UNIVERSITÀ degli STUDI di CATANIA

DIPARTIMENTO DI INGEGNERIA ELETTRICA, ELETTRONICA E INFORMATICA

PhD Dissertation in Computer Engineering and Telecommunications

Cycle XXVIII

PhD Thesis

METHODOLOGIES AND TECHNOLOGIES FOR INDOOR LOCALIZATION: A COMPUTER VISION BASED APPROACH

Candidate

Gaetano Carmelo La Delfa

Catania, 2015

Coordinator

Prof. Dr. V. Carchiolo

Academic Tutor

Prof. Dr. V. Catania

Company Tutor

Ing. M. Turolla

ABSTRACT

The massive diffusion of smartphones we are seeing in the last years, the growing interest for everything related to wearable devices and Internet of Things (IoT), the exponential rise of Location Based Services (LBS: services based on the position of the users and accessible through the device) has meant that technologies capable of determining the position of the user inside a specific context have taken a crucial role in the consumer sector. In outdoor environments, GPS (Global Positioning System) can be considered today as a "standard de facto", while the localization and navigation in indoor environments still remains one of the technological challenges of the next years.

Indoor Positioning Systems (IPS) have a remarkable importance in a lot of important market segments such as the retail sectors for contextual advertising (commercial centers, supermarkets), touristic and transportation sectors (airports, museums), healthcare sectors (hospitals). Sometimes, in emergency situations they can make the difference between life and death. Even if actually doesn't exist a definitive solution as efficient and precise as GPS and with all its advantages, various approaches and methodologies has been proposed in the last

years in scientific literature, and several technologies are appearing into the market. The researches have focused, rather than on realizing a "general purpose" IPS with high performances everywhere, on the development of a variety of solutions suitable for the specific place of deployment and which meet the specifics precision, security, invasiveness and cost requirements.

In the first part of this doctoral dissertation, after an overview on the main methodologies used for locating a user in an indoor environment, we will analyze the state of the art and present some of the most interesting cases. We will focus particularly on approaches which utilize the smartphone's inertial sensors and 2D visual markers based computer vision techniques. In the second part of the dissertation we will propose our own solution to the indoor localization problem. Such solution consists in a visual markers system deployed onto the area of interest's floor and a step detection algorithm. We will compare the performances of three types of markers, taken from the scientific literature, from the point of view of the specifications required by the suggested solution, and choose the most better performing one. We will then propose a client - server architecture for managing the whole process of tracking the user inside a building, and present an implementation of the client, on the iOS platform. Finally in the last part of the dissertation, we will show the obtained results, and talk about the possible future works.

ACKNOWLEDGEMENTS

I would say thanks to prof. Vincenzo Catania for his support and his guide during these three years, and Telecom Italia, for the opportunity it gave me and for its help. I also wish to thank all my colleagues for the fruitful discussions, the advices and also the criticisms, and my friends who have endured me during this period. Finally I would to say thank to my parents, because without their moral and material help nothing of this would have been possible.

CONTENTS

CHAPTER

ONE

INTRODUCTION

During the last 5-10 years two new kinds of technologies - the smartphone, and the World Wide Web - have radically changed our way of live, communicate, have social relationships, interact with the world around us. The ubiquitous internet connectivity, the smartphone which is always with us, with its computation capabilities increasing year by year and with a lot of embedded sensors in every new released device, have made this "object" the perfect gateway between the physical world and the digital world, giving to developers and startups the perfect instruments for creating innovative applications and services that use new paradigms to achieve what we think should be the main purpose of the scientific research: make better the citizen quality of life.

Most of these application and services are strictly related to the position of the user and to the context-related information so become fundamentals - as they are enablers - technologies and solutions for

the user localization. In outdoor environments, the GPS technology (Global Positioning System), integrated nowadays in every kind of smartphone, is almost a standard de facto : it is available in every moment, with an high level of efficiency and precision (around few meters). In indoor environments however it can't be used because, due to the existent physical barriers (roofs, walls etc.) and potential sources of interferences, the signal can't reach the device. There is thus the necessity of finding alternative smartphone-based methodologies for integrating localization and navigation inside buildings and closed spaces.

1.1 Motivations

Motivations which have led me to the development of this project come from various general considerations about the primary role that technology has (and will have) in our life. Smartphone (and the ecosystem around it) in particular, was the main object of my considerations: in fact today, users increasingly rely on its apps to solve the most common problems of daily life such as accessing their bank accounts, looking for the closest restaurant, booking something, reading news. Most of these apps need to know the user position and the information related to the context in which the user is. These apps fall in the Location Based Services category (LBS), provide information and functionality related to the proximity of the device to a specific point of interest (POI) or geographic area, and in the last years they are diffusing in sectors such as tourism, restoration, contextual advertising. In outdoor environments Location Based Services are very common,

especially in the consumer category. They exploit the geographic position, calculated by using the GPS technology, to provide specific functionality for the service. In indoor environments instead, due to the lack of a low cost and better performing localization technology, Location Based Services are not very common. Another category of apps - often working in synergy with LBS - which is recently growing in popularity is the indoor navigators category. Navigators let the user to orientate himself inside an unknown building, reach a particular point of interest, track assets etc. These kind of apps are useful (sometimes fundamentals) in airports, museums, hospitals, emergency situations.

In summary, our analysis has led us to affirm that it is really important, in the actual technological landscape, the study and the development of solutions, possibly accurate, non-invasive and low costs, capable to fill the existing gap between outdoor and indoor localization/navigation systems.

1.2 Objectives

The goal of the present PhD dissertation is the in-depth study of the state of the art for what concern methodologies and technologies for indoor localization, and the implementation of a smartphone-based solution which meets the following criteria:

- **Low Cost:** the system must not require the installation of complex and expensive hardware infrastructures inside the building. In fact, this would lead to a strong resistance in the adoption of the solution, due to the high financial investment and also to

the need of infrastructure maintenance, which should be made by experienced personnel.

- **Low invasiveness:** the system must change as little as possible the environment in which it operate, from a purely aesthetic point of view but also from an electromagnetic (or sound) wave emissions point of view.

- **Ease of installation:** the system must let its deployment inside an environment also to a non-experienced personnel in an easy way.

- **Ease of use for the final user:** the app for the final user must have an intuitive UI (user interface). It must not need any cognitive workload for the user to auto-locate himself. Moreover, it must let the user to reach the desired area inside the building, easily.

- **Accuracy:** the system must have an high level of accuracy, according to the specific requirements.

- **Scalability:** the general system architecture must be scalable and simple to integrate into commercial solutions.

Even if there are a lot of solutions for indoor localization/navigation using smartphones, our researches have shown that actually none of them is capable to integrate all the previously seen requirements in an effective manner.

1.3 Contributions

The smartphone-based solution which will be proposed (in order to let the user to auto-locate himself inside a building), will be based on a combination of (a) computer vision techniques and (b) inertial navigation using the device's motion sensors.

Specifically, a 2D visual markers system will be deployed onto the target building's floor, and each marker will have an associated position on the building's map. The developed app, by using the smartphone's camera, will decode the marker ID and use that information to obtain the user position. Dead reckoning algorithms will be applied to track the user between two markers (due to accuracy reasons, a pedometer will be used). Every time a marker will be encountered, the cumulative dead reckoning error will be reset.

A server side part of the system will be responsible for managing the download of the indoor maps (and all the associated information, such as layers, associations between marker ID and position, GPS coordinates of the building etc.) on the client. The development of the whole architecture will pass through the following phases:

1. Analysis of the intrinsic features which the place of deploy (the floor) has, for what concerns the used indoor localization's methodology.

2. Analysis of the 2D visual markers systems' state of the art. According to (1) the results of the analysis, (2) the deployment place's required requirements and (3) the availability of free, open source, cross platform and well-maintained libraries, a specific marker will be chosen.

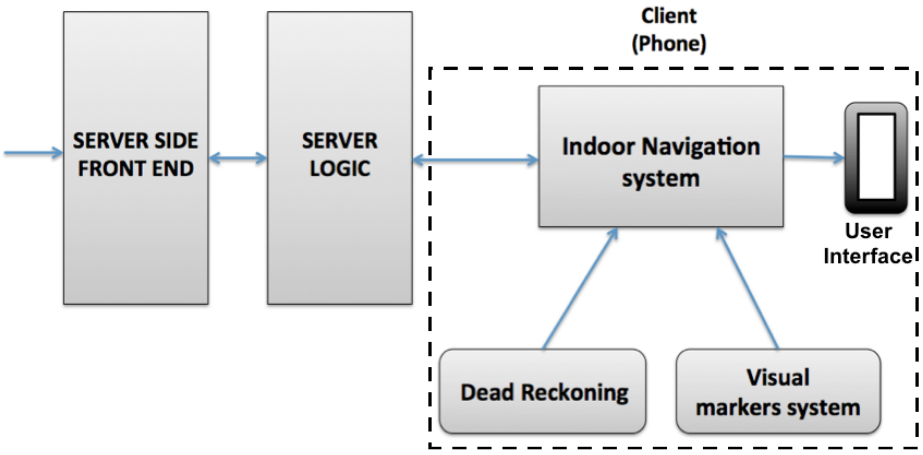

Figure 1.1: *Overview of the indoor localization and navigation system based on 2D visual markers and Dead Reckoning*

3. Analysis of the inertial navigation systems present in the scientific literature, and of the frameworks provided by the chosen development platform (iOS's Apple platform).

4. Integration of (1) the software library for marker decoding, and (2) dead reckoning algorithm in an indoor localization/navigation client app.

5. Design of the server architecture and data structures which will be exchanged with the client.

The Figure 1.1 give an overview of the whole system we worked on.

1.4 Report outline

This dissertation is organised in 9 chapters (including this introduction) as follows:

Chapter 2 presents an overview on the main methodologies and technologies actually used for localization/navigation in indoor environment, and tries to explain why the smartphone was chosen as preferred developing platform.

Chapter 3 presents a deep scientific literature analysis, focusing on few related works which are more significant from a scientific point of view.

Chapter 4 presents the proposed indoor localization/navigation approach, which use dead reckoning and a visual markers system deployed onto the floor. A list of some of the most important visual markers is presented.

Chapter 5 give an overview on the most known visual markers in scientific literature, and focus on three of them from the point of view of our approach.

Chapter 6 presents the server side structure of the proposed indoor localization system.

Chapter 7 presents the client side structure of the proposed indoor localization system.

Chapter 8 summarizes the results.

Chapter 9 gives some conclusions and proposes future works.

CHAPTER

TWO

BACKGROUND

In this chapter we will present the motivations which have led us to choose the smartphone as preferred development platform (particularly, the Apple operating system, iOS) instead of focusing on dedicated hardware-based solutions, and give an overview on the main methodologies and technologies actually used in the indoor localization/navigation field.

2.1 Smartphone-based IPS

Even if, currently, the indoor positioning systems (IPSs) market is quite wide and proposes often, especially in industrial sector, solutions which use dedicated hardware and technologies specifically designed for indoor localization, in consumer sector we can affirm that smartphone has become the main platform on which the researchers and startuppers efforts have focused. The reasons are various, some more

obvious because of practical issues, other more "technical" but not less important.

The main one, is undoubtedly the fact that the smartphone is become in the modern society an essential tool which we bring always with us, everywhere we go. It contains a certain number of sensors which let to sense the context and turn the simple information about the indoor position in something else, capable of enabling a lot of functionality and services (augmented reality, LBS, smart navigation, pathfinding) which considerably enhance the user experience. Moreover, these kind of devices in the next years will be further enhanced and enriched with new typologies of sensors, new features and functionality: this guarantee to the IPS designer, without any cost, the availability of increasingly powerful hardware, new directions in terms of development of the system, efficiency.

Today in the mobile field, there are two big players which together cover almost all the smartphones market: Apple, with its mobile operating system iOS, and Google, with the mobile operating system Android (Microsoft recently is starting to conquer a small slice of the mobile market thanks to the operating system Windows Phone). Both of them have worked, and are still working (with huge financial investments) on the indoor positioning field, as further proof of the fact that it is a strategic and extremely interesting sector.

In the present dissertation we choose to use iOS as development platform, both because we already developed for the platform before, so the learning curve was not so steep, and for the specific device's features: less fragmentation than Android, users usually tend to update soon their operating system to the last release, camera and sensors are top quality. In addition, the Integrated Development Environment

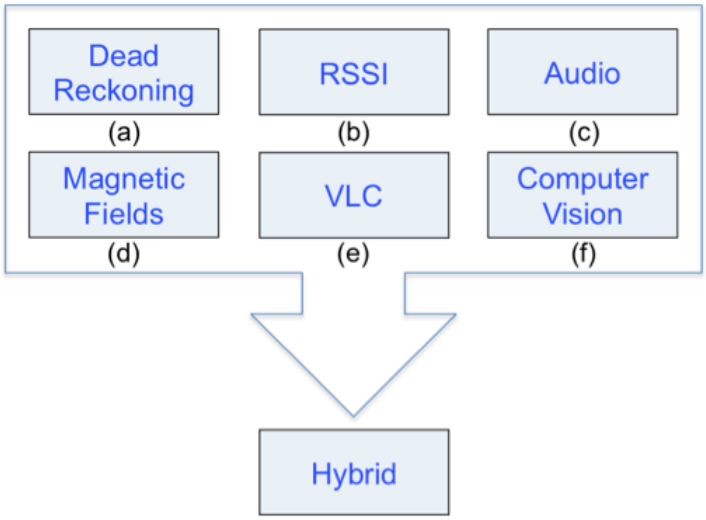

Figure 2.1: *Overview of the main indoor localization techniques*

(IDE) is developer-friendly, there are easy to use APIs (Application programming interfaces), and a huge and very active online community. Our prototype was thus developed and tested on an iOS based device, particularly on an iPhone 5S.

2.2 Overview of IPSs

Thanks to its importance from an economic point of view, nowadays, indoor navigation is a very hot topic. Various approaches and solutions have been proposed in literature to address the challenge in a simple and scalable way, and also a lot of commercial solutions are appearing into the market. In Figure 2.1 we give a brief and non-exhaustive summary of the main techniques used to locate the user inside an environment.

2.2.1 Dead Reckoning systems

They use accelerometer, magnetometer and gyroscope sensors embedded into the smartphones to provide fast estimation of the user position, starting from a known position. Even if the Dead Reckoning can theoretically track precisely the user, practically the system is subject to an high drift error introduced by the sensors, which make it unusable after few seconds. For this reason, usually a step counter algorithm is used to calculate the covered distance, the gyroscope and the compass are used to determine the orientation of the motion and a periodical recalibration is performed in order to reset the cumulative drift error [1], [2], [3], [4], [5].

Typically, dead reckoning systems are used in combination with other indoor localization technologies to improve the accuracy of the whole system.

2.2.2 RSSI systems

Received Signal Strength Indication (RSSI) systems exploit the RSSI of the radio signals present in the environment, typically Wi-Fi signals, available for free in public buildings, or, recently, Bluetooth Low Energy (BLE) signals.

They can use triangulation/trilateration (a minimum of three radio emitters, placed in known positions, are needed) to detect the position of the device or, more frequently, information from a previously generated RSSI fingerprint database of the environment to estimate the position of the user. Triangulation/trilateration techniques rely on geometry and the fact that the RSS is inversely proportionate to the square of the distance between the emitters and the target device,

to estimate the position of the user. However, due to the interferences from the environments, the multipath effect etc., these systems are complex and not very reliable.

Fingerprinting techniques rely on the comparison between the RSS measured by the smartphone in an unknown position and the pre-recorded RSS value measured in a small range around that position. They usually are splitted into two phases: an Offline phase of measuring the signal strength at selected measure points in the analyzed area in order to build the fingerprint database, and an Online phase which uses the values stored in that database, plus the RSS values collected during runtime, to estimate the position of a user [6], [7], [8], [9], [5]. Fingerprinting techniques are more accurate than triangulation/trilateration techniques, even if, due to the fact that they need a pre-mapping phase, are very susceptible to strong changes in the environment.

2.2.3 Audio systems and Magnetic systems

Audio systems exploit controlled (usually malls, consumer stores and museums are equipped with loudspeakers) or uncontrolled (for example acoustic background fingerprint) ambient sounds to allow a simple smartphone to cheaply determine its location [10], [11]. Depending on the particular adopted approach, they can be used both for localizing the user in a precise way or as add-on to increase the accuracy of some other indoor technology.

Magnetic field systems use the indoor ambient magnetic fields (caused, for example, by elevators, escalators, doors, pillars, and others ferromagnetic structures) to build a magnetic map of the environ-

ment. The uniqueness magnetic signatures from this map (magnetic fingerprint) are exploited by the smartphone to solve the indoor localization problem [12], [13]. The interest from both researchers and startup around this approach is increased in the last years; one of these startups in particular, IndoorAtlas[14], inspired by the capability of some animals to use the earth's magnetic fields for orientation detection and navigation, proposed a commercial indoor positioning solution based on a similar principle.

2.2.4 Visible Light Communication Systems

Visible Light Communication (VLC) systems exploit the susceptibility of LEDs to the amplitude modulation at high frequencies to transmit information into the environment. If the frequency is greater than a flicker fusion threshold, the lighting functionality is preserved because the modulation is unperceivable for the human eyes, while it is possible to perform accurate indoor positioning.

The transmitter modulates the LEDs light with the data signals (these data are strictly related to the position of the specific LED). The receiver contains a photodiode which receives such signals, elaborates them and extracts the information about the position of the device.

Thanks to the fact that the sensors inside the smartphone cameras are array of photodiodes, it is possible to use the smartphone as device for VLC-based indoor localization [15], [16].

2.2.5 Computer Vision Based Systems

Recently, thanks to the high performances cameras and high computational capabilities of the last generation smartphones, researchers are focusing on Computer Vision systems which rely on complex, CPU-intensive marker-based or markerless computer vision algorithms to determine the position of the user [17], [18], [19].

In marker-based approaches, usually a set of easily detectable, predefined 2D visual markers are deployed into the environment, and computer vision techniques are used to decode the ID which them contain. Each marker ID has an associated known position on an indoor map, which let to position the user in the correct coordinates with an high precision. Markerless-based approaches rely on image analysis: the position of the user is obtained from what the smartphone sees in the environment. Usually there is an offline phase where a database of the features is created. A subsequent online phase, performs (a) image acquisition through the smartphone camera, (b) image segmentation and relevant features extraction through computer vision algorithms and (c) matching between the detected features and the pre-generated database of features.

Beside this, a lot of other different computer vision approaches are studied in scientific literature, each one with its strengths and weaknesses.

2.2.6 Hybrid Systems

Usually, Hybrid techniques and technologies are used to improve the accuracy, reduce costs and enhance the performances of the whole indoor positioning system [20], [21]. This is particularly true for indoor

positioning systems based on the smartphone, thanks to the fact that it embeds sensors (such as inertial sensors and magnetic sensors), it integrates two cameras, it has multiple antennas, integrates Bluetooth Low Energy etc.

CHAPTER

THREE

RELATED WORKS

In this chapter we will go deeper on the scientific literature analysis, focusing on few related works which, in our opinion, are more significant and interesting from a scientific point of view.

3.1 UNLOC

Researchers from Duke University in 2012 proposed UNLOC (Unsupervised Indoor Localization) [20], [22]. Their solution (actually patent pending) started from the following considerations:

(a) All the indoor localization solutions which have a good level of accuracy require a pre-mapping of the environment. Usually, this procedure is complex to perform and more or less expensive, depending on the approach. Moreover, if the environment conditions change, a new pre-mapping phase is necessary, to preserve

the same level of accuracy.

(b) It is possible to calculate the motion trajectory of a smartphone by using its accelerometer, gyroscope and compass (*Dead Reckoning*). Even if at the beginning the trajectory is pretty accurate, due to the drift error generated by the sensors, it diverges after few meters so, a periodic recalibration of the position is needed.

(c) Certain locations in indoor environments presents in the sensor domains identifiable signatures (which they call *Landmarks*) generated by elevators, escalators, Wi-Fi etc..

According to what the authors say, is therefore possible to use dead reckoning (step counter, gyroscopes and compass) to track the user, and periodically reset the error when the user encounters a landmark. This theoretically lets him to navigate inside a building without any previous knowledge of it. Regarding the landmarks, the scientific paper distinguishes between seed landmarks and organic landmarks. *Seed landmarks* are structures in the buildings such as stairs or elevators that force the users to behave in a predictable way so they impress an identifiable signature in the sensors domain. *Organic landmarks* are ambient signatures which are not known a priori so they must be dynamically recognized (Wi-Fi or 3G/GSM deadspots, magnetic signatures in specific areas etc.). The Figures 3.1a, 3.1b and 3.2 explain better the concepts.

The density of landmarks is obviously fundamental. The tests conducted by the authors showed that big buildings such as airports and malls are rich of landmarks coming from the environment: lights, sounds, magnetic fields, 3G and Wi-Fi, escalators, elevators, stairs

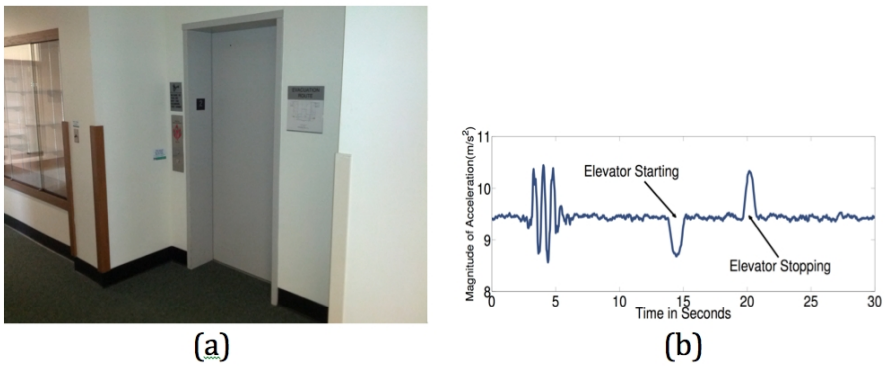

Figure 3.1: *Seed landmarks examples: (a) elevator, (b) pattern impressed in the inertial sensors domain.*

Figure 3.2: *Organic landmark example: a small area where there is not Wi-Fi coverage.*

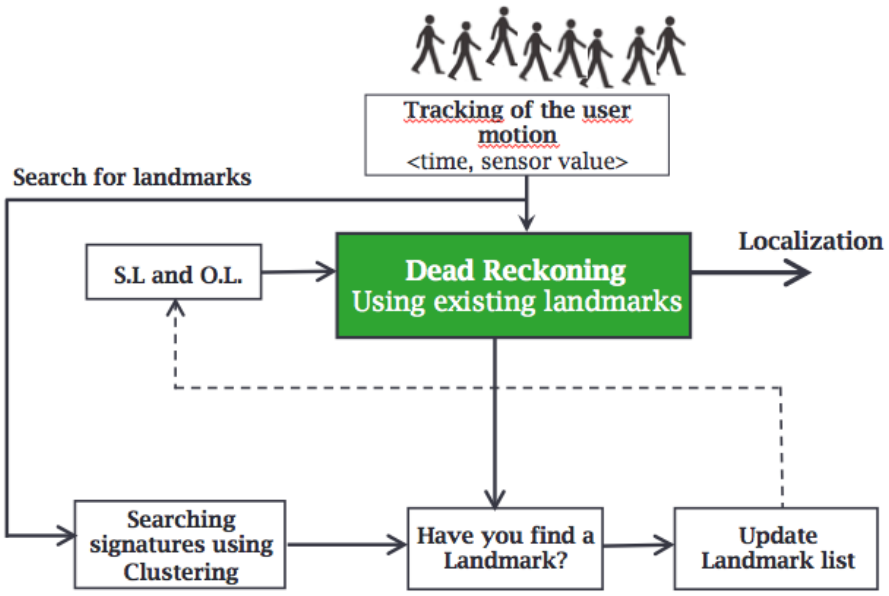

Figure 3.3: *UNLOC recursive algorithm*

etc. Moreover, the landmarks don't need to be unique in all the building, because it is possible to perform a Wi-Fi based partition of the whole area into sub-areas, so the landmarks must be unique in only in each sub-area. The estimation of the landmarks positions (initially unknown) and the identification of new ones, are performed by elaborating data which come from all the users: every new user improves the previous measurements. The Figure 3.3 shows the recursive algorithm.

The diagram in Figure 3.4 shows the accuracy of the algorithm, which the authors say is around 1.2 mt. As we can see, the drift error is periodically reset when a landmark is encountered.

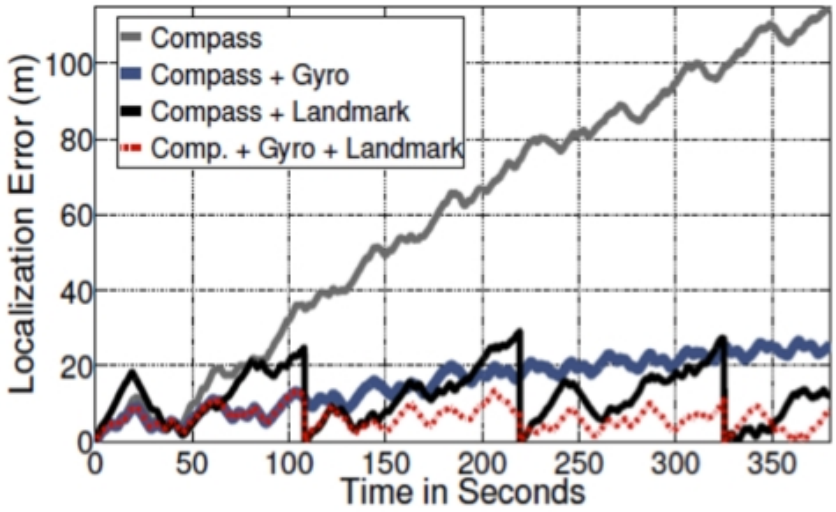

Figure 3.4: *Error from dead reckoning reduces with gyroscope, and further with landmarks*

3.2 LED based infrastructure for Indoor Positioning

In *"Visible Light Communication: Opportunities, Challenges and the path to the Market"* [15], the authors of the paper suggest the use of LEDs and Visible Light Communications (VLC) to localize the user inside an environment in an accurate way.

On the transmitter side, the modification to the LED lighting infrastructure is cheap and simple (power efficient switch-mode amplifiers are already present on the LED lamps, so the only cost comes from the programmable logic devices which drive the amplifier). Moreover, actually the use of LEDs as light sources in the commercial/industrial markets is increasing, so the basic LEDs infrastructure already exists in a lot of buildings.

On the receiver side, Harald Haas (one of the pioneers on this field) [16] shows that it is possible to exploit the rolling shutter effect of CMOS-based camera sensors [23] to let a mobile phone to decode the information transmitted by the LEDs infrastructure (based on the fact that during the normal use of the phone, while the user is watching the screen of the device, the front camera is directed toward the roof, so it can perceive the LED light). The information is captured in the camera in the form of light and dark bands which are then decoded by the smartphone.

Thanks to VLC the congestion of the 2.4GHz ISM (Industrial, Scientific and Medical) band could be alleviated because it operates in a different band. In addition, till today the visible spectrum doesn't need licenses, and is actually unused. Moreover, it is possible spatial

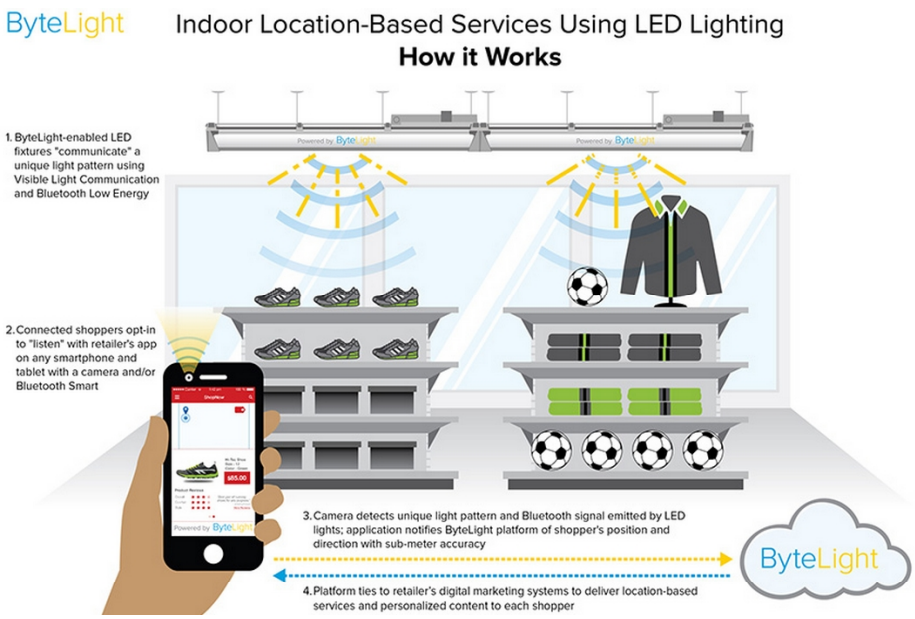

Figure 3.5: *Overview of the ByteLight system*

reuse thanks to the fact that the light doesn't cross the walls. Of course, there are some disadvantages: these systems need to have a line of sight to work properly; moreover, if the indoor space doesn't have a LEDs infrastructure, the whole system could be expensive to install. Also, if we use the smartphone as receiver, the approach, due to the fact that is based on the camera, consumes a lot of energy.

Several startups, in recent years are proposing indoor navigation commercial solutions based on VLC. The most known, ByteLight [24] (we can see an overview of their system on Figure 3.5), combines VLC, BLE and inertial device sensors to transform LED lights into indoor location waypoints. Its platform promises an accuracy of 1 meter.

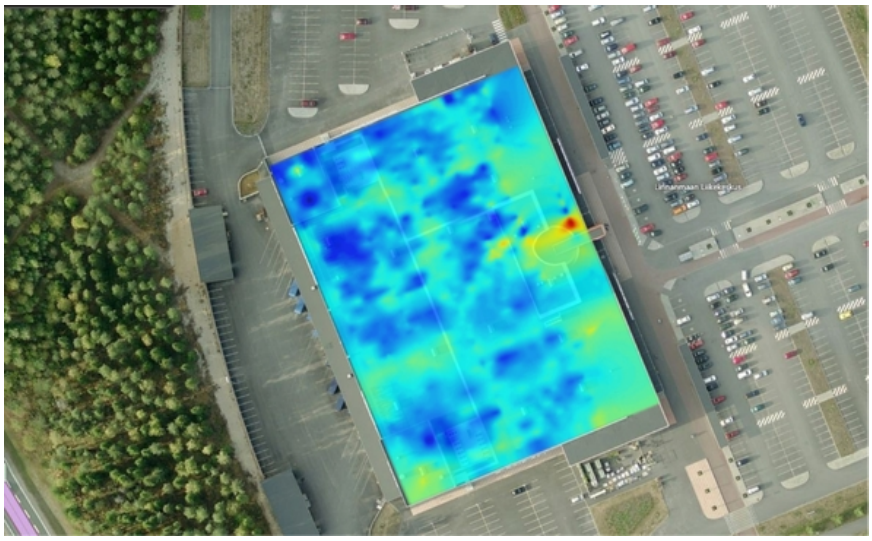

Figure 3.6: *Magnetic fingerprint of an environment. Different colors represent different magnetics intensity levels.*

3.3 IndoorAtlas

The interest from both researchers and startups around the magnetic fields approach is increased a lot in the last years; the finnish startup IndoorAtlas [14], inspired by the capability of some animals to use the earths magnetic fields for orientation detection and navigation, proposed a commercial indoor positioning solution based on that principle. In their white paper they suppose that modern buildings often contain steel structures, which create a sort of magnetic fingerprint of the environment (Figure 3.6), and exploit these anomalies to localize the user.

The indoorAtlas platform provides a cloud-based location service and an API set to communicate with this service. The location service

connects to the map database, which hosts the magnetic field data collected from the building using the *IndoorAtlas Map CreatorTM* application [14].

3.4 Apple IPSs

Apple (and Google too) has included APIs for indoor positioning in the last released SDK. It provided an enhancement on its Core Location framework [25] to let developers an easy transition between outdoor and indoor navigation. Moreover, it provided for free a new portal to add or edit local business listings (with some features such as public access in the building, one million or more of visitors per year, Wi-Fi enabled, etc.): *Apple Maps Connect* [26]. The listings (or corrected listings) appear on Apple Maps on the PC and on the mobile, so the user can track himself inside them. Behind the scene, Apple (and Google, which has a similar feature) uses mixed technologies such as Wi-Fi fingerprint, BLE (Bluetooth Low Energy) iBeacons and dead reckoning to perform the indoor localization task.

3.5 ArtoolKit based indoor localization

As we stated before, marker based approaches rely on 2D visual markers, which can be easily decoded even by a low-cost smartphone, to let the user to track his position inside buildings. B. L. Ecklbauer, in his thesis [27], proposes the recognition of multiple custom ArtoolKit visual markers [28] inside an image captured by the camera to deduce the position of an Android smartphone, with no additional data

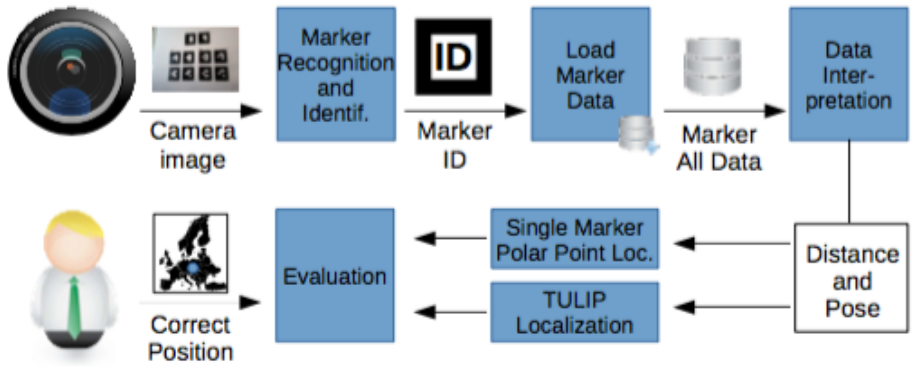

Figure 3.7: *ArtoolKit based indoor positioning system scheme*

sources, except for the knowledge of the markers positions. In the first part of his thesis, he compares the detection/decoding performances of QR-Code against ArtoolKit markers, showing some interesting measurements about speed detection and detection rate under different light conditions and for different sizes of the markers. Then he proposed a prototype of ArtoolKit based indoor positioning system. The architecture of the prototype is illustrated in Figure 3.7.

As we can see, the process of localization start with a marker's recognition and identification phase: the camera image is analyzed using ArtoolKit library, and the marker IDs are extracted. The data related to the marker ID are loaded in the application. By a data interpretation phase (which uses the retrieved information about marker position and the marker distance and pose), the localization approaches can be triggered. The single marker polar point localization calculates the position of the user by mean of only one recognized marker. The TULIP localization (Trilateration utility for locating

internet protocol addresses) calculates a 2D position using 3D information and at least three markers, recognized at once. The final evaluation phase compare the localization approaches to the previously entered correct position.

3.6 Custom Marker-based IPS

In the report *"An Indoor Navigation System For Smartphones"* [17] the authors propose a simple, custom, color-based 2D visual marker, to obtain the user position and orientation, and a step detection algorithm to track the user between two markers. The system relies on the robust OpenCV library for the marker detection and for avoiding the obstacles along the path. It does not require any expensive alterations to the infrastructure or any prior knowledge of the sites layout. The Figures 3.8a, 3.8b show the custom, coloured marker. The marker encodes both information about its position and its orientation respect to the camera and can be scanned from any angle. The authors suggest to print the marker such that it fill an A4 paper. The Figure 3.9 shows the angular shift that let to calculate the orientation of the marker. To detect the free walkable space around the user, the authors implemented a simple obstacle detection algorithm based on boundary detection through Hough Transform and Canny Edge Detector.

In the absence of custom markers, the user is tracked by using a simple pedometer and multiplying the number of steps for the users stride length (of course, by considering also the orientation). Despite of the simplicity and scalability of the last two marker based tech-

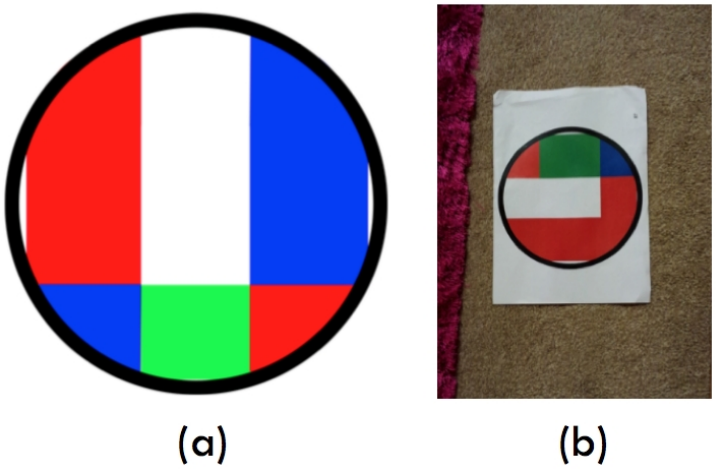

Figure 3.8: *(a) Coloured custom marker (b) Custom marker deployed onto the floor*

Figure 3.9: *Angular shift that lets to calculate the orientation*

niques, there are some drawbacks such as (a) the need of a line of sight, (b) the sensitivity to light changes, (c) the size of the marker, which must be as smaller as possible in order to be minimally invasive, (d) the fact that the app does not work in real-time, and (e) the cognitive workload for the user who has to look for the marker in order to auto-locate himself (these procedures, if could be annoying for normal people, can become very difficult for people with visual deficits).

3.7 Conclusions

The solutions and approaches showed in this chapter, are only a small part of the whole indoor localization world. In the next chapters, we will introduce a computer vision based approach which aim at solve some of the drawbacks of this category of techniques, and propose a server/client architecture.

CHAPTER

FOUR

IPS WITH VISUAL MARKERS DEPLOYED ONTO THE FLOOR

As we have seen in the previous chapters, even if there are some drawbacks, computer vision based indoor localization systems are becoming more interesting recently, mainly for three reasons: (1) the smartphone camera is becoming more powerful in every new release of the device, (2) the approach works well even without internet connection, and (3) it is low cost. Markerless approaches are used when visual markers are undesirable due to aesthetic reasons. Because they are complex to realize, really CPU intensive, often require a considerable workload before they can start to work and also need frequent re-calibrations, they are less common than marker based approaches.

In the next paragraphs we will focus on marker based approaches (a visual marker can be considered as a mark, which can be detected on camera image in the visible spectrum of light). Our analysis showed

that, in these kind of systems, usually the visual markers are placed on different parts of the buildings, such as walls, doors, pillars etc. This results in a feeling of discomfort for the final user who has to consciously search for the marker each time he want to know his position: the user experience gets worse and it is not possible to use the system as a navigator. Moreover, it is not possible to deploy the system in indoor open spaces.

4.1 Why onto the floor?

In order to let the user to auto-locate himself without any cognitive workload, and starting from the observation that when the user launches the app in navigation mode, the camera is in the palm of his hand and will be necessarily directed towards some part of the floor, we have proposed in this dissertation an hybrid approach to the indoor localization/navigation through smartphone, which uses a 2D visual markers system deployed onto the floor to estimate the position with a good level of accuracy (depending on the density of markers), and dead reckoning to track the user between two markers. Each marker has an associated 2D position on a previously generated indoor map of the building.

Place the 2D visual marker system onto the floor makes the localization process through computer vision algorithms, almost transparent for the final user: the smartphone, during the normal functioning of the app in navigation mode, by running its camera in background analyzes each frame, looking for a visual marker (the user doesn't need to consciously look for the marker.) Moreover, the floor has some in-

trinsic features which can be exploited to improve the performances:

- Almost uniform, prior-known background pattern of the floor, as shown in Figure 4.1a. It is possible to use this feature to improve the speed of the decoding algorithm and to reduce the physical size of the marker.
- Almost fixed, prior-known size of the marker inside the frame, as shown in Figure 4.1b. It depends on the distance between the camera - which is on the palm of the hand - and the floor, and makes easier and faster to find the marker inside the frame, which brings to a detection speed improvement.
- Major probability for the marker to be in the upper part of the frame as shown in Figure 4.1c, due to the fact that when the user launches the app to navigate inside a building, he moves forward. Relying on this, it is possible to first analyze a subportion of the frame in order to further improve the detection speed, or use the saved time to apply some filters to such subportion in order to enhance the quality of the image.
- Prior-known markers positions, so it is possible to reduce the errors by considering that each decoded marker must necessarily be one of the markers in the boundary of the previously decoded marker.

4.2 Visual marker required features

To realize an efficient indoor navigation system based on 2D visual markers deployed onto the floor, a critical point is the choice of the

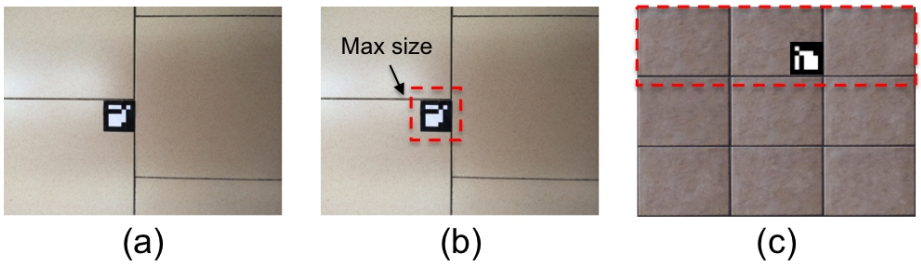

Figure 4.1: *(a) Uniform background pattern of the floor. (b) Max size of the marker inside the frame. (c) Major probability for the marker to be in the upper part of the frame*

visual marker which best fits the particular place of deployment. The characteristics that the chosen marker must have, considering that when the user uses the system he is moving (usually with low speed), are:

1. *Small size:* this feature is required to reduce the invasiveness of the system. We have to find the best compromise between size, speed of detection/decoding and robustness of the algorithm.

2. *Real-time detection:* To make the auto-localization process through visual markers transparent for the final user, the detection must be as fast as possible.

3. *Robustness to changes in light conditions:* this feature is required because typically the marker will be deployed in high dynamic environments, characterized by the presence of other people, on/off switching of lights, shadows etc.

4. *Robustness in detecting blurred or out-of-focus markers*, caused

by too fast movements.

In the next sections, we will talk about dead reckoning techniques to track the user between two markers, give a brief overview on the most famous markers and introduce three 2D visual markers which best fit our requirements: Vuforia (created by Qualcomm Technologies), ArUco marker (created by A.V.A. group from University of Cordoba), and AprilTag (created by Edwin Olson from University of Michigan). We chose these markers because they are well-documented, opensource (AprilTag and ArUco) or with a free-to-use available SDK (Vuforia), and have good global performances.

4.3 Dead Reckoning

Dead Reckoning is the name of the process which estimates the position of a user by exploiting a previously known position (the starting point) and the measurements made by an inertial sensors system. Geometrically speaking, if we know a point, the distance covered by the user (calculated by using speed and time) gives us a circle of possible new positions. The motion direction, lets to determine on which point in that circle is located the user. Modern smartphones are usually equipped with accelerometers (Figure 4.2) which theoretically let to apply a double integration on the measured acceleration in order to estimate the traveled distance, and gyroscopes/compasses to estimate the motion direction. Unfortunately, due to the cumulative errors generated by these sensors (which strongly deteriorate the accuracy after just a couple of seconds) this approach is completely useless for indoor localization. Instead, a better approach is using inertial sensors mea-

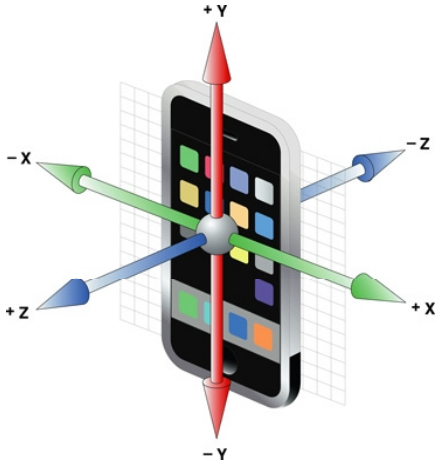

Figure 4.2: *The three axes of the device. The accelerometers measure the linear accelerations along these axes.*

surements to count the number of steps of the user: if we can know some information about the user (age, sex, height), it is possible to estimate the step length and then, by multiplying the number of steps for the step length, we obtain the required distance. By exploiting gyroscope and compass, then, we can estimate the trajectory. Such kind of systems are usually called *Pedometers*. There are a lots of scientific papers about how to realize an accurate pedometer, and also, a lots of real applications which are based on those principles.

In *"A Reliable and Accurate Indoor Localization Method Using Phone Inertial Sensors"*, [2] the authors - from Microsoft Research Asia - developed a reliable algorithm for step detection and dynamic step length estimation (which is adapted on the specific user, and estimated for each step), together with a way to calculate the heading direction for each step. The scheme on Figure 4.3 shows such

system. The *step detector* module uses the data collected from the accelerometers: it considers the magnitude of 3-axis accelerometer reading, applies on it a low pass FIR digital filter, and then searches for the peaks and valleys of the waveform, in order to identify a single step. Heuristic constraints and a validation phase (which exploit the Dynamic Time Warping algorithm) refine the results and avoid the false positives. The *heading estimator* module uses the data collected from the compass, gyroscopes and accelerometers to estimate the user motion direction at each step, in any position of the phone, with high reliability. To get an accurate step model, the authors start from a generic one, provided at the beginning, and then collect the data generated by the specific user to learn a personal model. This phase is very important because the generic step model is inaccurate due to the fact that the real step length is influenced by various factors such as the height and weight of the user, the ground types, the shoes etc. The *particle filter* algorithm, combines information coming from the step length estimator and the heading estimator, together with the constraints coming from the floor map, to continuously adapt the personal step model and estimate the position of the user on the map step by step.

Quite recently, both Samsung and Apple have added hardware to their devices, to better support the step counter applications. Apple in particular, starting from the iPhone 5S, have introduced in their smartphones a dedicated motion coprocessor called M7 (with iPhone 6/6 plus and iPhone 6S/6S plus new version of such chip, M8 and M9 were used). The coprocessors collect, store and elaborate data (in a power-efficient way) coming from the inertial sensors even if the device is in background mode and lets the applications to retrieve such data

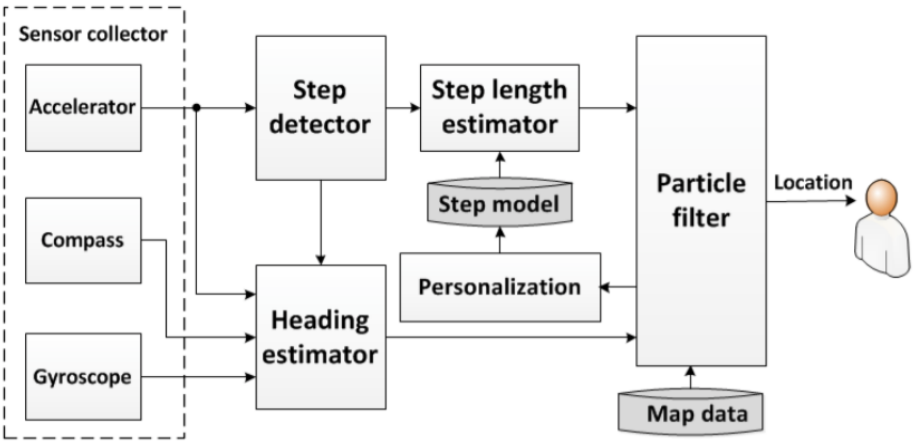

Figure 4.3: *Pedometer architecture*

through the CoreMotion Framework.

4.4 Visual markers in scientific literature

A visual marker system is composed by a set of 2D visual markers and a computer vision algorithm capable to detect and decode each marker using a smartphone camera or other computer vision technologies. Today, thanks to their low cost, flexibility and simplicity (a normal smartphone is able to generate and read most of them) there are several visual markers in the market. In the following subsections we show some of them.

Figure 4.4: *An example of Quick Response Code*

4.4.1 Quick Response Code (QR-Code)

The most known and used visual marker is probably the QR-Code (Figure 4.4) [29]. QR-Code is a bidimensional barcode invented in 1994 by the japanese company Denso Wave to track assets [30]. In 1999 Denso Wave released QR-Code under a free license (it was defined and published as ISO standard), which made QR-Code very popular, in Japan at the beginning and few time later, in all over the world.

Today we can see QR-Codes almost in any commercial product, in websites, mobile applications, for assets tracking, mobile payments etc. Also, it is very well documented and there are hundreds of free software libraries (Apple, natively includes the support for QR-Code scanning and decoding) for almost all the platforms, which make it very easy to incorporate. There are several standards for encoding the data inside the QR-Code. In Figure 4.5 is shown an example of QR-Code structure. As we can see, there is a grey area with small black dots which contains the data, three larger squares (4.1) used to

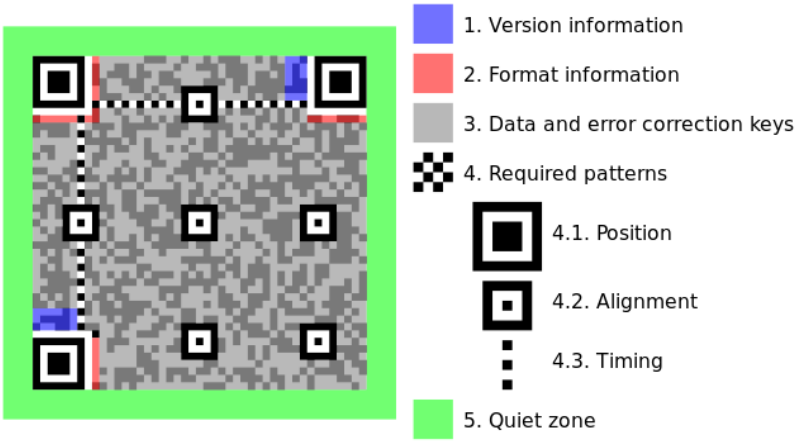

Figure 4.5: *QR-Code structure example*

determine the correct position of the QR-Code, and several smaller squares (4.2) to detect the alignment. Around the QR-Code, there is a (usually white) quiet zone, to help its detection. The version information fields (1) contain the symbol version of the QR-Code. It can range from Version 1 to Version 40. Each version has a different module configuration or number of modules (the module refers to the black and white dots that make up QR-Code) and a maximum data capacity, according to the amount of data, character type and error correction level. Figure 4.6 and Figure 4.7 show versions range from 1 to 4.

The QR-Code has a Reed-Solomon based error correction algorithm capable of restoring data if the QR-Code is dirty or damaged. The third row of the table in Figure 4.6 shows the four correction levels:

1. **Level L**: approximatively 7% of data restoration rate for total

Version	Modules	ECC Level	Data bits (mixed)	Numeric	Alphanumeric	Binary	Kanji
1	21x21	L	152	41	25	17	10
		M	128	34	20	14	8
		Q	104	27	16	11	7
		H	72	17	10	7	4
2	25x25	L	272	77	47	32	20
		M	224	63	38	26	16
		Q	176	48	29	20	12
		H	128	34	20	14	8
3	29x29	L	440	127	77	53	32
		M	352	101	61	42	26
		Q	272	77	47	32	20
		H	208	58	35	24	15
4	33x33	L	640	187	114	78	48
		M	512	149	90	62	38
		Q	384	111	67	46	28
		H	288	82	50	34	21

Figure 4.6: *QR-Code Versions Table*

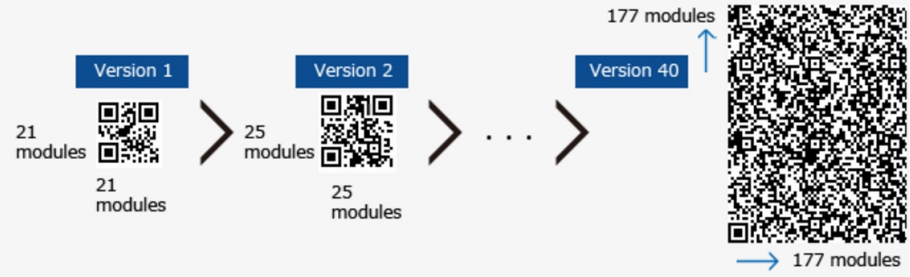

Figure 4.7: *QR-Code Versions*

codewords.

2. **Level M**: approximatively 15% of data restoration rate for total codewords.

3. **Level Q**: approximatively 25% of data restoration rate for total codewords.

4. **Level H**: approximatively 30% of data restoration rate for total codewords.

(NOTE: codeword is a unit that constructs the data area. One codeword of QR-Code is equal to 8 bits).

A single QR-Code can theoretically store 7089 numeric characters or up to 4296 alphanumeric characters even if the more characters it stores, the harder is the decoding process through a simple smartphone. Due to the fact that QR-Code was designed for storing a large amount of data, it has not good real-time performances, even if we use the symbol version 1 (the less dense version). For this reason, it is not good for our purposes. To guarantee real-time performances, usually a visual marker which stores just a simple binary code is used.

4.4.2 ArtoolKit

ArtoolKit is a software library for building Augmented Reality applications, which is capable of determine the pose of the camera respect to a physical marker (ArtoolKit marker), almost in real time. It was developed and released in 1999 by dr. Hirozaku Kato in the Human Interface Technology Laboratory (HIT Lab) at the University of Washington. After a lot of years, finally On may 13, 2015 it was released

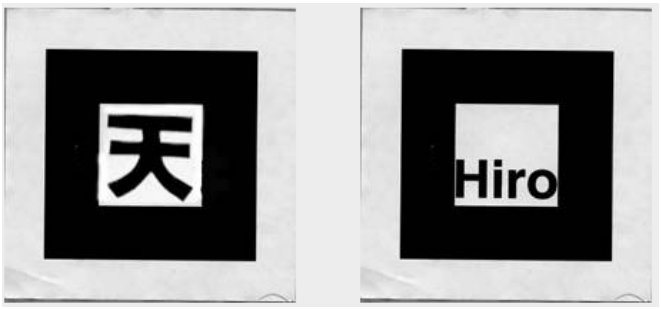

Figure 4.8: *ArtoolKit sample markers*

opensource with all the professional features enabled [28], [31]. On Figure 4.8 two examples of possible ArtoolKit markers are shown.

The shape of an ArtoolKit marker can be, in theory, any image surrounded by a black square (even if, for performance reasons, usually the marker consists in a very simple black and white image). In fact, ArtoolKit algorithms are based on the more general template matching approach: the detected marker is unwarped and adapted to the marker templates (same size and scale), and a similarity value is calculated; the template with the highest similarity value is the correct marker, while all the recognized markers with a similarity value lower than a threshold are rejected (to speed and improve the process, before applying the template matching, the greyscale image is converted in black and white image).

The Figure 4.9, taken from the official ArtoolKit website, explains the basic principles on which the library is based. The scheme focuses on using the library for augmented reality, but for our indoor localization purposes with visual markers deployed onto the floor, we can simply use each marker as a beacon for identifying a position on

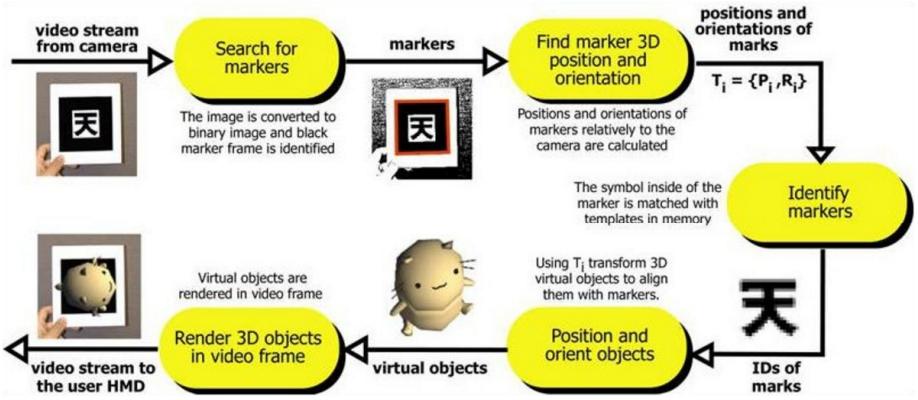

Figure 4.9: *Basic steps in ArtoolKit library*

a map, and augment such information with visual information (such as arrows for turn-by-turn navigation etc.). In the following, a brief description of the various steps.

1. The smartphone camera captures video of the real world.

2. The library searches through each video frame for any square shapes.

3. If a square is found, the library uses some mathematics to calculate the position of the camera relative to the black square.

4. Once the position of the camera is known, a computer graphics model is drawn from that same position.

5. This model is drawn on top of the video of the real world and so appears stuck on the square marker.

6. The final output is shown back in the display, so when the user

Pattern size (cm)	Usable range (cm)
6,985	**40,64**
8,89	**63,5**
10,795	**86,36**
18,7198	**127**

Table 4.1: *different marker sizes for different distance marker-camera*

looks through the display he see graphics overlaid on the real world.

Similarly to all the other visual markers, even ArtoolKit marker has some limitations. As stated in the Artoolkit official website, results are affected by lighting conditions: overhead lights may create reflections and glare spots on a paper marker and so make it more difficult to find the marker square. To reduce the glare, markers can be made by more non-reflective material. Moreover, detection/decoding depends also on the marker orientation respect to the camera; as the markers become more tilted and horizontal, less and less of the center patterns are visible and so the recognition becomes more unreliable. Also, sometimes markers are confused one with another or falsely detected in the background. Finally, for small markers, we have a small possible distance marker-camera, as we can see in Table 4.1. (These results were obtained, by the authors, by making markers of different sizes, placing them perpendicular to the camera and moving the camera back until the markers were decoded).

4.4.3 Bokode

On Figure 4.10 is shown Bokode, an innovative (and patent-covered) marker invented by the MIT (Massachusetts Institute of Technology) Media Lab [32]. It may be read by a simple camera from a distance of over 4 meters, it is circular and very small (only 3 mm of diameter). According to what the authors say, it works by exploiting the Bokeh effect [33] of ordinary camera lenses, which maps a cone of rays exiting from an out of focus scene point into a disk shaped blur on the camera sensor. From a diffuse point, all these rays have roughly the same radiance, so the bokeh takes the shape of the round aperture. The Media Lab team created a directionally varying set of encoded rays by placing a small diameter (3mm), short focal length (8 mm) lenslet over a printed binary code. The imaged bokeh has a much wider diameter and it shows a magnified view of the binary code allowing the decoding of thousands of bits. As we can see in Figure 4.10 a bokode is just a tiny dot for the human eye and for a camera in sharp focus. But the binary code inside it appear very clearly when the camera is out of focus (it is possible to observe features as small as 2,5 µm). Bokode can be used to decode an ID, to store a lot of information (lot more than QR-Code), and also to estimate the camera pose with respect to the marker. The one in Figure is an active bokode which uses a red LED as light source, but MIT Media Lab has created also a passive Bokode which can work without any LED and power source. Passive Bokode substitutes the LED with a retroreflector and uses the camera flash as the illumination source. Despite its really good performances and its innovativeness, it is clear that this marker is not suitable for our indoor localization system with visual markers deployed onto the

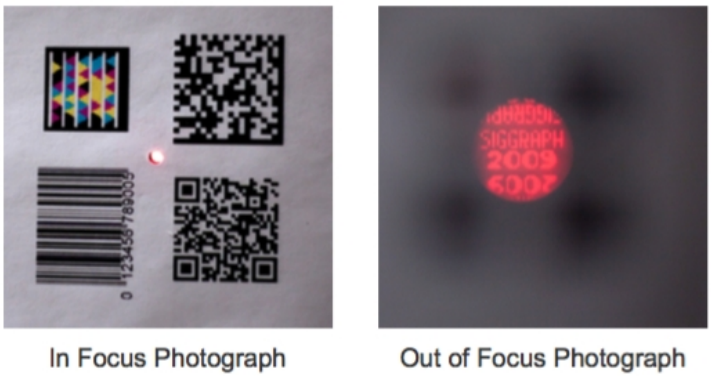

Figure 4.10: *In focus and out of focus bokode*

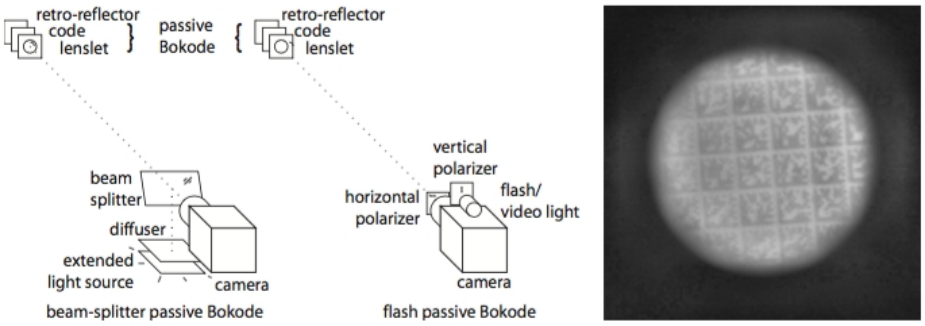

Figure 4.11: *passive bokode*

floor.

4.4.4 Other visual markers

The list of 2D visual markers, both patented and used for commercial purposes, or opensource, is actually very long, and required an accurate state of the art study. In fact, today visual markers are a consolidated and widely used technology and each marker has its own features which best suits to specific places of deploy, its strengths and its weaknesses. Understand deeply these characteristics was fundamental for choosing the correct visual marker for my specific situation. I will close this chapter by briefly listing some other markers:

- *Aztec code* (Figure 4.12a) [34] is a barcode invented by Andrew Longacre Jr. and Robert Hussey in 1995. It is similar to the QR-Code (large amount of stored data, Reed-Solomon error correction algorithm, public domain) but, differently from it, does not need a white border to be correctly decoded, so has the potential to use less space than other matrix barcodes.

- *ARTag* [35],[36] (Figure 4.12b) is a 2D marker (based on Artoolkit) which was invented in 2004 by Mark Fiala (Computational Video Group Institute for Information Technologies, National Research Council Canada) to robustify the original ArtoolKit marker system for what concern false positive detections and inter-marker confusion. It did this by replacing the correlation step with a digital symbol method. ARTag has a low and quantifiable error rate, reduced processing time, and can encode up to 2046 different IDs (it doesn't use patterns as in Artoolkit).

Figure 4.12: *Aztec code, ARTag, Intersense*

ARTag is supported by the open source Goblin XNA software [37].

- Other than the classical square markers we have also circular markers, which are stronger to perspective distortion and more precise; *Intersense* [38] (Figure 4.12c), is a commercial, patented, circular marker with high performances.

In the next chapter we will focus on three visual markers which best fit the requirements of our indoor localization system: Vuforia Marker, ArUco marker and AprilTag marker.

CHAPTER

FIVE

REAL TIME VISUAL MARKERS

The analysis of the visual markers state of the art brings us to restrict the choice of the best one that fits our requirements (first of all, the detection speed) to three possible candidates, which are shown in Figure 5.1. We have chosen these markers also because they can be freely used through opensource, well-documented libraries (AprilTag and ArUco) or free SDKs (Vuforia) and they are portable to all the major platforms. In this chapter, we give an overview of their features, strengths and weaknesses.

For what concerns our solution, the accuracy of the markers system is directly related to the number of times the marker is correctly decoded, under different conditions. We tested the markers in light (Figure 5.2a), medium (Figure 5.2b) and dark floor pattern (Figure 5.2c), in various light conditions, and for various distances camera-device and angle of scanning. In order to facilitate the detection we added a little white border around the markers. The tests were performed

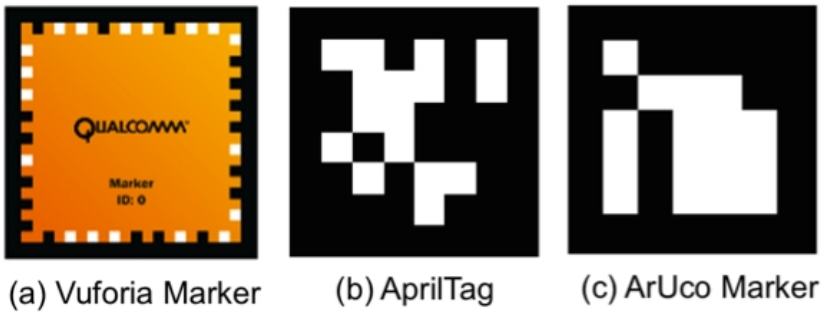

Figure 5.1: *Vuforia Frame marker, AprilTag, ArUco Marker*

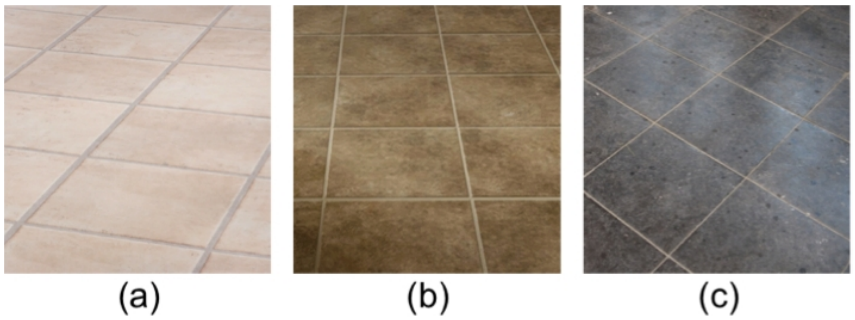

Figure 5.2: *(a) Light floor pattern, (b) Medium floor pattern, (c) Dark floor pattern*

with an iPhone 5S.

5.1 Vuforia Marker

Vuforia [39] is an augmented reality multi-platform SDK developed and maintained by Qualcomm. It is very powerful and offers to the developers a lot of functionality such as objects recognition, images recognition, shapes and text recognition. Moreover, the Vuforia SDK

can detect and estimate the pose (respect to the camera) of a special visual marker called Frame Marker (Figure 5.1a), which we can use for our indoor localization purposes. There are 512 frame markers, which are not generated by the application but are distributed as an archive. Each one encodes an ID (an integer between 0 and 511) on the binary pattern along the border, and needs an area around it (at least twice as wide as the thickness of the frame marker border), free of graphical elements, with a good contrast respect to the black frame. It needs to be entirely visible on camera image to be detectable, so it has not any tolerance to partial occlusions. The internal part of the marker is not used by the algorithm so it is possible to put inside an image or a logo, which makes the marker more esthetically good looking than other ones (but it is important that the internal design uses a contrasting, bright images or patterns in order to not deteriorate the performances of the detection phase). Due to the fact that we cannot have access to the source code, it is impossible to go deeper on the algorithms used by the SDK. However, by analyzing the APIs, we can deduce that (1) it is possible to set the size of the marker in the scene, (2) the markers are defined in the native code so, through a special class called *MarkerTracker*, the developer can create and destroy frame markers dynamically, and (3) there are three settings regarding the performances of the marker detection/decoding:

- *Mode-Optimize-Speed:* this provides a lower resolution (often 640×480, depending on the device) in order to achieve a higher frame rate and faster detection.

- *Mode-Optimize-Quality:* this provides a significantly higher resolution but a lower frame rate and a slower detection.

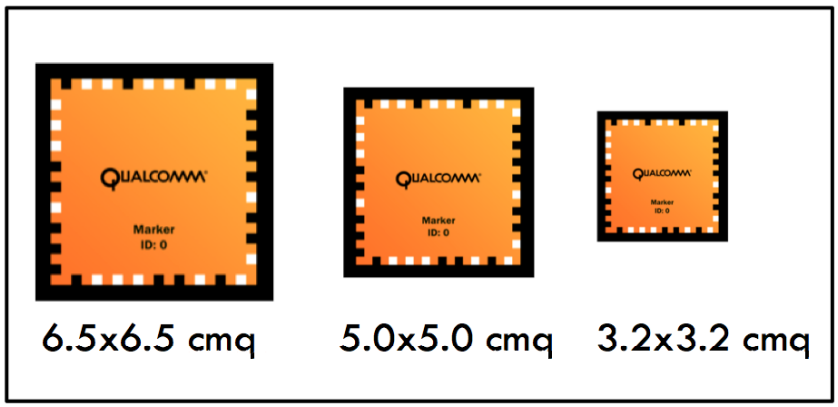

Figure 5.3: *Different marker sizes used for tests*

- *Mode-Default:* typically is equivalent to Mode-Optimize-Speed.

We performed some detection/decoding tests for different sizes of the marker: $(6.5 \times 6.5)cm^2$, $(5.0 \times 5.0)cm^2$, $(3.2 \times 3.2)cm^2$ and different distances marker-camera ($80cm$, $100cm$, $120cm$), in movement and with the smartphone in the palm of the hand, by set the *Mode-Optimize-Quality* option (Figure 5.3 and Figure 5.4). We repeated the tests in several lighting conditions.

5.1.1 Tests results

Table 5.1, 5.2 and 5.3 show the results of our tests for a distance marker-camera of,respectively, 80, 100 and $120cm$. We tried to detect/decode the marker, 30 times for each size, light condition and distance marker-camera. Table 5.4 gives a qualitative overview on the obtained results.

Our analysis shows that a marker size of $(6.5 \times 6.5)cm^2$ and

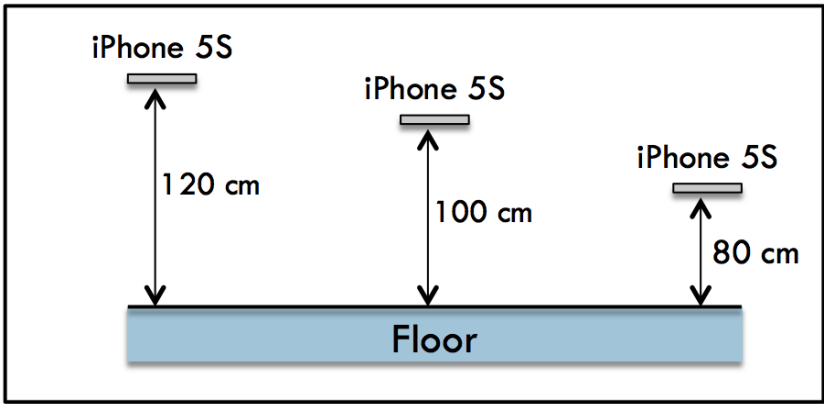

Figure 5.4: *Different distances marker-camera, used for tests*

30 Scans Dist. 80cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	26	4
Poor	26	4	22	8	0	30

Table 5.1: *Tests results on 30 total scans and 80cm distance marker-camera, in different light conditions, for different sizes of the Vuforia marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

30 Scans Dist. 100cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	24	6
Average	30	0	30	0	22	8
Poor	26	4	22	8	0	30

Table 5.2: *Tests results on 30 total scans and* 100*cm distance marker-camera, in different light conditions, for different sizes of the Vuforia marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

30 Scans Dist. 120cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	19	11
Average	30	0	30	0	0	30
Poor	26	4	22	8	0	30

Table 5.3: *Tests results on 30 total scans and* 120*cm distance marker-camera, in different light conditions, for different sizes of the Vuforia marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

Marker size	$6.5 * 6.5cm^2$			$5.0 * 5.0cm^2$			$3.2 * 3.2cm^2$		
Distance (cm)	80	100	120	80	100	120	80	100	120
Good light	+	+	+	+	+	+	+	-	-
Average light	+	+	+	+	+	+	-	-	x
Poor light	-	-	-	-	x	x	x	x	x

Table 5.4: *Qualitative evaluation of Vuforia Marker performances: '+' indicates that the marker is always decoded. '-' indicates that sometimes the marker is not decoded. 'x' indicates that the marker is never decoded in the specified conditions.*

$(5.0 \times 5.0)cm^2$, give good real-time performances for light (Figure 5.2a), medium (Figure 5.2b) and dark floor pattern (Figure 5.2c) in good and average light conditions. The performances gradually get worse for poor light conditions and if we reduce the size of the marker to $(3.2 \times 3.2)cm^2$ (and increase the distance marker-camera).

In conclusion, despite of the quite good overall performances and the fact that the SDK is well-maintained by a big company such as Qualcomm, the system has some drawbacks: (1) the source code is not accessible, so it is impossible to modify the algorithms in order to exploit the features of the floor, (2) the number of markers is fixed, which brings to a low flexibility, and (3) it is not possible to reduce a lot the size of the marker.

5.2 ArUco Marker

ArUco is a square visual marker realized by the AVA group from the University of Cordoba [40]. It can be decoded through the ArUco

library, which is cross platform (because openCV based), and open-source (BSD license). The library is written in C++, but it has a Java version and Python version available. Moreover, it seems to be well-maintained by the research group (last update was in 30-10-2014). Differently from other similar systems, ArUco does not provide a predefined set of markers: it is possible to generate the desired number of markers, with the desired number of bits n encoded inside each of them. The library maximizes the inter-marker distance (in order to avoid that few erroneous bits in the detection lead to a wrong, but valid, marker), the number of bit transitions (so there is less probability to confuse the marker with objects inside environments) and - based on the dictionary of generated markers - proposes an error correction algorithm which lets to correct a number of errors greater than the current state of the art. It is also possible to estimate the pose of the marker with respect to the camera. To be detectable, an ArUco marker must be entirely visible on camera image, but it is possible to manage the occlusion by using ArUco markers board. Since ArUco does not have a fixed number of bits, the performances of the detection/decoding algorithm vary depending on this parameter, which can be set according to the requirements of our use case: small areas can be covered with few markers, which means that the n can be reduced, which in turn brings to a faster detection/decoding phase.

As before for Vuforia, we performed some detection/decoding tests for different sizes of the markers (same sizes of Vuforia markers) and different distances marker-camera, under the same conditions. We chose to generate 512 AruCo markers, with $n = 4$. We also set the capture resolution to 640×480, and the focus mode to an optimal value. We repeated the tests in several lighting conditions, and in

30 Scans Dist. 80cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	30	0
Poor	30	0	30	0	26	4

Table 5.5: *Tests results on 30 total scans and* $80cm$ *distance marker-camera, in different light conditions, for different sizes of the ArUco marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

three types of floor: light floor pattern (Figure 5.2a), medium floor pattern (Figure 5.2b) and dark floor pattern (Figure 5.2c).

5.2.1 Tests results

Table 5.5, 5.6 and 5.7 shows the results of our tests for a distance marker-camera respectively of 80, 100 and $120cm$. We tried to detect/decode the marker, 30 times for each size, light condition and distance marker-camera. Table 5.8 gives a qualitative overview on the obtained results.

Our tests show that ArUco works very well, in any light conditions for a marker size of $(6.5 \times 6.5)cm^2$ and $(5.0 \times 5.0)cm^2$ with a distance marker-camera of $80cm$, $100cm$ and $120cm$ in any type of floor pattern. The performances get a little bit worse (but better than Vuforia) if we reduce the size of the marker to $(3.2 \times 3.2)cm^2$ or increase the distance marker-camera to $120cm^2$, for average and poor light condi-

30 Scans Dist. 100cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	30	0
Poor	30	0	30	0	24	6

Table 5.6: *Tests results on 30 total scans and* 100*cm distance marker-camera, in different light conditions, for different sizes of the ArUco marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

30 Scans Dist. 120cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	19	11
Poor	30	0	30	0	0	30

Table 5.7: *Tests results on 30 total scans and* 120*cm distance marker-camera, in different light conditions, for different sizes of the ArUco marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

Marker size	$6.5 * 6.5cm^2$			$5.0 * 5.0cm^2$			$3.2 * 3.2cm^2$		
Distance (cm)	80	100	120	80	100	120	80	100	120
Good light	+	+	+	+	+	+	+	+	+
Average light	+	+	+	+	+	+	+	+	-
Poor light	+	+	+	+	+	+	-	-	x

Table 5.8: *Qualitative evaluation of ArUco Marker performances: '+' indicates that the marker is always decoded. '-' indicates that sometimes the marker is not decoded. 'x' indicates that the marker is never decoded in the specified conditions.*

tions. ArUco source code is accessible for the developer: thanks to this, it is possible to modify the algorithms in order to adapt them to the scenario described in Chapter 4. Also the possibility to set the number of markers and bits increases a lot the flexibility of the system. In conclusion, ArUco is a good choice for an indoor localization system with visual markers deployed onto the floor, when the requirements are flexibility and real-time performances.

5.3 AprilTag

AprilTag is a square visual marker developed for robotic applications by Edwin Olson, in the April Robotic Laboratory at University of Michigan [41]. The opensource library lets to detect an AprilTag in an image, decode the ID of the marker, and estimate its 3D pose and orientation respect to the camera. The library is written in pure C with no external dependencies, and appears to be well-documented and well-maintained (last version update was 20-10-2014), robust to

changes in light conditions and view angle, and with real-time performances.

We performed some detection/decoding tests by choosing the recommended pre-generated markers family 36h11 (36 bit markers with minimum hamming distance between codes of 11), which consists of 518 different markers, and by using the same marker sizes, marker-camera distances and conditions of the previous Vuforia and ArUco cases. For the initial tests, we used the AprilTag iOS application, developed by Edwin Olson and available on the US Apple Store for free. The application lets the user to set some parameters:

- *Decimation (*1-4*):* it lets to reduce the resolution of the analyzed image.

- *Refine Tag Positions (On/Off):* if it is set to On, the algorithm spends more time trying to precisely localize tags.

- *Refine Tag Decodes (On/Off):* if it is set to On, the algorithm spends more time trying to decode tags.

- *Camera Focus (from* 0 *to* 1*):* it lets to arbitrarily set the focus to a given value.

Since the most important requirement for our scenario is the detection/decoding speed, we set both Refine Tag Positions and Refine Tag Decodes options to Off value, the Camera Focus to the optimal value for our scenario and the decimation to the maximum value.

30 Scans Dist. 80cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	30	0
Poor	30	0	30	0	30	0

Table 5.9: *Tests results on 30 total scans and* $80cm$ *distance marker-camera, in different light conditions, for different sizes of the AprilTag marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

5.3.1 Tests results

Table 5.9, 5.10 and 5.11 show the results of our tests for a distance marker-camera of, respectively 80, 100 and $120cm$. We tried to detect/decode the marker, 30 times for each size, light condition and distance marker-camera. Table 5.12 gives a qualitative overview on the obtained results.

The results show that AprilTag works very well in all tested light conditions and for almost all tested sizes and marker-camera distances, in any type of floor. The availability of the source code (which lets the developer to modify the algorithms in order to adapt them to the floor features), the speed of the system and the small marker size make AprilTag the best choice for an indoor, marker-based localization system when the flexibility about the number of markers is not required.

30 Scans Dist. 100cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	30	0
Poor	30	0	30	0	30	0

Table 5.10: *Tests results on 30 total scans and* 100*cm distance marker-camera, in different light conditions, for different sizes of the AprilTag marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

30 Scans Dist. 120cm	**Marker size 6.5x6.5 cmq**		**Marker size 5x5 cmq**		**Marker size 3.2x3.2 cmq**	
LIGHT	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Good	30	0	30	0	30	0
Average	30	0	30	0	30	0
Poor	30	0	30	0	25	5

Table 5.11: *Tests results on 30 total scans and* 120*cm distance marker-camera, in different light conditions, for different sizes of the AprilTag marker. 'Yes' means the marker was correctly decoded, 'No' means the marker was not decoded or incorrectly decoded.*

Marker size	$6.5 * 6.5cm^2$			$5.0 * 5.0cm^2$			$3.2 * 3.2cm^2$		
Distance (cm)	80	100	120	80	100	120	80	100	120
Good light	+	+	+	+	+	+	+	+	+
Average light	+	+	+	+	+	+	+	+	+
Poor light	+	+	+	+	+	+	+	+	-

Table 5.12: *Qualitative evaluation of AprilTag Marker performances: '+' indicates that the marker is always decoded. '-' indicates that sometimes the marker is not decoded. 'x' indicates that the marker is never decoded in the specified conditions.*

CHAPTER

SIX

LOCALIZATION SYSTEM: SERVER SIDE

The final step of our study was the design of an indoor localization architecture as simple as possible (but flexible enough to guarantee future updates) both for the person who will deploy the system, and for the final users as well. In order to guarantee these requirements, we split the architecture in a Server side and Client side. The Server side is mainly responsible for presenting an easy-to-use UI to the indoor system administrator, which lets him to make the whole system working. Moreover, it exposes Rest APIs to the client, in order to exchange data with it. The Client side (among other things) is responsible for: (a) communicating with the Server side in order to download data about a specific building, and (b) letting the user to navigate inside that building by running indoor localization algorithms, and orienting himself through pathfinding algorithm.

In the following, after introducing some concepts related to the system, we will present the server side of the architecture. In the next

chapter we will show the client side.

6.1 Path-Points

If we look at the most of the indoor maps of buildings, we can see that usually the user is forced to walk in predefined paths such as corridors, aisles etc. Considering this, and the fact that each indoor localization system is affected by errors, is pretty useless (and also, not good from a user experience point of view) to track the user's path, point by point. Figure 6.1 (which represents the ground floor's indoor map of the building 13 in our academic campus in Catania) shows this condition. Tracking the user, point by point, leads to an irregular path without adding any information content: the user is walking along the corridor, and probably he just wants to reach a specific room. Moreover, it leads to an higher computational complexity due to the big space of the possible positions.

In the proposed approach the indoor system administrator (through the user interface) must set on the map a certain number of points that we define *path-points*: these points, which can be represented by an array of coordinates (x, y) as in 6.1, discretize the walk-over area.

$$[(x_0, y_0), (x_1, y_1), ...(x_n, y_n)] \tag{6.1}$$

Any user location on the map will be converted to one of these points. In particular, the chosen path-point will be the one with the minimum euclidean distance (equation 6.2) from the estimated point.

$$d = \sqrt[2]{[(x - x_0)^2 + (y - y_0)^2]} \tag{6.2}$$

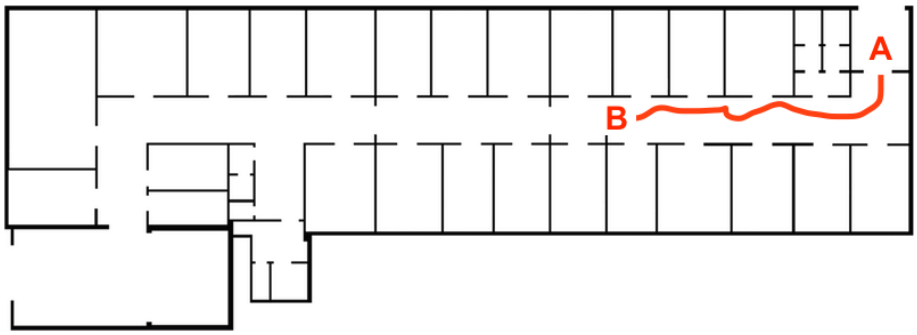

Figure 6.1: *ground floor's indoor map of the building 13 in academic campus in Catania. Example of irregular path caused by the errors in the localization process.*

To be more clear, if the path-points array is $[(30, 30), (30, 40), (30, 50)]$ and the calculated position is $(30, 33)$, by applying the equation 6.2 we obtain $d = 3$ for the path-point $(30, 30)$, $d = 7$ for the path-point $(30, 40)$, and $d = 17$ for the path-point $(30, 50)$. The chosen path-point will be thus $(30, 30)$. The Figure 6.2 explains better the concept. The set of path-points is packaged in a JSON structure and sent to the client, together with other data.

Listing 6.1: *JSON structure sent to the client for what concerns the set of path-points*

```
        .........
PathPoints:[
    {
        x: 30,y: 30,
                layers:{
                        ProfessorsRooms :"Mario Rossi"
```

```
            }
        },
        {
            x: 30,y: 40,
                layers :{
                        ProfessorsRooms :"Mario Rossi"
                }
        }
        .........
        ]
```

The JSON structure in Listing 6.1 shows the details of what it is sent to the client. Specifically, we sent to the client the coordinates of all the path-points; To each path-point we can associate one or more *informative layers* (the layer ProfessorsRooms in the case on Listing 6.1) and the area (in the correspondent layer) the path-point belong to (The Mario Rossi room). The concept will be more clear in the next paragraph, where we will explain more about the layers.

6.2 Informative Layers

During the design of the indoor localization app, we tried to think about what the user wants from an indoor navigation system. Of course he needs his position in an indoor map, but probably he needs also a way to reach a specific point of interest. And, often, he has only partial information about the place he wants to reach, sometimes not directly connected to the physical place: for example, he wants to meet the Professor X, or, in a shopping center, finds where are the

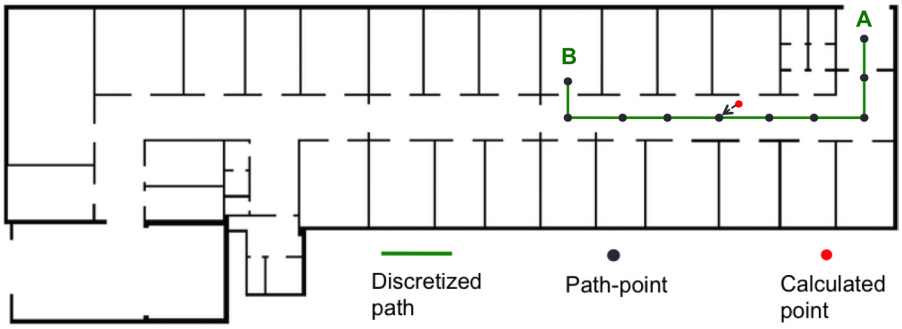

Figure 6.2: *ground floor's indoor map of the building 13 in academic campus in Catania. Example of discretized path using path-points.*

computers. In order to make the system flexible enough to support these use cases, we introduced the layers:

- The *Basic Layer* is essentially the planimetric of the building. It is composed by the rooms (identified by names), the corridors and the walls. Using this layer, the user can navigate from a point $A = (x_0, y_0)$ to a point $B = (x_1, y_1)$.
- The *Informative Layers* are additional layers that we can superimpose on the basic layer to have more information about the specific building's floorplan. For example, it is possible to have the layer *ProfessorsRooms* which superimposes the names of the professors who work in all the rooms plus a list of associated metadata for each professor, which can be used by the pathfinding algorithm in the client side to help the user reaching the desired location.

The set of informative layers associated to a specific building's floor is packaged in a JSON structure and sent to the client, together with

other data.

Listing 6.2: *JSON structure sent to the client for what concerns the set of informative layers*

```
        .........
layers :{
        ProfessorsRooms: {
                "Mario Rossi" :{
                        phone: 095112233
                        subject: "computer science"
                },

                "Giuseppe Verdi" :{
                        phone: 095445566
                        subject: "electronic"
                }
        .........
}
```

The JSON structure in Listing 6.2 shows what it is sent to the client. Specifically, we sent to the client the list of all the informative layers associated with the building's floor (in the structure 6.2 we only have one informative layer, named *ProfessorsRooms*, with two rooms, Mario Rossi's room and Giuseppe Verdi's room. Two type of metadata are added in each room: *phone* and *subject*). Of course, the indoor system administrator through the user interface can easily add/remove as much informative layers as he needs.

The Figure 6.3 shows the basic layer of the building 13 in academic campus in Catania (the rooms are named with the letters of the al-

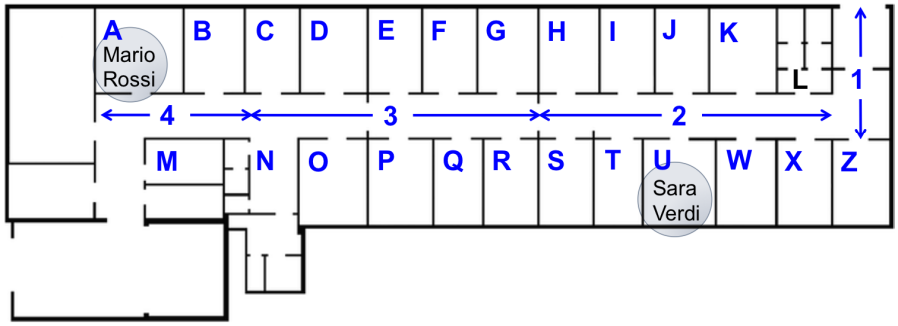

Figure 6.3: *ground floor's indoor map of the building 13 in academical campus in Catania with the informative layer "ProfessorsRooms" superimposed on it.*

phabet), with, superimposed, the informative layer *ProfessorsRooms* which shows that in room *A* works the professor Mario Rossi and in room *U* works the professor Sara Verdi (we omitted all the other professors' names).

It is thus possible for the user of the app, to search for professor Mario Rossi and navigate from his position to Mario Rossi's room without any previous information about the room name of the professor Mario Rossi.

6.3 Markers deployment

Based on the analysis we made on Chapter 5, we chose AprilTag as preferred marker: it has excellent real time performances even in poor light conditions and for small size of the marker. The indoor system administrator has the responsibility to place a set of AprilTag markers both virtually onto the indoor map, and physically onto the floor of

the building, in correspondent positions (with the constrain that each marker must be coincident with one of the path-points). Each marker has an associated ID which can be decoded by the client through the AprilTag library. The association between the ID and the coordinates on the map lets to perform the indoor localization.

The JSON structure in Listing 6.3 shows what is sent to the client. Specifically, we sent to the client the list of all the aprilTag markers IDs, and the associated coordinates on the map.

Listing 6.3: *JSON structure sent to the client for what concerns the list of aprilTag markers*

```
        .........
"tags-coordinates":[{
                      "idTag":0,
                      x: 40,
                      y: 50
                      },
                      {
                      "idTag":1,
                      x: 40,
                      y: 70
                      },
                      {
                      "idTag":2,
                      x: 40,
                      y: 100
                      },
                      .........
```

```
                    {
                    "idTag":30,
                    x: 200,
                    y: 150
                    }
                    ]
.........
```

6.4 Serve side structure

The Figure 6.4 shows how we logically split the server side part of the system.

- The *User Interface Logic* block is responsible for managing everything related to the UI. It gets the input from the indoor localization system administrator and sends it to the Server Side Logic. In particular, the User Interface Logic lets the administrator to: (a) insert a new building in an outdoor map (name, address, brief description); insert building floors data (name, floor's level, description, indoor floor map, scale); (c) add or edit floor's information (path-points, AprilTag markers, layers data etc.).

- The *Server Side Logic* block is responsible for managing all the data coming from the administrator's input. It elaborates the input information and generates all the data structures which the client needs. These data structures are then sent to the Data Exchange Logic.

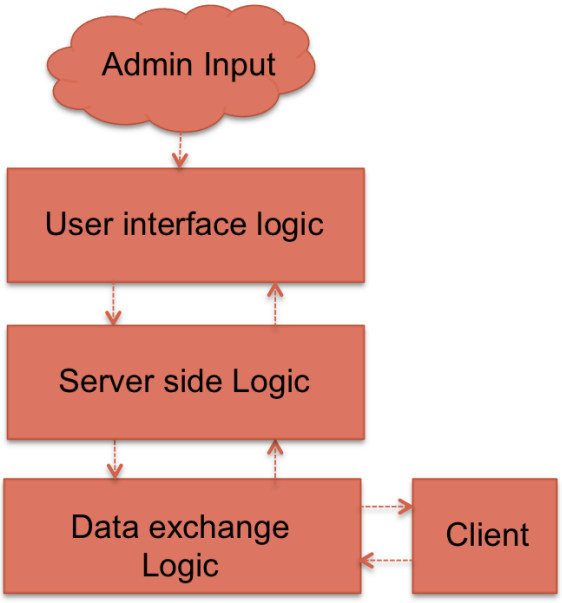

Figure 6.4: *Overview of the server side structure*

- The *Data Exchange Logic* block is responsible for exchanging data with the client through REST APIs. We will see more in detail the client side in the next chapter.

CHAPTER

SEVEN

LOCALIZATION SYSTEM: CLIENT SIDE

As we stated in the previous chapters, our smartphone-based indoor localization system relies on a simple hypothesis: when the user wants to navigate inside an unknown building, he has the phone in the palm of the hand and with the back camera directed toward some part of the floor. Given this, the first basic idea was to deploy onto the floor a visual markers system and use computer vision algorithms to decode such markers and estimate the position of a user. We performed some simple tests using QRCodes, and started thinking to the disadvantages of this approach. One of the main problems was the invasiveness of the system, under different points of view: too many markers to deploy onto the floor (in order to have a good accuracy), and too big marker's size needed for a properly detection. Moreover, the non real time performances, made the approach too complicated from a user point of view. These observations led us to the study of the state of the art for what concerns the visual markers. Our purpose was to find a

visual markers system which was: (a) opensource, (b) with real time performances and (c) capable to decode markers with a small size. We evaluated a lot of these markers, and, finally, for the reasons we saw on chapter 5, we chose the AprilTag marker. This kind of marker solved two of the problems we found, but still, the system was quite invasive, due to the high number of markers that we needed to deploy onto the floor.

The second idea was then to use the inertial navigation for tracking the user (dead reckoning): even if this kind of approach produce an high drift error introduced by the sensors (which made it unusable after few seconds), we could track the user by using it, and rely on the aprilTag markers for resetting such cumulative error before it becomes too high. To improve the final result, we opted for using a pedometer instead of the raw data coming from accelerometers. Moreover in order to make simpler the prototype, and focus on the approach, we avoided to integrate the motion direction estimation through gyroscope and compass in the final app. Instead, we manually indicate the direction of motion by tapping on the screen.

The preliminary raw results were quite good so we started to design a client-server prototype in order to better test the approach.

In the previous chapter we saw the server side of the system. In this chapter we will see the more important client side, where the indoor localization logic is implemented.

7.1 Swift

For the development of the prototype we chose the Apple platform, with operating system iOS7 or higher. In particular, the device used for tests was an iPhone 5S. During the prototype's development phase, we encountered some problems due to the introduction by Apple of a completely new programming language called *Swift* (it is still under development) in substitution of the "old" Objective-C language, and to the upgrade of the IDE XCode from version 6 to version 7, which have forced us to several "fixes" on the code to let the application working properly.

Swift is a modern programming language (which has required years of work) with:

- A very clear syntax which makes the code easy to write, flexible, safer than Objective-C and with an high readability.
- A totally object oriented model: everything in Swift is an object.
- Support for the frameworks Cocoa and Cocoa Touch, enhancement on the compiler and the debugger.
- Strongly typed.
- Automatic memory management (ARC, automatic Reference Counting).

These are only some of its main features, which make it perfect for developers who come from another programming language. Moreover, thanks to the *"mix and match"* feature, Swift is fully compatible with Objective-C, which means that is possible (through a special bridging

header file) to write apps that have a mixed-language codebase, and thus guarantee the compatibility with (a) old libraries written before the Swift introduction and (b) new libraries written in C language, such as the aprilTag library we used in our project. By looking the attention Apple is reserving to Swift, it is clear that Objective-C will be gradually abandoned: because it is a subset of the C language, it is just partially object oriented (it has both objects and scalar types, and objects must be inserted inside C pointers). In addition, it has a complex syntax and is less typed than Swift (which often lead to errors).

For these reasons, the indoor localization prototype was developed in Swift, with some Objective-C libraries incorporated in the project, and some Objective-C code integrations (using the bridging header file) when they are needed.

7.2 Used Libraries

To not reinvent the wheel, before we started coding we searched for opensource libraries which could be useful for our project. In the following, we briefly talk about these libraries.

1. **SwiftyJSON**: because the data exchange between server and client is made through JSON structures, we searched for a library capable to manage these structures in an easy way, on the client side. SwiftyJSON has been the perfect one: it makes easy to deal with JSON in Swift. It is released under the permissive MIT license [42].

2. **PathFindingForObjC**: this library is used to let the user

searching for a specific path. It implements a lot of pathfinding algorithms such as *A**, *First Search*, *Dijkstra*, *Jump Point Search* etc [43]. As specified in the library name, it is written in Objective-C so we used the bridging header file for integrating it in the Swift project. It is released under the permissive MIT license [44].

3. **Alamofire**: it is a very powerful HTTP networking library written in Swift which leverages the iOS NSURLSession and URL Loading System in order to provide a simpler interface for networking operations. For example, to get a JSON object you just need to write the following code:

Listing 7.1: *get JSON object using Alamofire*

```
Alamofire.request(.GET, "http://address_of_server")
  .responseJSON {(request, response, JSON, error) in
            println(JSON)
 }
```

The library is released under the permissive MIT license [45].

4. **Progress HUD**: just a lightweight and easy-to-use HUD to inform the user when a specific view is loading, or send him other messages in a clean and user-friendly way[46].

5. **AprilTag**: this is the library used for decoding the aprilTag markers. It is the core part of the client side application.

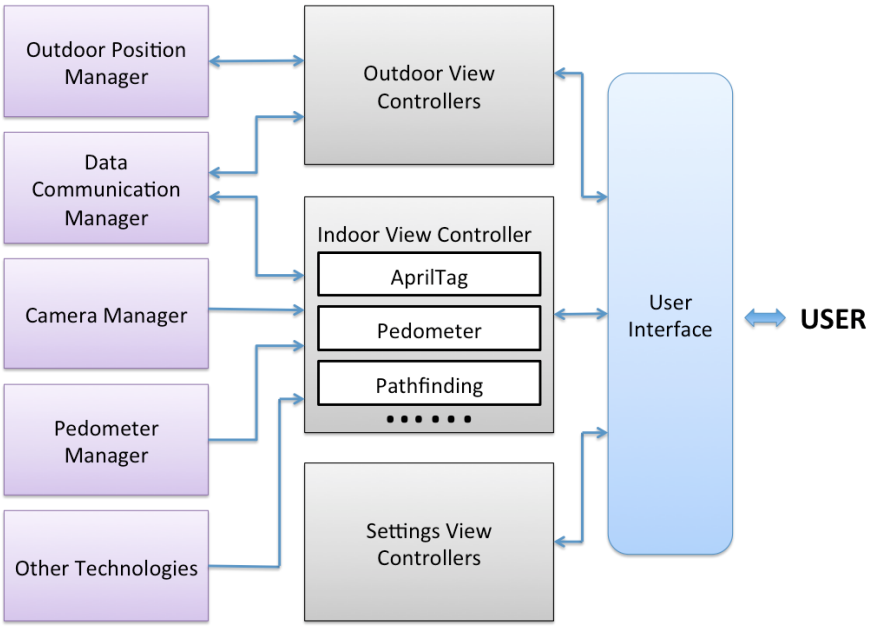

Figure 7.1: *Blocks diagram of the client side application.*

7.3 Structure of the app

In order to (1) build an iOS application, as modular as possible, which lets to add into the system other indoor localization technologies in an easy way; (2) maintain the code clean; (3) permit to reuse/modify the code with a little effort, we applied during the code development, the classical design patterns (Model-View-Controller, delegation, singleton) and designed the whole client application according to the blocks diagram in Figure 7.1.

In such diagram we can see that the structure is logically split into three main parts:

- *Managers:* these blocks are responsibles for dispatching information to the viewControllers. They manage the data exchange between client and server, collect data from device's hardware (GPS, CoreMotion chip and camera) and send it to the application.

- *ViewControllers:* these blocks, are responsible for the logic of the whole application. They get data from the managers and elaborate it in order to obtain the desired result.

- *User Interface:* this block is responsible for organizing the interaction between the user and the system.

7.3.1 Outdoor position manager

It Manages the outdoor position of the user. It uses the iOS CoreLocation framework's APIs to get the location fix from the device (typically, from the GPS chip or, if the GPS signal is not available for some reason, from the WiFi system or from the cell towers) and dispatch it to the *outdoor viewControllers* block. We embedded the code for doing this in the class *PositionManagerSingleton* which is a singleton: it means that there will be one and only one instance of the class, which can be accessible from various parts of the code. In Listing 7.2 we show the code for creating the singleton instance.

Listing 7.2: *Code for creating the singleton instance of PositionManagerSingleton class*

```
class var sharedInstance: PositionManagerSingleton {
       struct Static {
           static var instance: PositionManagerSingleton?
```

```
            static var token: dispatch_once_t = 0
        }

        dispatch_once(&Static.token) {
            Static.instance = PositionManagerSingleton()
        }
        return Static.instance!
    }
```

Two methods inside the aforementioned class let to start/stop the CoreLocation *CLLocationManager* object, which dispatches location fix and notifies it to the objects which need such information through the iOS notification center. The information about the user position is used by the *Outdoor ViewControllers*.

7.3.2 Data communication manager

It is the block which manages the communication with the server through REST APIs. For what concerns the "outdoor side" of the app, it downloads from the server the list of buildings (in a specific area) with an available indoor map, and passes it to the *Outdoor ViewControllers*. For what concerns the "indoor side" of the app, it is responsible for exchanging the JSON structures we saw in Chapter 6. We used Alamofire library for exchanging JSON structures with the server side of our indoor localization platform.

7.3.3 Pedometer manager

This block relies on the iPhone 5S M7 coprocessor (M8 and M9 for iPhone 6 and iPhone 6S) to get the number of steps the user did in a

time interval and dispatch these data to *Indoor ViewControllers*. M7 coprocessor is accessible to the applications through the CoreMotion APIs which were introduced with iOS7 version of the Apple operating system. CoreMotion makes available the *CMPedometer* class which provide properties and methods to interface with the chip and gives very accurate motion data. We embedded the code for managing the user steps in the class *DeadReckoningSingleton* which is a singleton. The code for creating the singleton instance is almost the same as on listing 7.2. In listing 7.3 we show the methods to start/stop acquiring the pedometer data.

Listing 7.3: *Code to start/stop of the pedometer*

```
let pedometer: CMPedometer=CMPedometer()
typealias AAPLStepUpdateHandler=(pData:CMPedometerData)->Void

//MARK: 1 - start pedometer
func startStepUpdates(handler:AAPLStepUpdateHandler) {
 pedometer.startPedometerUpdatesFromDate(NSDate(),
 withHandler: { pedometerData, error in
      if (error != nil) {
         self.handleError(error)
          }
         else {
               dispatch_async(dispatch_get_main_queue(), {
                   handler(pData: pedometerData!)
               })
         }
    })
```

```
}
//MARK: 2 - stop pedometer
func stopStepUpdates() {
    pedometer.stopPedometerUpdates()
}
```

After starting the pedometer, the app receives data (number of steps) from the device as it becomes available, approximately every 2.5 seconds. The singleton object dispatches the number of steps to the objects which need such information using the handler *AAPLStepUpdateHandler*. The information about the user position is used by the *Indoor ViewControllers*.

7.3.4 Camera Manager

This block manages the start and stop of the smartphone camera, in order to acquire the frames from it and pass such frames to the *Indoor ViewControllers* for the elaboration. We embedded the code for managing the camera in the class *VideSessionSingleton* which is a singleton. The code for creating the singleton instance is almost the same as on listing 7.2. The class, basically has three methods, *setupCaptureSession()*, *startRunning()*, *stopRunning()*. The first one is used to set some important parameters for acquiring frames in a proper way: choose the back camera, set the resolution and the output format of the frames, set the focus mode etc. (listing 7.4).

Listing 7.4: *Setup of the capture session: mains blocks of code*

```
func setupCaptureSession() {
..................
```

```
//set the resolution of the acquired frames
captureSession.sessionPreset=AVCaptureSessionPreset1920x1080
.................
//video frames are dropped if they arrive late.
videoDataOutput.alwaysDiscardsLateVideoFrames = true
//set the pixel format type
videoDataOutput.videoSettings=[kCVPixelBufferPixelFormatTypeKey:
Int(kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange)]
//Restricts the autofocus to Far range
tempVideoDevice.autoFocusRangeRestriction =
AVCaptureAutoFocusRangeRestriction.Far
.................
}
```

The second and third ones are simply used to launch the camera session and stop it when the app is not being used (listing 7.5).

Listing 7.5: *Setup of the capture session: main blocks of code*

```
//Boolean variable to save the state of the capture session
var _running = false
//MARK: start Running
func startRunning() {
//if the session is already in running you don't need to
//run the startRunning
  if _running == true{
    return
  }
  captureSession.startRunning()
    _running = true
}
```

```
//MARK: stop Running
func stopRunning() {
//if the session is already in stop you don't need to
//run the stopRunning
  if !_running {
     return
  }
  captureSession.stopRunning()
  _running=false
}
```

The singleton object dispatches the frames to the *Indoor ViewController* using the iOS delegation mechanism.

7.3.5 Outdoor View Controllers

The main purposes of these controllers are to: (1) elaborate everything related to the outdoor position of the user; (2) send the results to the *user interface* block for display. Specifically when the app is launched, the class *MainViewController* receives the user position from the *outdoor position manager*. It uses this position to make a query on the server (through the *data communication manager*) and gets the list of all buildings (in a specific range) which have an available indoor map. The listing 7.6 shows the result of the query. The Figure 7.2a shows this result on the user interface.

Listing 7.6: *JSON structure which represents the list of buildings in a specific range*

```
..............
{"buildings": [{
```

```
            "id": 1,
            "name": "building 13",
            "mapURL": "http://address_of_image_map",
            "scale_px_mt": 20,
            "coordinates":[37.525664, 15.074474],
            },
            {
             other building....
            }
            ...................
            ]
}
```

The class *SearchBuildingViewController* lets to search for a specific building by name and send the results to the user interface (Figure 7.2b, 7.2c). We don't go deeper in this class.

7.3.6 Settings View Controllers

The main purposes of these controllers are: getting the user inputs for what concerns the settings of the app, and update these settings. Through these controllers is possible to set the range around the user position (in meters) where to search for buildings with an available indoor map, calculate the path to go from the current position of the user to a specific point of interest through the A* pathfinding algorithm, set the step length of the user, choose the informative layers which the app can download from the server etc. Moreover, it is possible to adjust some parameters about the aprilTag library in order to obtain an optimal performance of the aprilTag decoding algorithm.

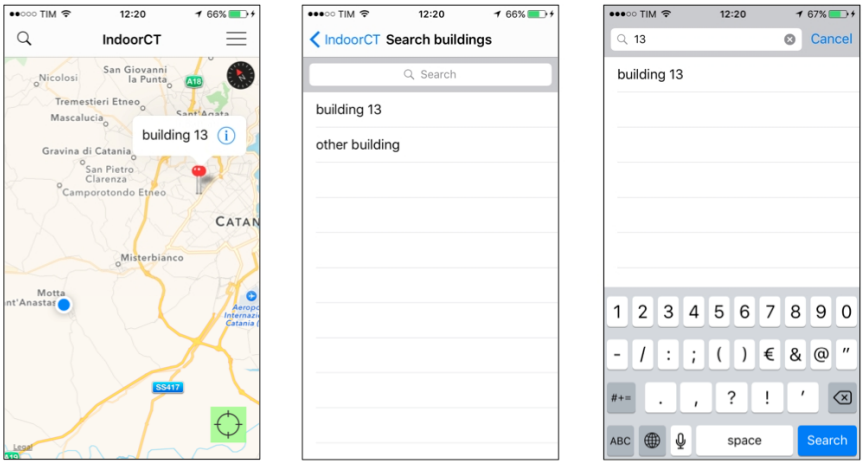

Figure 7.2: *Screenshots of the app. (a) View which contains the outdoor map of the area where the user is located. The pins represent the buildings which have an available indoor map. (b),(c) Views which let the user to search for a specific building.*

7.3.7 Indoor View Controller

After choosing the building of interest on the outdoor map by clicking on a specific pin, the *indoor View controller* block, through the *data manager* block, downloads the indoor map of the building with all the related data (path-points, layers, list of markers) in a JSON format, and loads it on the user interface (We already saw the JSON structures related to a building, which the server sends to the client, in chapter 6). Moreover, the *pedometer manager* block starts counting the steps of the user, and the *camera manager* block starts sending frames to the *indoor view controller* (which is registered as its delegate). The "job" of this controller, is to elaborate each frame which comes from camera using the aprilTag library; if an aprilTag marker is found, the controller converts its ID in a (x, y) position on the map, resets the number of steps, and sends such position to the user interface. Between two aprilTag markers, the user is tracked by using the data coming from the pedometer: the number of steps is multiplied for the step length and, using the information about the scale of the map, converted in an (x,y) position.

The listing 7.7 shows the initial setup for the aprilTag library: the aprilTag function *tag36h11_create()* lets to select the pre-generated *36h11* family of markers (composed by $6x6$ bits and an Hamming distance of 11; on Figure 7.3 we can see some example of markers belonging to this family, and other two families, *25h9* and *16h5*). The aprilTag function *apriltag_detector_create()* creates the detector object and sets the default parameters for detecting/decoding the markers (as we saw in chapter 5). Of course it is possible to modify these parameters in order to get the optimal results for our situation of

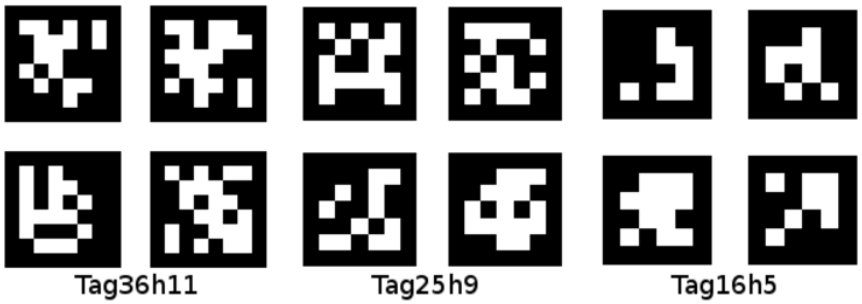

Figure 7.3: *Examples of markers belonging to the aprilTag families 36h11, 25h9, 16h5.*

deploy. The aprilTag function *apriltag_detector_add_family()* adds the previously created family to the aprilTag detector.

Listing 7.7: *Setup of the capture session: main blocks of code*

```
..............
let family: UnsafeMutablePointer <apriltag_family_t>
let detector: UnsafeMutablePointer <apriltag_detector_t>
.............
required init?(coder aDecoder: NSCoder) {

        //1 - choosing the 36h11 family of aprilTag markers
        family = tag36h11_create()

        //2 - creation of the detector
        detector = apriltag_detector_create()

        //3 - add the chosen family to the detector
        apriltag_detector_add_family(detector, family)
```

```
        super.init(coder: aDecoder)
}
```

The listing 7.8 shows the signature of the delegate method (which is located inside the *indoor View Controller*) where the *camera manager* dispatches the frames coming from the camera:

Listing 7.8: *method signature for video capture delegate*

```
func captureOutput(captureOutput: AVCaptureOutput!,
didOutputSampleBuffer sampleBuffer: CMSampleBuffer!,
fromConnection connection: AVCaptureConnection!) {
..............
..............
}
```

Inside this method, we perform the frame's elaboration and integrate the results with the data coming from the pedometer. The first step is to convert the frame in the format the aprilTag library needs in order to work properly (listing 7.9):

Listing 7.9: *The frame is converted in the format which the aprilTag library needs*

```
............
let imgToDecode: UnsafeMutablePointer<image_u8_t>
                 = image_u8_create(width, height)
imgToDecode.memory.stride = strideGray
imgToDecode.memory.buf = baseAddressInt8Gray
............
```

The second step (listing 7.10) consists in: (a) searching for a marker inside each frame and if we found it, decoding its ID. (b) Converting

the ID in a couple of coordinates (x, y) on the reference system of the map. (c) Getting the closest path-point to such coordinates: this point represents the estimated position of the user. (e) Resetting the collected number of steps. If there is not a marker inside the frame, we continue to look for markers in the subsequent frames, and in the meantime we track the user by converting the number of steps from the last time we reset it, in a couple of coordinates (x, y) on the reference system of the map and, as before, by getting the closest path-point to such coordinates which will represent the estimated position of the user.

Listing 7.10: *Conversion of the information coming from camera and pedometer in a position on the indoor map*

```
.............
var nTags:CInt = 0
let arrayOfTags =
apriltag_detector_detectTAGS(detector,imgToDecode,&nTags)

//There is at least one marker
if numberOfTags>0 {
  //take the first tag
  let idTag = Int(arrayOfTags[0].memory.id)
  var x,x_pathPoint = CGFloat()
  var y,y_pathPoint = CGFloat()

  //searching in the array of markers for the correct ID
  for var tag = 0; tag<building.tagsList.count; tag++ {
   let idTagOnList = building.tagsList[tag]["idTag"]!
   if idTagOnList == idTag {
```

```
    //get the coordinates associated with the IDs
    x = CGFloat(building.tagsList[tag]["x"]!)
    y = CGFloat(building.tagsList[tag]["y"]!)
    x_pathPoint = getClosestPathpointX()
    y_pathPoint = getClosestPathpointX()

    //reset the number of steps
    stepsBetweenMarkers = 0
    prevNumberOfSteps = currNumberOfSteps
   }
   else {
    stepsBetweenMarkers=
        currNumberOfSteps-prevNumberOfSteps

    currentNumberOfSteps = getDataFromPedometer()
    x = getXPositionFrom(currentNumberOfSteps,scale)
    y = getYPositionFrom(currentNumberOfSteps,scale)
    x_pathPoint = getClosestPathpointX()
    y_pathPoint = getClosestPathpointX()
   }
  }
  //UI updated with the estimated position
  //(x_pathpoint,y_pathpoint)
  ............
}
```

The Figure 7.4 shows how the results of the whole indoor localization process are displayed on the user interface. The blue circle on the screenshot 7.4a is placed in a (x, y) position on the map correspondent to a specific aprilTag marker ID. With the green circles (screenshot

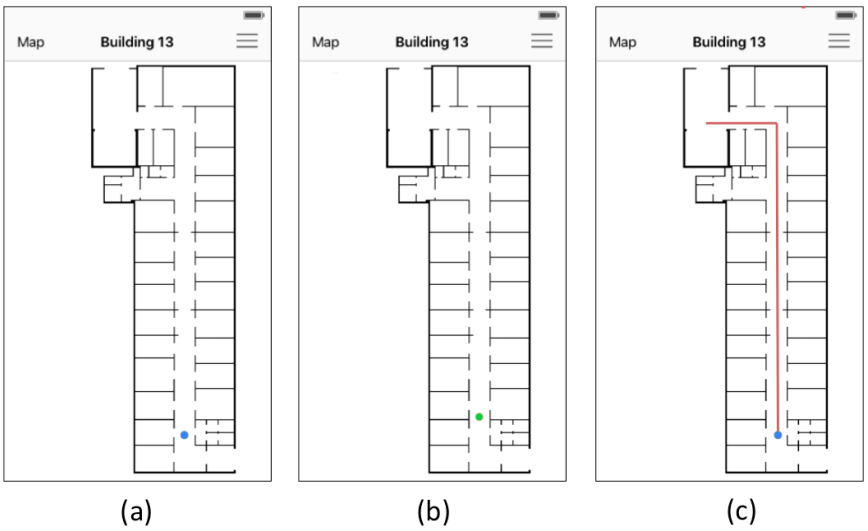

Figure 7.4: *Screenshots of the app. (a) Position correspondent to a specific marker ID. (b) Position estimated through the pedometer. (c) Calculated path, to go from the current user position to the exit of the building*

7.4b) we track the user between a couple of aprilTag markers using the pedometer. The screenshot 7.4c shows the path (calculated using the A* pathfinding algorithm and by considering the path-points) to go from the current user position to the exit of the building.

CHAPTER

EIGHT

RESULTS

We tested our prototype in the building 13 in academic campus in Catania. The Figure 8.1 shows the GPS coordinates on the map, where the building is located. We placed on the site and on the indoor map of the building, 9 aprilTag markers, from $ID = 0$ to $ID = 8$; in the process of deploying the visual marker system, we chose strategic points onto the floor (such as in correspondence of doors), with a maximum distance between markers of about 8 meters. The Figures 8.2a, 8.2b show the floor with some markers deployed on it, while the Figures 8.3a, 8.3b show the test path and the indoor map of the building with the markers deployed on top of it.

According to what we said in Chapter 6, we also positioned on the indoor map, a set of path-points along our test path which are superimposed on the visual markers system, at a maximum distance from each other of about 2 meters. We can see the result of this process in Figure 8.4.

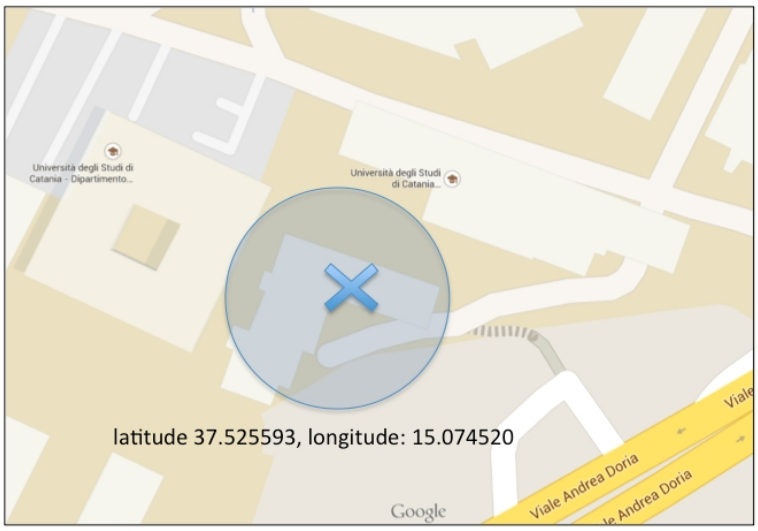

Figure 8.1: *GPS coordinates of the Building 13, academic campus in Catania.*

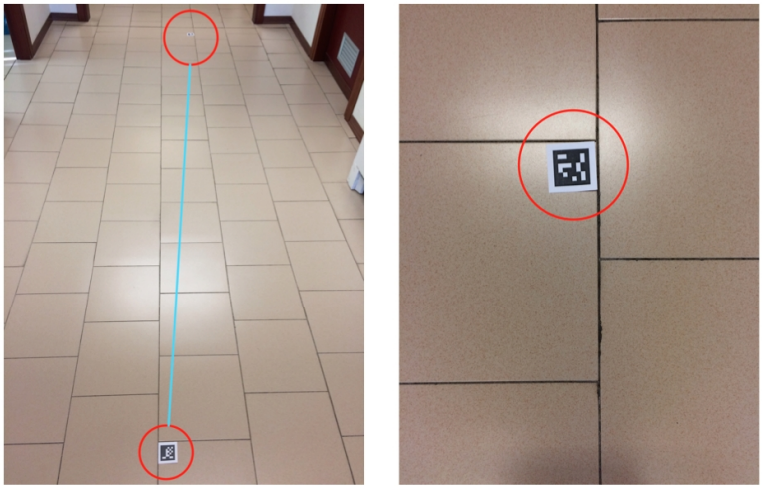

Figure 8.2: *Floor with the markers on top of it.*

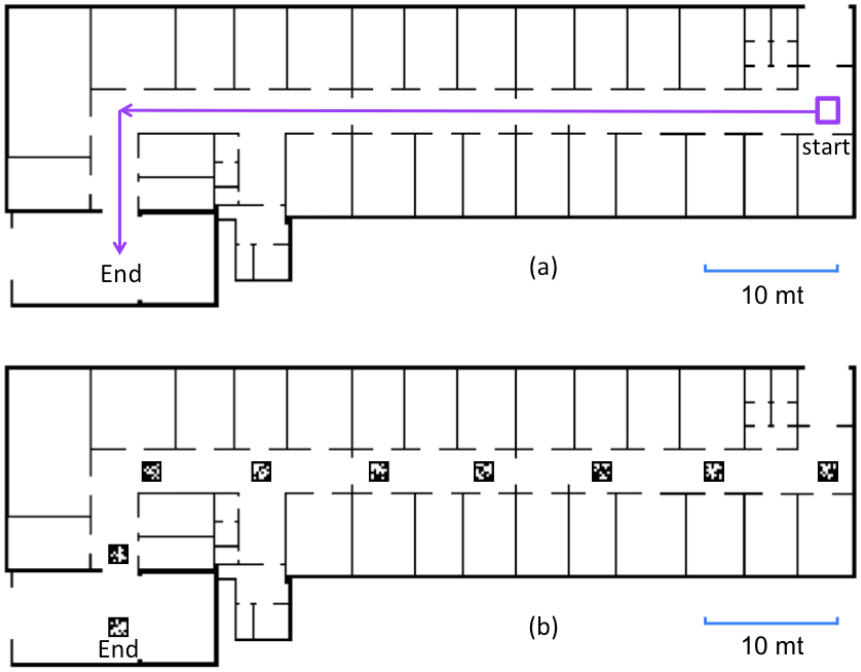

Figure 8.3: *indoor map of the building 13. (a) Test path. (b) Visual marker system deployed on top of the map.*

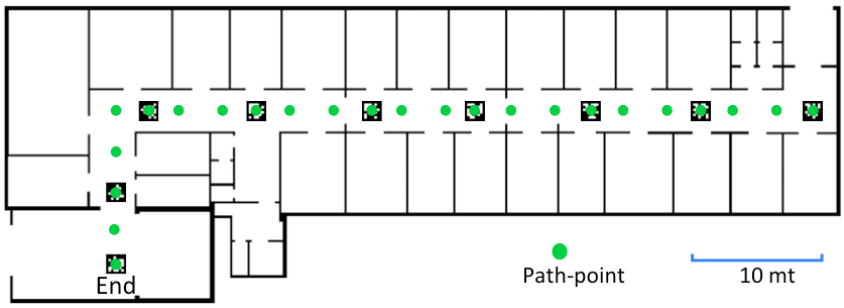

Figure 8.4: *Path-points placed on top of the map.*

8.1 IPS evaluation without Pedometer

Firstly we tested the prototype in the building 13, under good light conditions, using an iPhone 5S as testing device, and with the pedometer disabled. At the beginning we chose as size of the printed marker $(3.2 \times 3.2)cm^2$: we experimented that the detector sometimes failed in the detection of marker, especially when (1) there was a lot of light falling on it and (2) the user didn't walk slowly. We expected this: even if in the tests made on Chapter 5 (in a controlled environment) we saw that in good light conditions aprilTag works well for any tested size, in the real situations there are a lot of external variables that can reduce the performances. Another important factor that influenced a lot the performances was the quality of the printed marker and the material on which we printed it. Higher quality printed markers were easily detected than lower quality ones.

We made several tests in order to find the best trade-off between size of the marker, printing quality and performances. After these tests, we chose to print the marker in a standard quality and with a size of $(6.0 \times 6.0)cm^2$ in order to guarantee that almost all markers were correctly decoded. We didn't quantitatively measured the speed of the detection/decoding: even if this is an important factor in our prototype, due to the highly dynamic environments where the system were deployed, it didn't make any sense. Instead, we verified that the detection/decoding speed were less than 250 milliseconds under all light conditions, which is the value that gives us almost real time performances. The table 8.1 shows the qualitative results we obtained, by repeating the test path five times under different light conditions and by calculating the average percentages. In poor light conditions,

Light conditions	**Less than 250 ms**	**More than 250 ms**	**Not decoded**
Good Light	100%	0%	0%
Average Light	100%	0%	0%
Poor Light	73%	13%	14%

Table 8.1: *Qualitative evaluation of the AprilTag detection/decoding speed.*

14% of markers were not decoded while the decoding time was higher than 250 milliseconds for 13% of markers.

The Figures 8.5a, 8.5b, show the results of navigation: as we can see, in good and average light conditions the test path is followed perfectly. In poor light condition, three markers are not decoded: low lights and shadows projected on them make them undetectable.

8.2 evaluation of the IPS using both AprilTag markers and Pedometer

We repeated the previous tests (in good light conditions) by integrating the pedometer provided by the Apple CoreMotion framework which, according to what we experimented, is accurate enough (it lose about a couple of steps every one hundred steps, in our tests). The error increases a little when the number of steps grew. Every position estimated using the pedometer is reconducted to one of the path-points through a distance calculation; two kind of pedometer-related errors can happen:

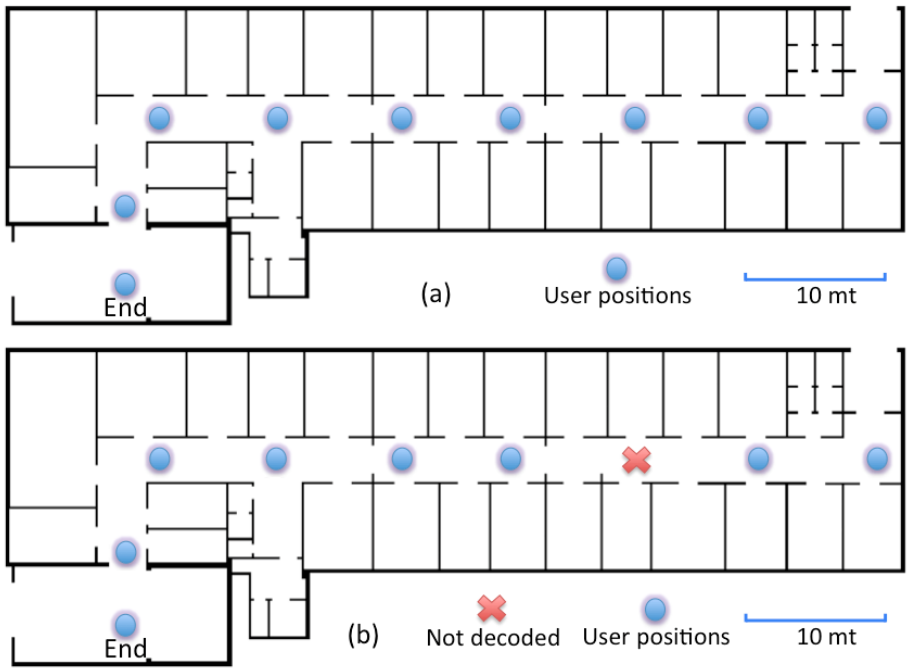

Figure 8.5: *(a) Good and average light conditions: all the markers along the test path are decoded. (b) Poor light conditions: some markers along the test path are not decoded*

1. due to the fact that the Apple pedometer APIs return the location fix more or less every 2.5 seconds, and depending on the density of the path-points, the speed of the user and his step length, the algorithm can miss one or more path-points between two markers.

2. considering two aprilTag markers, and considering the path-points between such markers, the algorithm fails if it places the user in the wrong path-point.

The Figure 8.6a shows the results of one of our tests. For the test we forced an almost constant step length ($100cm$), and a controlled, slow, way of walk. Three path-points are missed by using the Apple pedometer. This uncertainty happens especially at the beginning of the test path, and corresponds to the bootstrap of the pedometer. Anyway, the user is always placed in the correct path-point. The Figure 8.6b shows the results for an uncontrolled (not too fast) walks: We didn't forced the step length. In this case, we have three missed path-points and occasionally the algorithm places the user in the wrong path-point, due to the error in the step length.

The Table 8.2 resumes the qualitative results we obtained, by repeating the test path five times both by walking in a controlled and uncontrolled way.

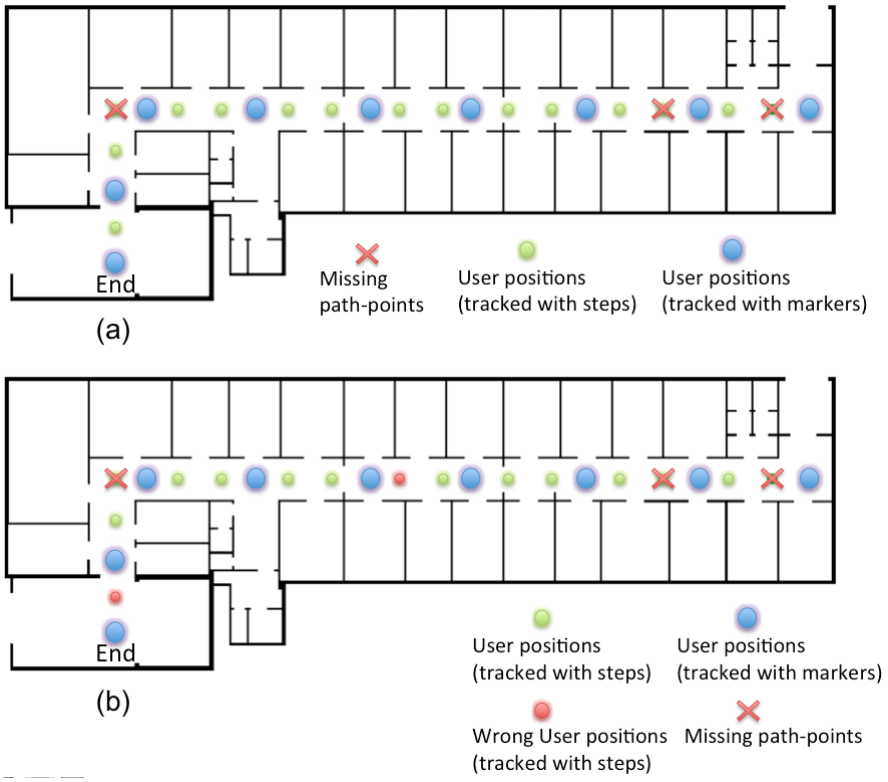

Figure 8.6: *(a) Controlled walking: three path-points are missed. (b) Uncontrolled walking: three path-points are missed and there are two wrong user positions.*

	Controlled Walks	**Uncontrolled Walks**
Missed path-points (%)	21%	19,3%
Wrong positions (%)	2,2%	14,2%

Table 8.2: *Missed path-points percentage and wrong positions percentage for a controlled and uncontrolled walk.*

CHAPTER

NINE

CONCLUSION AND FUTURE WORKS

In this dissertation, we addressed the problem of realizing an hybrid indoor localization/navigation system with three fundamental characteristics: scalability, low costs, and absence of any complex infrastructure to deploy into the environment. We achieved these objectives by deploying a visual markers system onto the floor (based on the idea that when the user launches the application to navigate inside the building, his camera is necessarily directed towards the floor) and by integrating into the system a pedometer.

In the first part of the dissertation, after an overview on the state of the art, we focused on choosing the best marker for an indoor navigation system with visual markers deployed onto the floor. The analysis led us to choose three visual markers which have features that match with our scenario: Vuforia marker, ArUco marker and AprilTag marker. A deep analysis of these markers has conducted us to choose for our prototype, AprilTag. The second part of the disser-

tation was focused on the description of our approach to the indoor localization. We presented a server-client architecture and described both the server side and the client side. Also, we gave some details about our client prototype. The approach can be successfully used in a lot of applicative scenarios such as in the hospitals to let the visitors move toward the correct medical department, in the airports to drive the user to the gates and to send him context-based offers, in the museum, etc.

9.1 Future Works

There is a great amount of enhancements to do in our work, both on the server-side and the client-side. On the server side we are planning to go beyond the server structure description and realize a fully functional server-side web application which can let us to test other indoor localization technologies with less effort. In fact, we experimented that a lot of time is wasted, during the development and testing of a new indoor localization technology, on managing everything related to the indoor maps such as downloading it, add/remove layers, considering floors and transitions between them, ecc. From the client point of view we are planning to increase the speed of the aprilTag detection/decoding algorithm and to reduce the marker size by exploiting some feature of our scenarios such as the uniform pattern of the floor, the fixed size of the marker inside the captured frame, the different probabilities for the marker to be in different parts of the frame. Also, we want to increase the modularity of the mobile application, in order to connect new technologies to it in an easier way, and try to use the bluetooth

low energy technology to (1) perform a raw localization when there is not a line of sight and (2) segment big areas in smaller areas in order to reuse the same set of markers. Also, we are working on the realization of a set of tools for rapid and accurate benchmarking of marker-based indoor localization technologies. Finally, from a pratical point of view, we are investigating the opportunity to embed the tags into the tiles.

BIBLIOGRAPHY

[1] Y. Liu, M. Dashti, and J. Zhang, "Indoor localization on mobile phone platforms using embedded inertial sensors," in *Positioning Navigation and Communication (WPNC), 2013 10th Workshop on*, pp. 1–5, IEEE, 2013.

[2] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, "A reliable and accurate indoor localization method using phone inertial sensors," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, (New York, NY, USA), pp. 421–430, ACM, 2012.

[3] S. Beauregard and H. Haas, "Pedestrian dead reckoning: A basis for personal positioning," in *Proceedings of the 3rd Workshop on Positioning, Navigation and Communication*, pp. 27–35, 2006.

[4] J. Bajo, J. F. De Paz, G. Villarrubia, and J. M. Corchado, "Self-organizing architecture for information fusion in distributed sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2015, 2015.

[5] A. Buchman and C. Lung, "Received signal strength based room level accuracy indoor localisation method," in *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pp. 103–108, Dec 2013.

[6] C. Fuchs, N. Aschenbruck, P. Martini, and M. Wieneke, "Indoor tracking for mission critical scenarios: A survey," *Pervasive Mob. Comput.*, vol. 7, pp. 1–15, Feb. 2011.

[7] J. Prieto, J. F. De Paz, G. Villarrubia, F. De la Prieta, and J. M. Corchado, "Self-organizing architecture for information fusion in distributed sensor networks," *International Journal of Distributed Sensor Networks, In Press.*

[8] D. Han, S. H. Jung, M. Lee, and G. Yoon, "Building a practical wi-fi-based indoor navigation system," *IEEE Pervasive Computing*, vol. 13, no. 2, pp. 72–79, 2014.

[9] Y. Liu, Q. Wang, J. Liu, and T. Wark, "Mcmc-based indoor localization with a smart phone and sparse wifi access points," *Pervasive Computing and Communications Workshops, IEEE International Conference on*, vol. 0, pp. 247–252, 2012.

[10] A. Mandal, C. V. Lopes, T. Givargis, A. Haghighat, R. Jurdak, and P. Baldi, "Beep: 3d indoor positioning using audible sound," in *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pp. 348–353, IEEE, 2005.

[11] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik, "Indoor localization without infrastructure using the acoustic background

spectrum," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, (New York, NY, USA), pp. 155–168, ACM, 2011.

[12] K. S. Pathapati Subbu, "Indoor localization using magnetic fields," 2011. AAI3529276.

[13] J. Haverinen and K. A., "A global self-localization technique utilizing local anomalies of the ambient magnetic field.," pp. 3142–3147, IEEE International Conference on Robotics and Automation, 2009. ISBN 978-1-4244-2789-5.

[14] J. HAVERINEN, "Utilizing magnetic field based navigation," July 11 2013. US Patent App. 13/734,365.

[15] A. Jovicic, J. Li, and T. Richardson, "Visible light communication: opportunities, challenges and the path to market," *Communications Magazine, IEEE*, vol. 51, no. 12, pp. 26–32, 2013.

[16] C. Danakis, M. Z. Afgani, G. Povey, I. Underwood, and H. Haas, "Using a CMOS camera sensor for visible light communication," in *Workshops Proceedings of the Global Communications Conference, GLOBECOM 2012, 3-7 December 2012, Anaheim, California, USA*, pp. 1244–1248, 2012.

[17] A. Chandgadkar and W. Knottenbelt, "An indoor navigation system for smartphones," 2013.

[18] S. Arias Bellot *et al.*, "Visual tag recognition for indoor positioning," 2011.

[19] S. Saito, A. Hiyama, T. Tanikawa, and M. Hirose, "Indoor marker-based localization using coded seamless pattern for interior decoration," in *Virtual Reality Conference, 2007. VR '07. IEEE*, pp. 67–74, March 2007.

[20] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: Unsupervised indoor localization," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, (New York, NY, USA), pp. 197–210, ACM, 2012.

[21] D. Zachariah and M. Jansson, "Fusing visual tags and inertial information for indoor navigation," in *2012 IEEE/ION Position Location And Navigation Symposium (PLANS)*, IEEE - ION Position Location and Navigation Symposium, pp. 535–540, IEEE, 2012. QC 20110412. Updated from manuscript to conference paper.

[22] I. Constandache, R. Choudhury, and I. Rhee, "Towards mobile phone localization without war-driving," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, March 2010.

[23] M. Meingast, C. Geyer, and S. Sastry, "Geometric models of rolling-shutter cameras," 2005.

[24] ByteLight, "Scalable indoor location." `https://www.bytelight.com`.

[25] Apple, "Taking core location indoors." `https://developer.apple.com/videos/play/wwdc2014-708/`.

[26] Apple, "Apple maps connect." https://mapsconnect.apple.com/.

[27] B. L. Ecklbauer, "A mobile positioning system for android based on visual markers," 2014.

[28] H. Kato, "Artoolkit." http://www.hitl.washington.edu/artoolkit/.

[29] Wikipedia, "Qr code — wikipedia, the free encyclopedia," 2015. https://en.wikipedia.org/w/index.php?title=QR_code&oldid=684423662.

[30] D. Wave, "Answers to your questions about qr-code." http://www.qrcode.com/en/.

[31] S. Siltanen and V. teknillinen tutkimuskeskus, *Theory and Applications of Marker-based Augmented Reality*. VTT science, 2012.

[32] A. Mohan, G. Woo, S. Hiura, Q. Smithwick, and R. Raskar, "Bokode: Imperceptible visual tags for camera based interaction from a distance," *ACM Trans. Graph.*, vol. 28, pp. 98:1–98:8, July 2009.

[33] Wikipedia, "Bokeh — wikipedia, the free encyclopedia," 2015. https://en.wikipedia.org/w/index.php?title=Bokeh&oldid=681441980.

[34] A. Longacre Jr and R. Hussey, "Two dimensional data encoding structure and symbology for use with optical readers," 1997.

[35] M. Fiala, "Artag, an improved marker system based on artoolkit," 2004.

[36] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, (Washington, DC, USA), pp. 590–596, IEEE Computer Society, 2005.

[37] "Goblin xna: A platform for 3d ar and vr research and education." `http://monet.cs.columbia.edu/projects/goblin/goblinXNA.htm`.

[38] L. Naimark and E. Foxlin, "Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker," in *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, ISMAR '02, (Washington, DC, USA), pp. 27–, IEEE Computer Society, 2002.

[39] Qualcomm, "Qualcomm vuforia," 2014.

[40] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[41] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, IEEE, May 2011.

[42] T. Ruoyu Fu, "Swiftyjson." https://github.com/SwiftyJSON/SwiftyJSON.

[43] Wikipedia, "Pathfinding — wikipedia, the free encyclopedia," 2015. [Online; accessed 28-November-2015].

[44] JasioWoo, "Pathfindingforobjc." https://github.com/wbcyclist/PathFindingForObjC.

[45] Alamofire, "Alamofire." https://github.com/Alamofire/Alamofire.

[46] R. Code, "Progresshud." https://github.com/relatedcode/ProgressHUD.