. E., and Zamani, E. D. (2012). Enhancing the Tourism Experience through Mobile Augmented Reality: Challenges and Prospects. *International Journal of Engineering Business Management*, 4:10.

[Kourogi and Kurata, 2003a] Kourogi, M. and Kurata, T. (2003a). A method of personal positioning based on sensor data fusion of wearable camera and self-contained sensors. In *Multisensor Fusion and Integration for Intelligent Systems, MFI2003. Proceedings of IEEE International Conference on*, pages 287–292. IEEE.

[Kourogi and Kurata, 2003b] Kourogi, M. and Kurata, T. (2003b). Personal positioning based on walking locomotion analysis with self-contained sensors and a wearable camera. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, page 103. IEEE Computer Society.

[Koyuncu and Yang, 2010] Koyuncu, H. and Yang, S. H. (2010). A survey of indoor positioning and object locating systems. *IJCSNS International Journal of Computer Science and Network Security*, 10(5):121–128.

[Kuipers, 1999] Kuipers, J. B. (1999). *Quaternions and rotation sequences*, volume 66. Princeton University Press, 2002.

[Kuo et al., 2014] Kuo, Y.-S., Pannuto, P., Hsiao, K.-J., and Dutta, P. (2014). Luxapose: Indoor positioning with mobile phones and visible light. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 447–458. ACM.

[Ladetto, 2000] Ladetto, Q. (2000). On foot navigation: continuous step calibration using both complementary recursive prediction and adaptive kalman filtering. In *Proceedings of ION GPS*, volume 2000, pages 1735–1740.

[Lakmal and Samarabandu, 2016] Lakmal, H. D. R. and Samarabandu, J. (2016). Novel velocity model to improve indoor localization using inertial navigation with sensors on a smartphone. In *Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on*, pages 1–4. IEEE.

[Lechner, 2015] Lechner, M. (2015). Augmented reality markup language 2.0 (arml 2.0). Encoding standard, OGC. `https://portal.opengeospatial.org/files/?artifact_id=62131`.

[Lee et al., 2012a] Lee, G. A., Dünser, A., Kim, S., and Billinghurst, M. (2012a). Cityviewar: A mobile outdoor ar application for city visualization. In *Mixed and Augmented Reality (ISMAR-AMH), 2012 IEEE International Symposium on*, pages 57–64. IEEE.

[Lee et al., 2012b] Lee, J. K., Park, E. J., and Robinovitch, S. N. (2012b). Estimation of attitude and external acceleration using inertial sensor measurement during various dynamic conditions. *IEEE transactions on instrumentation and measurement*, 61(8):2262–2273.

[Lefferts et al., 1982] Lefferts, E. J., Markley, F. L., and Shuster, M. D. (1982). Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429.

[Li et al., 2012] Li, B., Gallagher, T., Dempster, A. G., and Rizos, C. (2012). How feasible is the use of magnetic field alone for indoor positioning? In *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on*, pages 1–9. IEEE.

[Li and Wang, 2013] Li, W. and Wang, J. (2013). Effective adaptive kalman filter for mems-imu/magnetometers integrated attitude and heading reference systems. *Journal of Navigation*, 66(01):99–113.

[Link et al., 2011] Link, J. A. B., Smith, P., Viol, N., and Wehrle, K. (2011). Footpath: Accurate map-based indoor navigation using smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*, pages 1–8. IEEE.

[Lymberopoulos et al., 2015] Lymberopoulos, D., Liu, J., Yang, X., Choudhury, R. R., Handziski, V., and Sen, S. (2015). A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned. In *Proceedings of the 14th international conference on information processing in sensor networks*, pages 178–189. ACM.

[MacIntyre, 2017] MacIntyre, B. (2017). argon-aframe. `https://github.com/argonjs/argon-aframe`.

[MacIntyre et al., 2011] MacIntyre, B., Hill, A., Rouzati, H., Gandy, M., and Davidson, B. (2011). The Argon AR Web Browser and standards-based AR application environment. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 65–74.

[Madden, 2011] Madden, L. (2011). *Professional Augmented Reality Browsers for Smartphones: Programming for junaio, Layar and Wikitude*. John Wiley & Sons. Google-Books-ID: gRpH49rlVRsC.

[Madgwick et al., 2011] Madgwick, S. O., Harrison, A. J., and Vaidyanathan, R. (2011). Estimation of IMU and MARG orientation using a gradient descent algorithm. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pages 1–7. IEEE.

[Mahony et al., 2008] Mahony, R., Hamel, T., and Pflimlin, J.-M. (2008). Nonlinear complementary filters on the special orthogonal group. *Automatic Control, IEEE Transactions on*, 53(5):1203–1218.

[Makni et al., 2014] Makni, A., Fourati, H., and Kibangou, A. Y. (2014). Adaptive kalman filter for mems-imu based attitude estimation under external acceleration and parsimonious use of gyroscopes. In *Control Conference (ECC), 2014 European*, pages 1379–1384. IEEE.

[Marins et al., 2001] Marins, J. L., Yun, X., Bachmann, E. R., McGhee, R. B., and Zyda, M. J. (2001). An extended kalman filter for quaternion-based orientation estimation using marg sensors. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 4, pages 2003–2011. IEEE.

[Markley, 1988] Markley, F. L. (1988). Attitude determination using vector observations and the singular value decomposition. *The Journal of the Astronautical Sciences*, 36(3):245–258.

[Markley and Mortari, 2000] Markley, F. L. and Mortari, D. (2000). Quaternion attitude estimation using vector observations. *Journal of the Astronautical Sciences*, 48(2):359–380.

[Martin and Salaün, 2010] Martin, P. and Salaün, E. (2010). Design and implementation of a low-cost observer-based attitude and heading reference system. *Control Engineering Practice*, 18(7):712–722.

[Masiero et al., 2014] Masiero, A., Guarnieri, A., Pirotti, F., and Vettore, A. (2014). A particle filter for smartphone-based indoor pedestrian navigation. *Micromachines*, 5(4):1012–1033.

[Matuszka et al., 2014] Matuszka, T., Kámán, S., and Kiss, A. (2014). A Semantically Enriched Augmented Reality Browser. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Kobsa, A., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Terzopoulos, D., Tygar, D., Weikum, G., Shumaker, R., and Lackey, S., editors, *Virtual, Augmented and Mixed Reality. Designing and Developing Virtual and Augmented Environments*, volume 8525, pages 375–384. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-07458-0_35.

[Mautz, 2012] Mautz, R. (2012). *Indoor positioning technologies*. PhD thesis, Habilitationsschrift ETH Zürich.

[Menozzi et al., 2014] Menozzi, A., Clipp, B., Wenger, E., Heinly, J., Dunn, E., Towles, H., Frahm, J. M., and Welch, G. (2014). Development of vision-aided navigation for a wearable outdoor augmented reality system. In *2014 IEEE/ION Position, Location and Navigation Symposium - PLANS 2014*, pages 460–472.

[Messelodi et al., 2015] Messelodi, S., Modena, C. M., Porzi, L., and Chippendale, P. (2015). i-street: Detection, identification, augmentation of street plates in a touristic mobile application. In *International Conference on Image Analysis and Processing*, pages 194–204. Springer International Publishing.

[Michel et al., 2015] Michel, T., Fourati, H., Genevès, P., and Layaïda, N. (2015). A comparative analysis of attitude estimation for pedestrian navigation with smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*, pages 1–10. IEEE.

[Moayeri et al., 2016] Moayeri, N., Ergin, M. O., Lemic, F., Handziski, V., and Wolisz, A. (2016). Perfloc (part 1): An extensive data repository for development of smartphone indoor localization apps. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*, pages 1–7. IEEE.

[Munguia and Grau, 2011] Munguia, R. and Grau, A. (2011). Attitude and heading system based on ekf total state configuration. In *2011 IEEE International Symposium on Industrial Electronics*, pages 2147–2152. IEEE.

[N Dev Group, 2017] N Dev Group (2017). Advanced Sensor Recorder.

[National Coordination Office for Space-Based Positioning, Navigation, and Timing, 2017] National Coordination Office for Space-Based Positioning, Navigation, and Timing (2017). How accurate is gps? `http://www.gps.gov/systems/gps/performance/accuracy/`. [Online; accessed 31-August-2017].

[(NGA) and the U.K.'s Defence Geographic Centre (DGC), 2015] (NGA), U. and the U.K.'s Defence Geographic Centre (DGC) (2015). The world magnetic model. `http://www.ngdc.noaa.gov/geomag/WMM`. [Online; accessed 17-July-2015].

[Nguyen, 2017] Nguyen, D. (2017). Ar location-based for android. `https://github.com/dat-ng/ar-location-based-android`.

[Nicholls and Powertech, 2013] Nicholls, G. and Powertech, I. S. T. (2013). Using augmented reality as an extension to utility GIS. *IST Powertech journal*, pages 62–64.

[Ojeda and Borenstein, 2002] Ojeda, L. and Borenstein, J. (2002). Flexnav: Fuzzy logi c expert rule-based position estimation for mobile robots on rugged terrain. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 317–322. IEEE.

[Oshman and Carmi, 2006] Oshman, Y. and Carmi, A. (2006). Attitude estimation from vector observations using a genetic-algorithm-embedded quaternion particle filter. *Journal of Guidance, Control, and Dynamics*, 29(4):879–891.

[Panel, 2014] Panel, T. (2014). Introduction of the trackmark activities. `http://ypcex.naist.jp/trakmark/ismar14/pdf/TrakMarkPanel.pdf`. [Online; accessed 31-August-2017].

[Peterson et al., 1997] Peterson, B., Bruckner, D., and Heye, S. (1997). Measuring gps signals indoors. In *International Association of Institutes of Navigation. World congress*.

[Piekarski and Thomas, 2002] Piekarski, W. and Thomas, B. (2002). Arquake: the outdoor augmented reality gaming system. *Communications of the ACM*, 45(1):36–38.

[Pryss et al., 2016] Pryss, R., Geiger, P., Schickler, M., Schobel, J., and Reichert, M. (2016). Advanced Algorithms for Location-Based Smart Mobile Augmented Reality Applications. *Procedia Computer Science*, 94:97–104.

[Puig Sanz, 2017] Puig Sanz, J. (2017). beyondar: Augmented Reality framework for Android based on geolocalization (GPS). original-date: 2013-07-25T20:58:56Z.

[Rappaport et al., 1996] Rappaport, T. S. et al. (1996). *Wireless communications: principles and practice*, volume 2. prentice hall PTR New Jersey.

[Rehbinder and Hu, 2004] Rehbinder, H. and Hu, X. (2004). Drift-free attitude estimation for accelerated rigid bodies. *Automatica*, 40(4):653–659.

[Renaudin and Combettes, 2014] Renaudin, V. and Combettes, C. (2014). Magnetic, acceleration fields and gyroscope quaternion (MAGYQ)-based attitude estimation with smartphone sensors for indoor pedestrian navigation. *Sensors*, 14(12):22864–22890.

[Renaudin et al., 2012] Renaudin, V., Susi, M., and Lachapelle, G. (2012). Step length estimation using handheld inertial sensors. *Sensors*, 12(7):8507–8525.

[Sabatini, 2006] Sabatini, A. (2006). Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing. *Biomedical Engineering, IEEE Transactions on*, 53(7):1346–1356.

[Sachs, 2010] Sachs, D. (2010). Sensor fusion on android devices: A revolution in motion processing. [Video] `https://www.youtube.com/watch?v=C7JQ7Rpwn2k`. [Online; accessed 9-April-2015].

[Scioscia et al., 2014] Scioscia, F., Binetti, M., Ruta, M., Ieva, S., and Di Sciascio, E. (2014). A Framework and a Tool for Semantic Annotation of POIs in OpenStreetMap. *Procedia - Social and Behavioral Sciences*, 111:1092–1101.

[Series, 2012] Series, P. (2012). Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 900 mhz to 100 ghz. *ICU: Genève, Switzerland*.

[Shepard, 2013] Shepard, D. P. (2013). *Fusion of carrier-phase differential GPS, bundle-adjustment-based visual SLAM, and inertial navigation for precisely and globally-registered augmented reality*. PhD thesis, The University of Texas at Austin.

[Shepperd, 1978] Shepperd, S. W. (1978). Quaternion from rotation matrix.[four-parameter representation of coordinate transformation matrix]. *Journal of Guidance and Control; 1; May-June*.

[Shoemake, 1985] Shoemake, K. (1985). Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM.

[Shuster et al., 1982] Shuster, M., Lefferts, E., and Markley, F. (1982). Kalman filtering for spacecraft attitude estimation. In *AIAA 20th Aerospace Sciences Meeting, Orlando, Florida*, volume 232.

[Shuster and Oh, 1981] Shuster, M. D. and Oh, S. (1981). Three-axis attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 4(1):70–77.

[Speiginer et al., 2015] Speiginer, G., MacIntyre, B., Bolter, J., Rouzati, H., Lambeth, A., Levy, L., Baird, L., Gandy, M., Sanders, M., Davidson, B., et al. (2015). The evolution of the argon web framework through its use creating cultural heritage and community–based augmented reality applications. In *International Conference on Human-Computer Interaction*, pages 112–124. Springer, Cham.

[Suh, 2010] Suh, Y. S. (2010). Orientation estimation using a quaternion-based indirect kalman filter with adaptive estimation of external acceleration. *IEEE Transactions on Instrumentation and Measurement*, 59(12):3296–3305.

[Susi et al., 2013] Susi, M., Renaudin, V., and Lachapelle, G. (2013). Motion mode recognition and step detection algorithms for mobile phone users. *Sensors*, 13(2):1539–1562.

[Sutherland, 1968] Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764. ACM.

[Thrun et al., 2001] Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2001). Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141.

[Torres-Sospedra et al., 2014] Torres-Sospedra, J., Montoliu, R., Martínez-Usó, A., Avariento, J. P., Arnau, T. J., Benedito-Bordonau, M., and Huerta, J. (2014). UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems. In *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 261–270.

[Valenti et al., 2016] Valenti, R. G., Dryanovski, I., and Xiao, J. (2016). A linear kalman filter for marg orientation estimation using the algebraic quaternion algorithm. *IEEE Transactions on Instrumentation and Measurement*, 65(2):467–481.

[Van Haute et al., 2015] Van Haute, T., De Poorter, E., Lemic, F., Handziski, V., Wirström, N., Voigt, T., Wolisz, A., and Moerman, I. (2015). Platform for benchmarking of rf-based indoor localization solutions. *IEEE Communications Magazine*, 53(9):126–133.

[Van Haute et al., 2013] Van Haute, T., De Poorter, E., Rossey, J., Moerman, I., Handziski, V., Behboodi, A., Lemic, F., Wolisz, A., Wirström, N., Voigt, T., et al. (2013). The evarilos benchmarking handbook: Evaluation of rf-based indoor localization solutions. In *2nd International Workshop on Measurement-based Experimental Research, Methodology and Tools*.

[Wahba, 1965] Wahba, G. (1965). A least squares estimate of satellite attitude. *SIAM review*, 7(3):409–409.

[Windows, 2017] Windows (2017). Sensor and location platform. `https://msdn.microsoft.com/en-us/library/windows/hardware/dn614612(v=vs.85).aspx`. [Online; accessed 31-August-2017].

[Woodman, 2007] Woodman, O. J. (2007). An introduction to inertial navigation. Technical report, University of Cambridge, Computer Laboratory.

[Yohan et al., 2000] Yohan, S. J., Julier, S., Baillot, Y., Lanzagorta, M., Brown, D., and Rosenblum, L. (2000). Bars: Battlefield augmented reality system. In *In NATO Symposium on Information Processing Techniques for Military Systems*, pages 9–11.

[Yun et al., 2012] Yun, X., Calusdian, J., Bachmann, E., and McGhee, R. (2012). Estimation of Human Foot Motion During Normal Walking Using Inertial and Magnetic Sensor Measurements. *IEEE Transactions on Instrumentation and Measurement*, 61(7):2059–2072.

[Zhang et al., 2008] Zhang, X., Li, Y., Mumford, P., and Rizos, C. (2008). Allan variance analysis on error characters of MEMS inertial sensors for an FPGA-based GPS/INS system. In *Proceedings of the International Symposium on GPS/GNNS*, pages 127–133.

[Zhou et al., 2009] Zhou, Z., Karlekar, J., Hii, D., Schneider, M., Lu, W., and Wittkopf, S. (2009). Robust pose estimation for outdoor mixed reality with sensor fusion. *Universal Access in Human-Computer Interaction. Applications and Services*, pages 281–289.

[Zhu et al., 2007] Zhu, R., Sun, D., Zhou, Z., and Wang, D. (2007). A linear fusion algorithm for attitude determination using low cost mems-based sensors. *Measurement*, 40(3):322–328.

Appendices

Appendix A

Contributions

A.1 Pseudo-code of our SHS approach

Data:
GRAVITY is the standard gravity, fixed at $9.8\ m.s^{-2}$
MIN_INTERVAL_TIME is minimum time between two steps, fixed at $0.5\ s$
THRESHOLD is the threshold to detect a step, fixed at $1.5\ m.s^{-2}$ **A** is a constant fixed at $0.45\ m$ **B** is a constant fixed at $0.2\ m$

```
isAlreadyOver = false;
timestampPreviousStep = -MIN_INTERVAL_TIME;
position = new 2DPosition();

Function void init(wgs84Position)
    position = wgs84ToSphericalMercator(wgs84Position);
end

Function void onNewAccelerationData(timestamp, accX, accY, accZ)
    verticalAcc = abs(accZ - GRAVITY);
    timeInterval = timestamp - timestampPreviousStep;
    if verticalAcc > THRESHOLD && !isAlreadyOver && timeInterval > MIN_INTERVAL_TIME then
        newStepDected(timestamp);
        timestampPreviousStep = timestamp;
        isAlreadyOver = true;
    else if verticalAcc < THRESHOLD && isAlreadyOver then
        isAlreadyOver = false;
    end
end

Function void newStepDected(timestamp)
    freq = 1/(timestamp - timestampPreviousStep);
    freq = min(max(freq, 1), 4);
    stepSize = A + B * freq;
    computeNextStep(stepSize);
end

Function void computeNextStep(stepSize)
    heading = SensorManager.getYaw();
    utmPosition.x = utmPosition.x + stepLength * cos(heading);
    utmPosition.y = utmPosition.y + stepLength * sin(heading);
end
```

A.2 Description of Outputs from Senslogs App

```
Sensors logs
Date: Jan 4, 2016 9:55:47 AM
Device: Nexus 5
Sensors Recorded: 13

== wifi.txt ==
For more information: http://developer.android.com/reference/android/net/wifi/ScanResult
    .html
timestamp BSSID SSID frequency(MHz) level(dBm) capabilities
 Capabilities describes the authentication, key management, and encryption schemes
     supported by the access point.

== magnetometer.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x y z estimated-bias-x estimated-bias-y estimated-bias-z
 All values are in micro-Tesla (uT)
 Factory calibration and temperature compensation are applied to this measurement

== step-detector.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp A sensor of this type triggers an event each time a step is taken by the user.
     The only allowed value to return is 1.0 and an event is generated for each step.
    Like with any other event, the timestamp indicates when the event (here the step)
    occurred, this corresponds to when the foot hit the ground, generating a high
    variation in acceleration.

== magnetometer-calibrated.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x y z
 All values are in micro-Tesla (uT)

== accelerometer.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x y z
 All values are in SI units (m/s^2)

== rotation-vector.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x*sin(θ/2) y*sin(θ/2) z*sin(θ/2) [cos(θ/2)] [estimated-heading-accuracy(
    radians)]
 The rotation vector represents the orientation of the device as a combination of an
     angle and an axis, in which the device has rotated through an angle θ around an
     axis <x, y, z>.

 The reference coordinate system is defined as a direct orthonormal basis, where:
 - X is defined as the vector product Y.Z (It is tangential to the ground at the device'
     s current location and roughly points East).
 - Y is tangential to the ground at the device's current location and points towards
     magnetic north.
 - Z points towards the sky and is perpendicular to the ground.

== linear-acceleration.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x y z
 All values are in SI units (m/s^2)
```

```
== location-wifi-cells.txt ==
For more information: http://developer.android.com/reference/android/location/
    LocationManager.html
timestamp latitude(degrees) longitude(degrees) altitude(meters) bearing(degrees)
    accuracy(meters) speed(meters/second)
 This provider determines location based on availability of cell tower and WiFi access
     points. Results are retrieved by means of a network lookup.

== gyroscope-calibrated.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x y z
 All values are in radians/second

== reference-timestamps.txt ==
timestamp id name Reference position clicked.

== gyroscope.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp x y z estimated-drift-x estimated-drift-y estimated-drift-z
 All values are in radians/second
 Factory calibration and temperature compensation is still applied to the rate of
     rotation (angular speeds).

== step-counter.txt ==
For more information: http://developer.android.com/guide/topics/sensors/sensors_overview
    .html
timestamp A sensor of this type returns the number of steps taken by the user since the
    last reboot while activated.

== nmea.txt ==
For more information: http://developer.android.com/reference/android/location/GpsStatus.
    NmeaListener.html
timestamp sentence NMEA sentences from the GPS.
 NMEA 0183 is a standard for communicating with marine electronic devices and is a
     common method for receiving data from a GPS, typically over a serial port.
```

A.3 Source Code of the JOSM Map Paint Style for Augmented Reality

```
/*
This map style file is dedicated to AR.
Version 0.3
*/


/* Feature */
node[feature=yes] { icon-image: "ar/poi.png"; icon-width:64; icon-height:64; }
area[feature=yes] { icon-image: "ar/bg/poi.png"; color:blue; fill-opacity:0.5; width:2;
    }

/* Feature with Audio */
node[feature=yes][audio] { icon-image: "ar/poi-audio.png"; icon-width:64; icon-height
    :64; }
area[feature=yes][audio] { fill-image: "ar/bg/poi-audio.png"; }

/* Feature with WebPage */
node[feature=yes][/^web-local-page$|^wikipedia$/] { icon-image: "ar/poi-url.png"; icon-
    width:64; icon-height:64; }
area[feature=yes][/^web-local-page$|^wikipedia$/] { fill-image: "ar/bg/poi-url.png"; }

/* Feature with Image */
node[feature=yes][image] { icon-image: "ar/poi-image.png"; icon-width:64; icon-height
    :64; }
area[feature=yes][image] { fill-image: "ar/bg/poi-image.png"; }

/* Feature with Video */
node[feature=yes][video] { icon-image: "ar/poi-video.png"; icon-width:64; icon-height
    :64; }
area[feature=yes][video] { fill-image: "ar/bg/poi-video.png"; }

/* Feature with 3D Model */
node[feature=yes][3dmodel] { icon-image: "ar/poi-3dmodel.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][3dmodel] { fill-image: "ar/bg/poi-3dmodel.png"; }


/* Feature with more than one document */
node[feature=yes][audio][/^web-local-page$|^wikipedia$/] { icon-image: "ar/poi-mixed.png
    "; icon-width:64; icon-height:64; }
area[feature=yes][audio][/^web-local-page$|^wikipedia$/] { fill-image: "ar/bg/poi-mixed.
    png"; }

node[feature=yes][audio][image] { icon-image: "ar/poi-mixed.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][audio][image] { fill-image: "ar/bg/poi-mixed.png"; }

node[feature=yes][audio][video] { icon-image: "ar/poi-mixed.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][audio][video] { fill-image: "ar/bg/poi-mixed.png"; }

node[feature=yes][audio][3dmodel] { icon-image: "ar/poi-mixed.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][audio][3dmodel] { fill-image: "ar/bg/poi-mixed.png"; }

node[feature=yes][/^web-local-page$|^wikipedia$/][image] { icon-image: "ar/poi-mixed.png
    "; icon-width:64; icon-height:64; }
area[feature=yes][/^web-local-page$|^wikipedia$/][image] { fill-image: "ar/bg/poi-mixed.
    png"; }
```

```
node[feature=yes][/^web-local-page$|^wikipedia$/][video] { icon-image: "ar/poi-mixed.png
    "; icon-width:64; icon-height:64; }
area[feature=yes][/^web-local-page$|^wikipedia$/][video] { fill-image: "ar/bg/poi-mixed.
    png"; }

node[feature=yes][/^web-local-page$|^wikipedia$/][3dmodel] { icon-image: "ar/poi-mixed.
    png"; icon-width:64; icon-height:64; }
area[feature=yes][/^web-local-page$|^wikipedia$/][3dmodel] { fill-image: "ar/bg/poi-
    mixed.png"; }

node[feature=yes][image][video] { icon-image: "ar/poi-mixed.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][image][video] { fill-image: "ar/bg/poi-mixed.png"; }

node[feature=yes][image][3dmodel] { icon-image: "ar/poi-mixed.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][image][3dmodel] { fill-image: "ar/bg/poi-mixed.png"; }

node[feature=yes][3dmodel] { icon-image: "ar/poi-3dmodel.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][3dmodel] { fill-image: "ar/bg/poi-3dmodel.png"; }

node[feature=yes][video][3dmodel] { icon-image: "ar/poi-mixed.png"; icon-width:64; icon-
    height:64; }
area[feature=yes][video][3dmodel] { fill-image: "ar/bg/poi-mixed.png"; }
```